Procedural Biome System

S. Leguijt

Amsterdam University of Applied Sciences

April 06, 2026

Abstract

This project explores 3D procedural terrain generation with a focus on creating coherent environments of multiple biomes. The system is designed as a modular pipeline in which terrain is generated from scratch using noise-based input parameters. A world layout is first produced through the combination of multiple noise maps, after which biomes are assigned based on this underlying structure. Each biome defines distinct terrain characteristics and is used to generate localized height maps, which are subsequently combined into a final terrain mesh.

A key focus of this research is the challenge of blending biome-specific terrain into a unified and visually consistent result. Differences in biome configurations introduce difficulties at biome boundaries, often resulting in visible artifacts. This project investigates how different configurations and blending approaches influence the final terrain, highlighting both the potential and the limitations of current methods for achieving seamless biome transitions.

Methods

The associated project with this paper is developed in the Unity game engine, using the C# programming language. Unity version 2022.3.30f1 is used, with the Universal Rendering Pipeline (URP). No other third-party licences or assets are used during development of the project.

Introduction

Procedural terrain generation is widely used in games and simulations to efficiently create large and varied environments. By generating terrain algorithmically rather than manually, developers can produce expansive worlds with a high degree of variation while maintaining control over key parameters. A common approach within procedural generation is the use of biomes, where different regions of the world are assigned distinct terrain characteristics such as elevation, roughness, and overall structure.

Despite its advantages, biome-based terrain generation introduces significant challenges. While individual biomes can be generated with consistent and controlled features, combining multiple biomes into a single coherent world is considerably more complex. Each biome may rely on different noise configurations and parameter settings, which can lead to abrupt transitions at biome boundaries. These discontinuities often result in visible artifacts in the terrain mesh, reducing the overall realism and visual quality of the generated world.

This project investigates a modular terrain generation pipeline designed to construct a biome-based world in several distinct stages. First, a world layout is generated using a combination of noise maps, producing structured data that defines large-scale features of the terrain. Based on this layout, biomes are assigned to different regions, resulting in a biome weight map that determines biome distribution. Terrain is then generated per biome using biome-specific noise settings, after which the resulting terrain data is blended into a final height map and converted into a mesh for visualization.

The primary focus of this research is the interaction between these stages, with particular attention to the blending of biome-specific terrain. The process introduces a large number of configurable parameters, each influencing the outcome in different ways. As a result, achieving smooth and natural transitions between biomes remains a challenging task. Therefore, the central problem addressed in this project is how to generate and combine biome-based terrain in a way that produces visually coherent results while minimizing artifacts at biome boundaries.

This project focuses specifically on terrain shape generation using noise-based methods. Other aspects of world generation, such as flora, fauna, and ecosystem simulation, are not considered. Additionally, the system is limited to a predefined set of biomes and does not aim to model real-world biome distributions.

1 Pipeline overview

The terrain generation system is structured as a modular pipeline divided into multiple stages. These stages transform input data into output, which other stages use to perform their actions. To generate the world, we start with defining the relevant biomes. Every biome has its own specific terrain features and characteristics, which can be tweaked to produce different results.

Next, a layout generator creates the layout of the world using multiple inputs. For this paper, input parameters such as elevation, temperature, erosion and humidity are used to generate maps using fractal Perlin noise.

The following stage of the pipeline is assigning biomes to the provided layout of the world. Using a combination of the generated maps during the layout stage, rules where each biome may be present can be crafted and assigned to each biome, allowing for per biome setups. The stage produces a map, where each point in the map holds the weight of every possible biome, indicating what biome(s) are present at every point.

The generation stage uses the generated maps to produce a final terrain mesh. To do so, the biome weights map is used to blend the borders between biomes, allowing the terrain in these areas to blend despite the potential differences in terrain. The resulting map can then be combined with maps from the layout data, such as the elevation map, to create a final height map. Using the final height map, a mesh is constructed where each point of the map translates to the height of the corresponding mesh vertex. The result is a mesh with visible terrain features in distinct areas, with blending applied around the borders.

A final step of the pipeline can be used to visualise the resulting terrain mesh. Different methods can be used to visualise the terrain. In this project, we explore assigning a colour to each vertex of the mesh and using a simple shader to visualise the colours on the mesh.

The structure of the pipeline acts as a moderate approach to implement other techniques. Other algorithms to generate a layout, assign biomes, or generate terrain can be independently explored without impacting other stages of the pipeline.

2 Foundations

Procedurally generated terrain in games can use a variety of techniques and algorithms to produce different results. Still, there are some techniques and approaches that seem to be the preferred approach in the industry. Terrain is often created by combining noise maps with mesh generation, wherein a noise map serves as input data that is used to construct a mesh. This section covers both these foundational elements more in depth and introduces the theory to combine them, which is used in the stages of the earlier mentioned pipeline.

2.1 Meshes

Polygon meshes are used in 3D graphics to represent geometry, forming the foundation of representing objects [2]. They define the shape of an object by breaking its surface into smaller shapes, usually triangles or quads, or a general polygon [2].

Vertices

A mesh is created by combining a set of 3D points. These points are called vertices and are defined by their corresponding coordinate in space. Vertices can be seen as the corners and dots of a shape, where connecting them form edges between the vertices [4].

Faces

Combining a set of vertices forms a face, which plays a role in computing the normal direction and projecting light. A face consists of a minimum of three vertices, forming a triangle, a quad consisting of four vertices, or a general polygon with more vertices.

