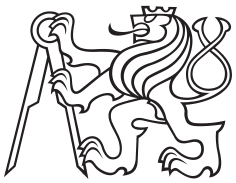
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Procedural Generation of Outdoor Scenes

Ondřej Kyzr

Supervisor: doc. Ing. Jiří Bittner, Ph.D.
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kyzr Ondřej** Personal ID number: **498969**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Specialisation: **Computer Games and Graphics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Procedural generation of outdoor scenes

Bachelor’s thesis title in Czech:

Procedurální generování venkovních scén

Guidelines:

Bibliography / sources:

[1] Smelik, Ruben M., et al. 'A survey on procedural modeling for virtual worlds.' Computer Graphics Forum. Vol. 33. No. 6. 2014.
[2] Noor Shaker, Julian Togelius, Mark J. Nelson. Procedural Content Generation in Games. Springer International Publishing. 2016.
[3] Hendrikx, Mark et al. Procedural Content Generation for Games: A Survey. In: ACM Trans. Multimedia Comput. Commun. Appl. 9.
[4] Petr Brachaczek. Modely 3D scén pro jízdní simulátor. Bakalářská práce, ČVUT FEL 2017.
[5] Jana Kejvalová. Procedurální generování 3D modelu dle mapových podkladů. Diplomová práce, ČVUT FEL 2019.
[6] Jan Kutálek. Procedurální generování prostředí pro videohry. Bakalářská práce, ČVUT FEL 2021.

Name and workplace of bachelor’s thesis supervisor:

doc. Ing. Jiří Bittner, Ph.D. Department of Computer Graphics and Interaction

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **17.02.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Ing. Jiří Bittner, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I want to thank my supervisor, doc. Ing. Jiří Bittner, Ph.D., for proposing this engaging topic and letting me change it to my liking. I would also like to thank my fiancée for helping me with grammar, spellchecking, and motivation.

Declaration

I declare that this work was written and implemented by me, and I have cited all sources I have used or was inspired by in the bibliography.

Prague, May 20., 2023

.....

Abstract

This bachelor thesis describes an implementation of an easy-to-use tool for the game engine Unity, which can procedurally generate outdoor scenes. The tool is split into several generators so that the user can choose the features of the terrain. These features include roads, human-made paths, hydraulic erosion, water bodies, and rivers. The resulting terrain is easily editable with many parameters and tools.

Keywords: Procedural generation, Terrain, Landscape, Hydraulic erosion, Rivers, Road, Paths

Supervisor: doc. Ing. Jiří Bittner, Ph.D.
Fakulta elektrotechnická,
Karlovo nám. 13,
12000 Praha 2

Abstrakt

Tato bakalářská práce popisuje implementaci snadno použitelného nástroje pro herní engine Unity, který dokáže procedurálně generovat venkovní scény. Nástroj je rozdělen na jednotlivé generátory, aby si mohl uživatel vybrat, jaké vlastnosti terén bude mít. Mezi ně patří silnice, lidmi vytvořené cesty, vodní eroze terénu, vodní tělesa a řeky. Výsledný terén je snadno upravitelný pomocí mnoha parametrů a nástrojů.

Klíčová slova: Procedurální generování, Terén, Krajina, Vodní eroze, Řeky, Silnice, Cesty

Překlad názvu: Procedurální generování venkovních scén

Contents

Figures

Tables

Chapter 1
Introduction

1.1 Idea

My primary idea was to create an accessible tool for Unity where a user can create realistic-looking outdoor terrain. The terrain should feature roads, winding paths made by humans, grass, rivers, and lakes. The user should be able to choose the look of the terrain so that different biomes (mountains, meadows, etc.) can be generated. The final terrain should resemble the European countryside, so I will not focus on the tool's ability to generate exotic biomes.

The user should be able to change the terrain to their liking using different parameters and tools. Each generation part (e.g., base terrain, roads, water, etc.) should have its own separate generator, allowing the user to choose the presence of these features and quickly change or completely revert them.

An example of procedurally generated terrain from the game No Man's Sky can be seen in Figure 1.1. This game features procedurally generated plants, animals, planets, and even solar systems. Another example can be seen in Figure 1.2, from the game Cube World. This game uses voxels, which are small cubes, to represent and generate the terrain in a similar way to the popular game Minecraft [25].

1.2 Tools Used

For this Bachelor Thesis, I decided to use the Unity game engine because it is one of the most used game engines in the industry and is very easy to learn. It is well documented, and many tutorials exist, which will make the implementation easier. Personally, I have the most experience with Unity when it comes to game engines, which was the main choice factor.

Figure 1.1: A screenshot of a procedurally generated planet from the game No Man's Sky [24].

Figure 1.2: An example of a procedurally generated terrain using voxels from the video game Cube World [26].

Chapter 2
Related Work

Terrain generation has been covered in a lot of books and articles. The whole process and its many variations will be covered in this chapter. When it comes to the generators of different features, I did not draw from many sources because most of the things I wanted to implement were my ideas, with a few exceptions. I found very good tutorials and sources tackling most of my implementation issues.

2.1 Terrain Generation

Landscapes in video games are usually represented either as a DEM (Digital Elevation Model) or a heightfield, in which we store the heights of the terrain. Some games utilize a 3D collection of data to represent layered terrain and caves[2]. The vertices of the grid are equally distributed, and each height of a vertex is procedurally generated. There are several methods for generating the heights of points, such as midpoint displacement, random generation, and noise generators.

The landscape can be separated into smaller parts called chunks. This proves to be quite helpful, especially when it comes to performance. Lague [3], in his Youtube series about landmass generation, successfully uses chunks for displaying only the relevant parts of the terrain. Chunks closest to the player character are the most detailed, and with distance, the chunks get less and less detailed until they stop appearing.

2.1.1 Midpoint Displacement (fractal terrain)

Early algorithms used a method called midpoint displacement. This technique is based on surface subdivision. We start with one quad and divide it into four quads by adding vertices halfway on each edge and one in the middle. The positions of these vertices are calculated as the average of the neighboring vertices plus a random offset. Using this method, an infinitely detailed landscape can be created. However, one cannot influence where features of the terrain are generated.[2] The subdivision process can be seen in Figure 2.1.