UV coordinates

To display textures on a mesh, the UV coordinates need to be laid out. This process, also known as UV mapping, comes down to making a 2D representation of your 3D mesh. Using a projection technique, the UV coordinates for a model can be produced, mapping a face of the mesh to a face of the UV map [3].

Generating meshes

In Unity, developers can construct their own meshes by creating a new Mesh instance and assigning the vertices and triangles manually. Unity uses a clockwise winding order to decide the front face of the triangle. So, to form the triangles, an array of integers should be supplied with the vertices that form each triangle in clockwise order. Additionally, the UVs of the mesh can be supplied if applying textures is desired. Besides the UV's, developers can also supply the mesh with an array of colours, where every element will be applied to a corresponding vertex. In our project, we use this approach to visualise the results of the generated terrain instead of textures.

2.2 Noise

Noise is a manifestation of randomness, like a series of random numbers in a line (1D) or a grid (2D) [1]. For games, noise functions can be used to generate a set of random numbers, which is core of procedurally generating something that look natural but varied [1]. Therefore, using noise with procedurally generation can be used to replace human decisions with random decisions that still create a believable world [11]. For generated terrain, noise is often used to create the shape of the terrain by translating a generated noise image into a heightmap. A heightmap is a grid of values, where each point is assigned a value from 0 to 1. These values can be visualised as an image or texture or seen as a description of elevation points for a piece of terrain, where lower values are near ground level, and higher values peaks [11].

Besides heightmaps, noise has many use cases in games; it can be used for audio, pictures and textures, modifying shapes, placing objects, creating maps and many more things [1].

2.2.1 Types of Noise

One of the main properties regarding noise is *frequency*. When visualising a 1D noise function, the frequency impacts the wavelength of the function. A low frequency function has a high wavelength, while a high frequency wave has a low wavelength [1]. If *frequency* controls the wavelength on the x-axis, *amplitude* controls the height of the waves on the y-axis.

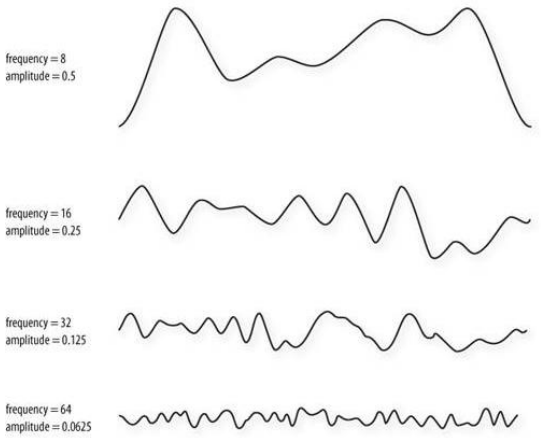


Fig. 1. Noise functions with varied frequencies and amplitudes

Colours can be used to label different types of noise, where the colour describes the frequency [1]. In the simplest form of noise, *white noise*, all frequencies contribute equally, which produces random values with no correlation between neighbouring points [11].

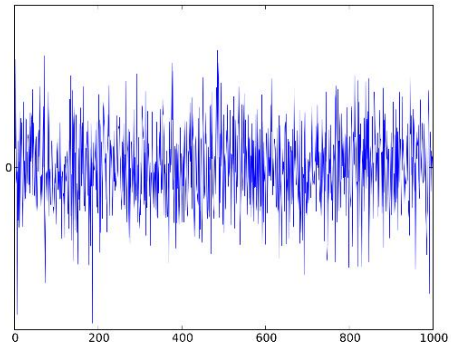


Fig. 2. 1D White noise represented in a graph [x]

Other types, such as *pink* or *red* (*Brownian) noise* emphasize lower-frequency noise, producing more gradual and smoother variations.

To generate terrain, smoother noise functions are used. Common examples include *Perlin noise* and *Simplex noise*, which produce continuous and gradual transitions between values. These properties make them well-suited for simulating natural terrain features. For terrain generation, Perlin and Simplex noise are often combined with *fractal noise,* also known as *fractal Brownian Motion (fBM) [5].*

Perlin noise

Perlin noise, developed by Ken Perlin, is a form of a *gradient noise* that takes in a set of random numbers and turns it into smooth and continuous connected values [11]. In 2D, this produces a grid of values where neighbours have similar values, which is essential for generating realistic terrain where features usually don't change abruptly [11].

The algorithm works by dividing a grid into cells and assigning a pseudo-random gradient vector to every corner of the grid. Then for every pixel in the cell, computing the dot product to every gradient vector produces a map of the grid. Because every pixel is influenced by four vectors, the resulting four maps can be blended using a smooth function to produce a single layer of Perlin noise [11].

Fig. 3. Perlin noise visualised as texture

Fractal Noise

Fractal Brownian Motion is the concept of layering multiple noise layers (*octaves*) on top of each other, where each *octave* increases the frequency while decreasing the amplitude of the noise function [6]. The technique of summing up multiple octaves of noise together with each different frequencies and amplitudes is called a fractal sum [6]. This is especially useful for terrain generation, because multiple octaves allow for adding finer details to larger generated features. In combination with Perlin noise, this can create good looking terrain features that can be easily adjusted for different results.

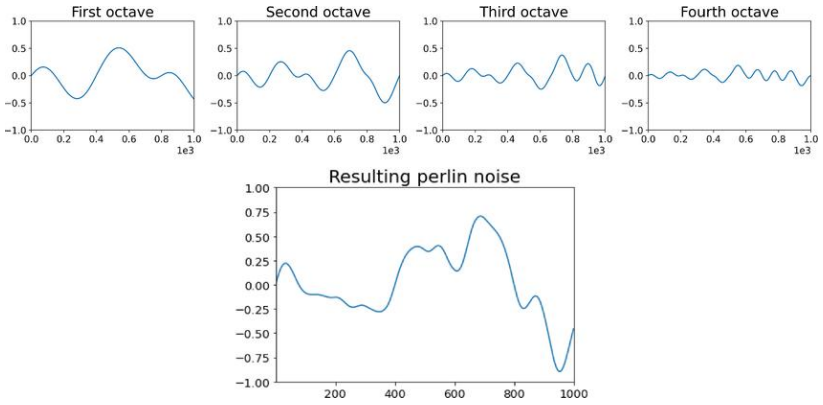


Fig. 4. Fractal sum result of multiple octaves of Perlin noise

2.2.2 Noise applied

There are many ways to generate noise [1]. Perlin noise can be used to generate smooth noise values that are somewhat coherent. Using fBM we can create more detailed looking noise. For generating terrain, we can use these techniques to generate a 2D heightmap. This heightmap is then used to create a mesh, where every value in the heightmap directly corresponds to a vertex's height in the mesh. To generate this heightmap, several settings can be used in the noise function to generate the desired results. An overview of the settings used in our project can be seen in figure x below.

In Unity, we make use of the built-in 2D Perlin noise function. This function generates a plane of wave-like values (Perlin noise) and returns a value between (0,0) and (1.0) from the plane based on the given coordinate input. [7]. Important to know is that this function is deterministic; given the same inputs, the output will always return the same value. This is crucial for procedural generation, because this allows the generated output to be fully replicable given the same *seed,* which is used in games like Minecraft and No Man's Sky to generate the same worlds multiple times.

Parameter	Description	Effect on terrain
Seed (int)	Integer value used to initialise the pseudo-random number generator. Can be randomized.	Different seeds produce entirely different areas of the world, without affecting the shape of the terrain.
Octaves (int)	Number of octaves that are used to generate the noise map.	More octaves add more detail to the terrain; fewer octaves produce more smooth looking terrain.
Persistence (float)	Controls how much the amplitude decreases per octave. For terrain, this value is usually around (0.5), meaning the amplitude decreases by half every octave	Lower values will result in each octave adding few details, while higher values will result in each octave having more influence on the resulting terrain.
Lacunarity (float)	Controls how much the frequency increases per octave. For terrain, this value is usually around (2.0), meaning the frequency increases by 2 every octave.	Lower values will add fewer terrain details with more gradual transitions. Higher values will increase the details and make terrain look more "rough".
Scale (float)	Controls how much the sampled noise is zoomed in or out.	Higher values is more zoomed in, creating larger terrain features. Smaller values results in more noisy terrain with more detail.
Offset (Vector2)	Controls where the 2D noise is sampled from. By randomising the offset, a different area of the Perlin noise plane will be sampled from.	Moves the terrain pattern without changing the structure, can be used to "scroll" through the generated map.

Fig.5. Overview of used noise settings

Creating a mesh using a noise generated heightmap with above input parameters can produce some interesting looking terrain, as shown in Figure X below. Changes in octaves, lacunarity and persistence affect the terrain itself, while changing the scale and offset allows for moving around in the noise plane, sampling different locations and zooming in/out.

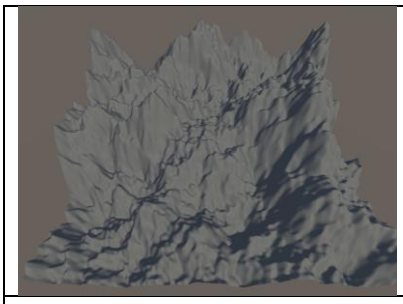
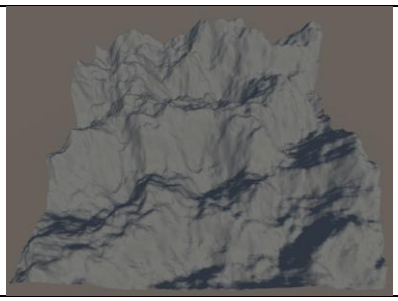
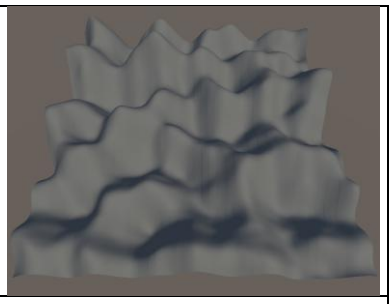
		
Scale: 100 Octaves: 5 Persistence: 0.5 Lacunarity: 2.0	Scale: 75 Octaves: 10 Persistence: 0.25 Lacunarity: 3	Scale: 50 Octaves: 3 Persistence: 0.1 Lacunarity: 2

Fig. 6. Meshes constructed from heightmaps with varied noise settings

Although the mentioned noise settings are used to shape the terrain, there is little control how the noise is distributed. This results in the generated maps having somewhat of a uniform distribution of values, where terrain features are visible but not distinctive enough in many cases. Details to the terrain can be added, as well as increasing peaks or valleys, but there is little control on the overall distribution of values, often resulting in adding details over the whole terrain instead of only affecting certain areas.

To introduce more dramatic terrain features, a remapping step can be applied after generating the noise map, where we introduce a set of spline points or a curve [8]. In Unity, an Animation Curve can be used. The curve will take in the values from the generated noise maps and return a mapped value on the curve, which can be adjusted to the desired results. This allows developers to control how the height of the terrain is distributed and can introduce a lot of new terrain features. For example, when flattening certain ranges of values in the curve, we can introduce flat areas in our terrain mesh. Alternatively, creating a steeper portion of the curve allows for exaggerating peaks. In figure x, the meshes are generated with the same seed and settings as in figure z, remapped with an animation curve to modify the resulting heightmap that is used for the mesh.

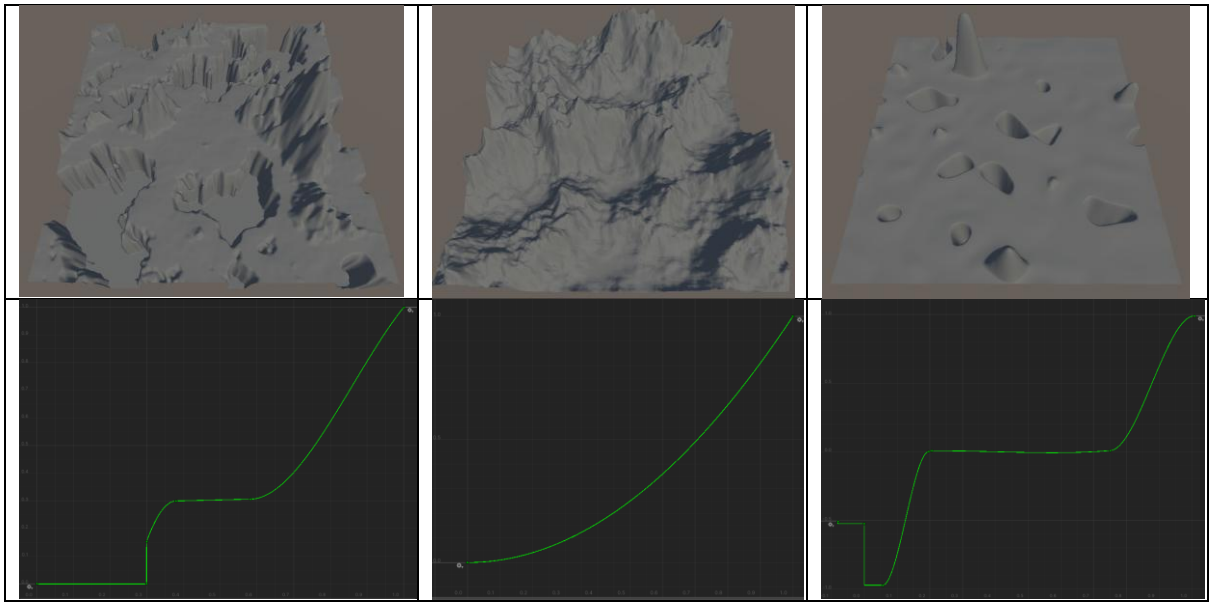


Fig. 7. Meshes constructed from noise heightmaps with remapping step

By combining Perlin noise and Fractal Brownian Motion, adjusting the input values and modifying the resulting output values, we can create many kinds of terrain maps with very different and distinct features. A potential limitation with this approach, however, is having to change and test a lot of different parameters. Because many small changes affect the end results, developers should be mindful of changing values abruptly and leaving values undocumented.

3 Pipeline

This chapter describes the implementations and results of the project, separated into pipeline steps, covering the concepts of biomes, generating biome-specific terrain and creating a coherent world with the generated elements.

3.1 Biome definitions

Before biomes can be used within the generation pipeline, they must first be clearly defined. In games like Minecraft and Terraria, biomes describe an environmental region with unique geography, plants and other characteristics [x]. In procedural terrain generation, a common approach is the *terrain-first* method when it comes to biomes. This approach starts with generating the terrain, after which biomes are assigned based on the resulting features. In many cases, height ranges from the resulting terrain are used to assign different biomes; such as water at low elevations, followed by sand, grass, hills, mountains and snow at progressively higher levels [x]. This can be practical for many terrain systems but may feel lacklustre when wanting to generate discrete biomes.

Another approach is *biome-first* assignation. In this process, biomes in a world are assigned before the terrain is generated, which allows the terrain to be generated based on each biome's characteristics. This allows for more control on each biome's features, where the placement of a biome does not decide the overall look of the biome.

The system presented in this project follows a biome-first approach. Biomes are defined primarily by their terrain characteristics, without additional features like flora and fauna, weather systems, or other real-life simulated factors. Each biome is associated with its own set of noise parameters that are used to generate its terrain. The diagram below gives a better understanding of each biome in our system and their desired look.

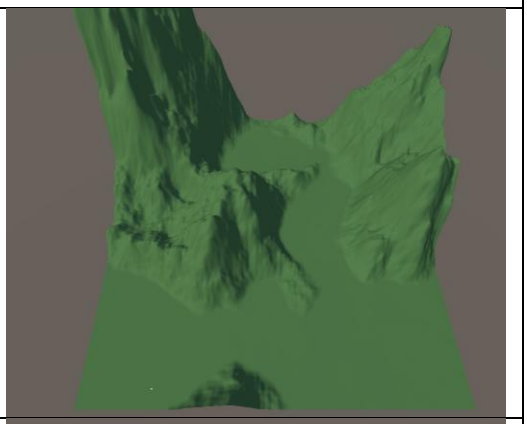
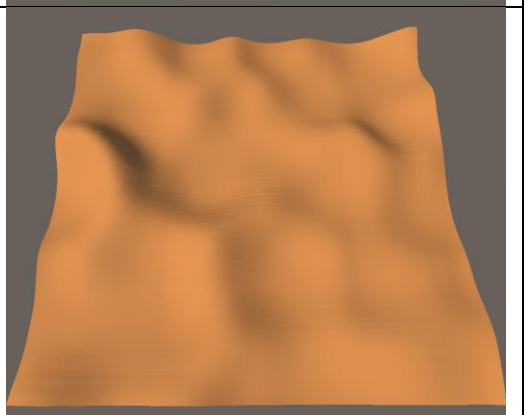
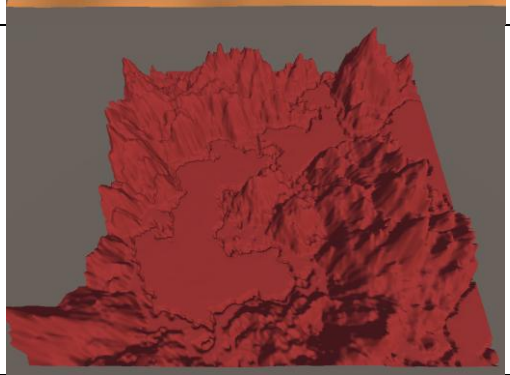
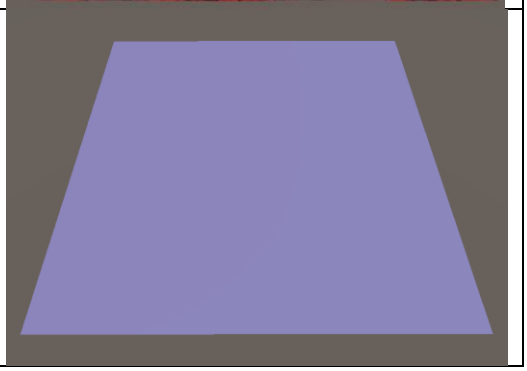
Biome	Terrain features	
Mountains	- Tall mountains with long slopes - Large height difference between lowest and highest points - Wide and large open areas near ground level, surrounded by large mountains	
Desert	- Smooth, stretched hills - Moderate height difference between low and high points - Few flat areas, many hills and small valleys.	
Volcanic	- Rough terrain - Moderate high mountains with many rocky details, and sharp peaks - Narrow flat areas visible around rough terrain	
Plains	- Completely flat terrain	

Fig.9. Overview of biomes

3.2 Layout generation

The next step in the pipeline is generating a layout of the world. In our system, this step is only responsible for producing data that further pipeline steps can use to generate their terrain and is completely independent of biomes.

As mentioned in previous sections, noise can be a great candidate to produce a somewhat random generated area. To produce the world layout, we will generate multiple noise maps, each with their own values.

To generate the noise maps, a similar approach to that of Minecraft is used. Minecraft uses a combination of five different noise maps: continentalness, erosion, temperature, humidity and peaks & valleys [8]. In our system, we use similar but not all maps as well; erosion, temperature, humidity and elevation maps are used to generate the layout.

Each of these maps represents a different property of the world. For example, elevation describes large-scale height variation, temperature can be used to distinguish between warm and cold regions, humidity indicates moisture levels, and erosion influences how smooth or rough terrain appears. By themselves, these maps provide limited information, however, when combined, they form a richer and more natural representation of the world's structure. Generating these maps using noise means every iteration can be different while still maintaining organic shapes, which is ideal for generating a natural layout of a simulated world.

These maps provide the core data for the world in this system, which is used to assign biomes and generate terrain in the next pipeline step. Other systems may want to use different maps to describe their world layout, which can even be done with different techniques and algorithms besides noise, such as Voronoi diagrams or Cellular Automata.

Examples of the generated maps can be seen in the figures below. Values range from 0-1, with 0 being black and 1 being a distinct colour.

Fig.10. Elevation map

Fig.11. Temperature map

Fig.12. Humidity map

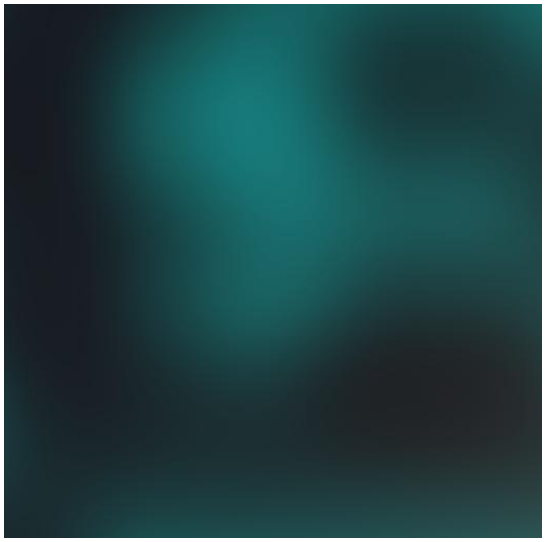


Fig.13. Erosion map

3.3 World generation

This section describes the generation of the world using the provided layout data. First, methods to assign biomes to the world are explored. After assigning biomes, the terrain mesh is generated by blending multiple biomes together smoothly.

3.3.1 Assigning biomes

Following the generation of the layout with multiple data maps, the next step in the pipeline is to use this data to assign the earlier defined biomes. To do so, we use another map which represents the weight of each biome at any given point. The result is a map of biome weights, where every position in the map contains data on the influence of each biome. This map is then used by the terrain generation step to create the final mesh; this will be covered in the next section.

To assign a biome and its weight to a given point, the mentioned maps from the layout are used. For every biome, we can conclude simple rules to describe how present they are depending on different areas of temperature, humidity, elevation and erosion; Desert biomes appear in lower elevation areas where temperature is high, while mountain biomes appear in higher elevation areas where temperature is low.

To get the weight of a biome at a given position in the map, sample the different maps from the layout and use those values as inputs for the rule calculations. Every rule returns a value based on the given inputs, which results in a single weight when added up together. This weight can be further adjusted by powering the result to a *blending factor* to control how strongly biomes overlap and blend. A higher blending factor results in prominent biomes dominating, while a lower factor results in biomes being more mixed together. After computing the weights for all biomes at a given point, it is important to normalise the weights, so they add up to 1. This ensures the relative influence of each biome can be computed from the weights directly and is essential for blending the terrain smoothly.
By using a weighted approach to assign biomes to the generated layout, hard biome borders are prevented. This ensures the final heightmap can be blended smoothly using the weights, gradually moving between different biome terrain. Additionally, this map can be used to visualise the generated terrain and adjust the layout maps if needed.

Fig. 14, 15, 16. Biome weight maps visualised

3.3.2 Terrain generation

The most important step in the pipeline is generating the terrain itself. By following the pipeline, most of the required data is already present, which leaves this step to do the following:

- Generate a heightmap for every biome
- Combine the heightmaps of every biome by blending them together based on their weight to create a final heightmap
- Use the final heightmap to generate a terrain mesh

In the first step of the pipeline, each biome has been configured with their own set of noise parameters to generate biome-specific terrain. Using these parameters to generate a heightmap for every biome then becomes trivial. Important to incorporate in this step is a *height multiplier* per biome, which multiplies the height value from the generated noise map by the given value. This is essential to ensure each biome's terrain amplitude is not multiplied by a global or average value.

Next, the individual biome heightmaps are combined using the biome weights map. By using the weight at every point in the map, we can conclude the contribution of every biome at any given point in the map. Instead of applying a simple average, the system performs a weighted blending operation, where biome contributions are scaled based on their relative influence. Additionally, a minimum weight threshold is applied, ensuring that only sufficiently dominant biomes contribute significantly to the final height. This helps reduce noise from minor biome influences and improves the clarity of biome regions.

The final height of a point can further be adjusted with a few other techniques. By supplying a baseline per biome, we can adjust relative positioning of biomes next to each other. This allows for biomes at higher elevation to appear from a higher point in the mesh and vice versa. Additionally, each biome's contribution to the height of the point can have a threshold behind a given influence, allowing developers to control how much a biome's weight and influence should be before contributing to the height of a point.

After blending the individual biome heightmaps into a single heightmap, the result is added onto the generated elevation map from the layout. This allows overall elevation of biomes to be preserved while layering biome specific details on top. The final heightmap is then used to construct a mesh, where each vertex corresponds directly to a value in the heightmap.

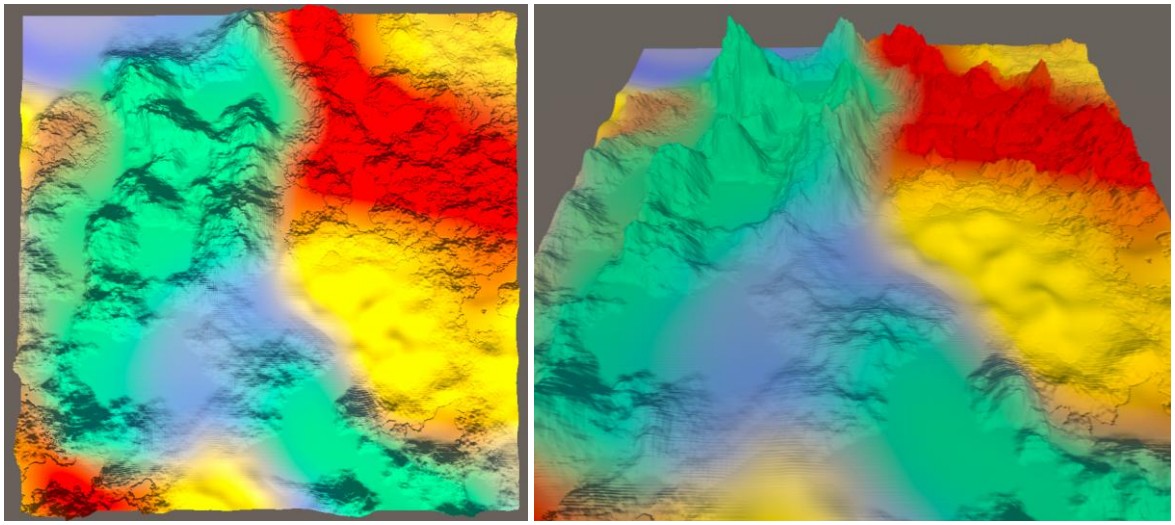


Fig. 17, 18. Resulting generated terrain from different angles

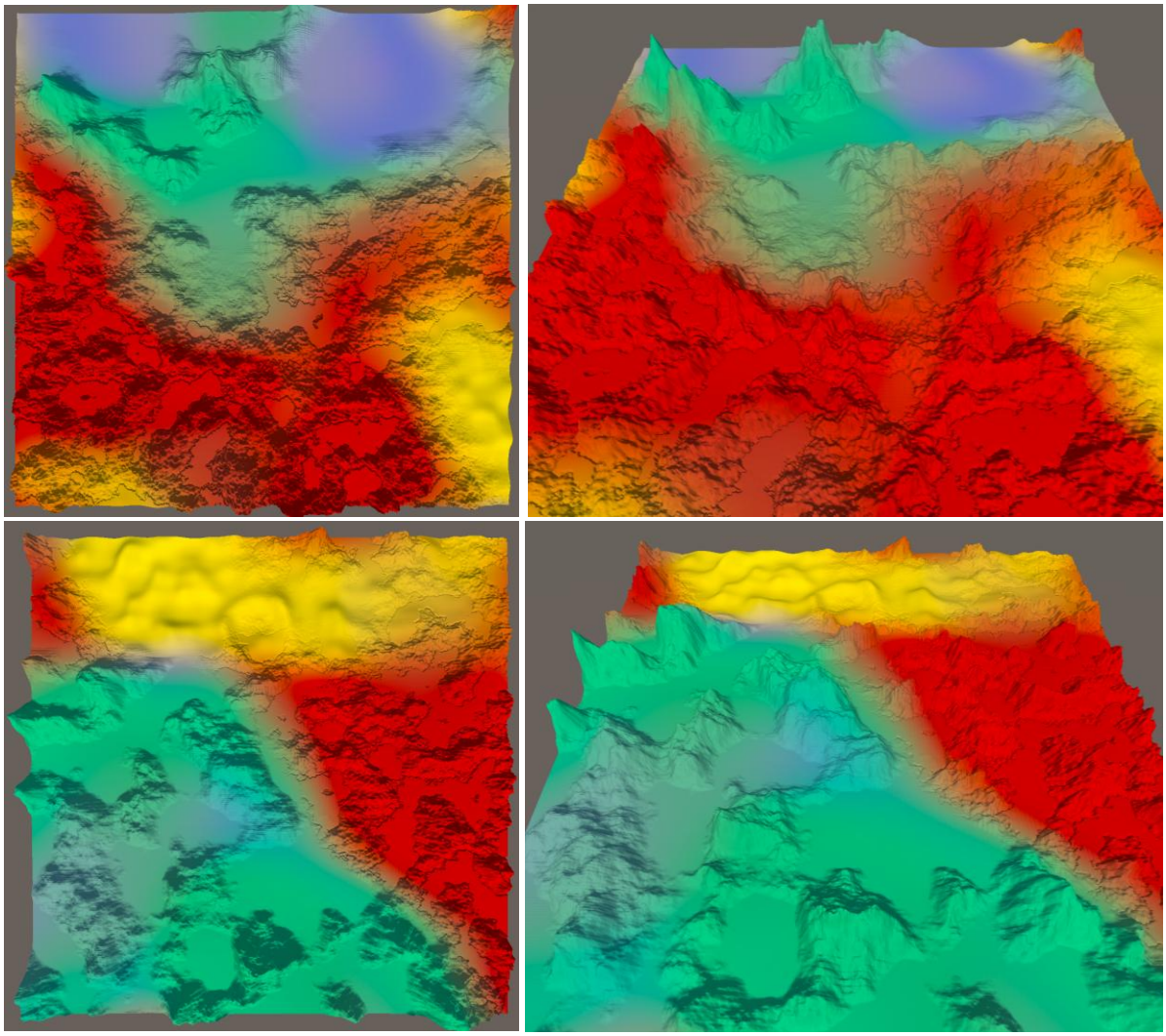


Fig. 17, 18, 19, 20. More generated terrain from different angles

3.4 Visualisation

Procedurally generated terrain can be visualised in many ways. Often, shaders are used to blend between different textures and colours. In this system, we use a very simple shader to render the colours of each vertex in the terrain mesh. The colour of each vertex is assigned in the generation stage after blending biomes. This way, many different colours can be used to visualise different values of the terrain. Each vertex's prominent biome can be visualised, a blend of colours based on biome weights, grayscale texture based on biome purity and many more ways. In our system, we configure multiple ways to quickly change the colours of the vertices.

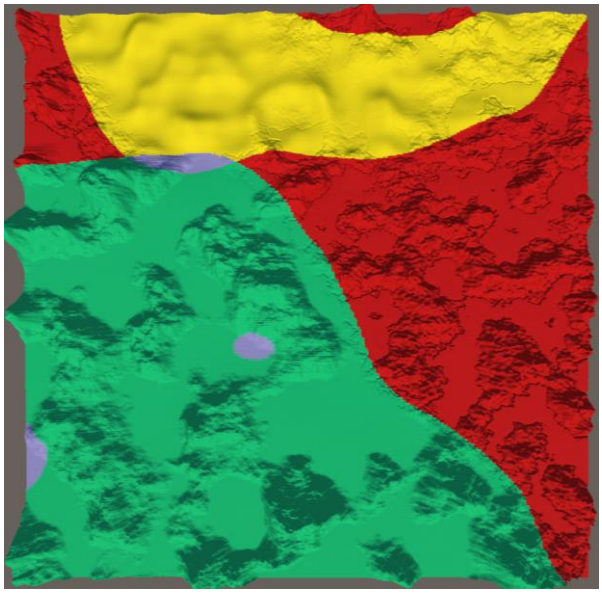


Fig. 21. Terrain with most prominent biome color

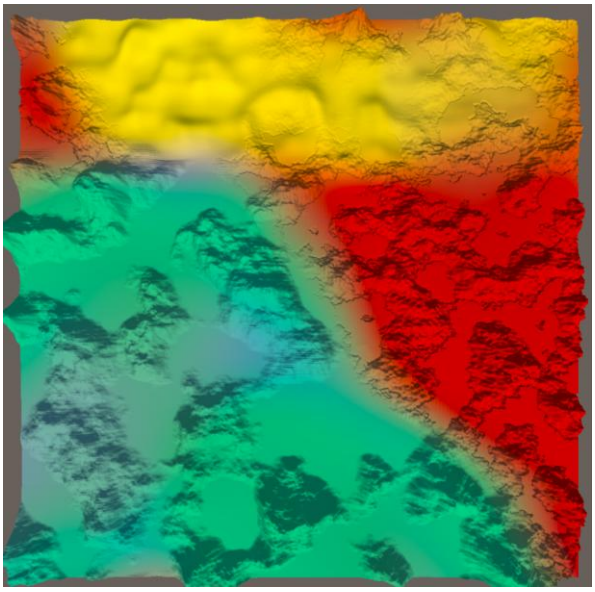


Fig. 22. Terrain with biome colours based on weights

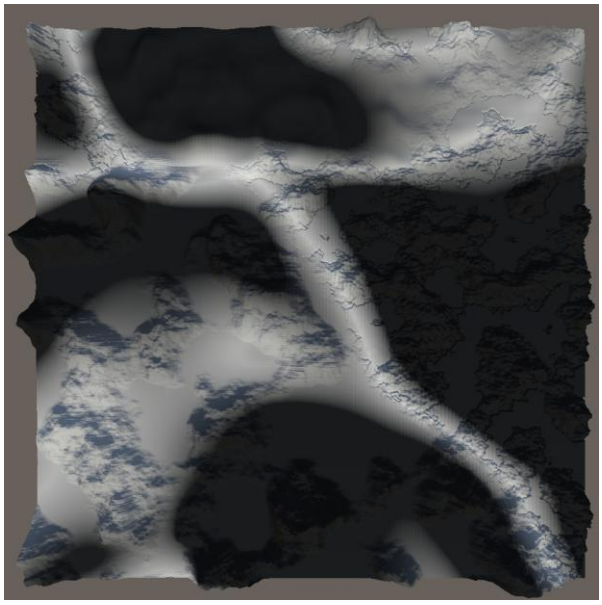


Fig. 23. Terrain with blending areas highlighted

Fig. 24. Terrain with biome colors where weight is above 75%

4 Conclusion

This project investigates how procedural generation techniques can be used to create coherent 3D terrain consisting of multiple pre-defined biomes. The paper describes a pipeline to follow to create terrain a terrain mesh from scratch using a combination of procedurally generated noise maps. By separating stages for defining biomes generating a world layout, assigning biomes and terrain generation, individual steps can be replaced with relative ease to experiment with other techniques and algorithms.

The use of Perlin noise in combination with Fractal Bownian Motion proved to be a solid solution to generate varied, controlled 3D terrain. Extending this concept with biome specific settings allowed for the generated terrain to be fully adjustable in relation to a given context, which is valuable for further projects.

The primary challenge regarding this project was combining the different noise maps to create smooth transitions between the borders. Although not entirely achieved with the tested input parameters, we can conclude this approach is usable to achieve the desired results.

References

[1] A. Patel, “Noise Functions and Map Generation,” 31 August 2013. [Online]. Available: https://www.redblobgames.com/articles/noise/introduction.html#rainbow. [Accessed 16 April 2026].

[2] “Introduction to Polygon Meshes,” [Online]. Available: https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-polygon-mesh/introduction.html#:~:text=Polygon%20meshes%20or%20meshes%20for,3D%20vertices%20(Figure%201). [Accessed 15 April 2026].

[3] Pluralsight Content Team, “Understanding UVs: Love or Hate Them, They're Essential,” 01 November 2022. [Online]. Available: https://www.pluralsight.com/resources/blog/software-development/understanding-uvs-love-them-or-hate-them-theyre-essential-to-know. [Accessed 16 April 2026].

[4] S. Prasanth, “From Code to Shape: A Beginner’s Guide to Unity Mesh Generation,” 03 May 2025. [Online]. Available: https://sivakumar-prasanth.medium.com/from-code-to-shape-a-beginners-guide-to-unity-mesh-generation-225cc994e5a6. [Accessed 15 April 2026].

[5] P. G. Vivo and J. Lowe, The Book of Shaders, 2015.

[6] “Value Noise and Procedural Patterns,” [Online]. Available: https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/procedural-patterns-noise-part-1/simple-pattern-examples.html. [Accessed 28 March 2026].

[7] Unity Technologies, “Mathf.PerlinNoise,” [Online]. Available: https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html. [Accessed 26 March 2026].

[8] H. Kniberg, “Minecraft terrain generation in a nutshell,” 06 February 2022. [Online]. Available: https://www.youtube.com/watch?v=CSa5O6knuwI&t=1413s. [Accessed 29 March 2026].

[9] “All Biomes in Minecraft,” [Online]. Available: https://help.minecraft.net/hc/en-us/articles/360046470431-All-Biomes-in-Minecraft. [Accessed 01 April 2026].

[10] S. Lague, “Procedural Landmass Generation,” 13 February 2016. [Online]. Available: https://www.youtube.com/watch?v=RDQK1_SWFuc&list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3&index=5. [Accessed 23 March 2026].

[11] The Taylor Series, “How to turn a few Numbers into Worlds (Fractal Perlin Noise),” 16 August 2023. [Online]. Available: https://www.youtube.com/watch?v=ZsEnnB2wrbI&list=PL4-R075-4oR8WflHWDs79uFqWSzeikc44. [Accessed 27 March 2026].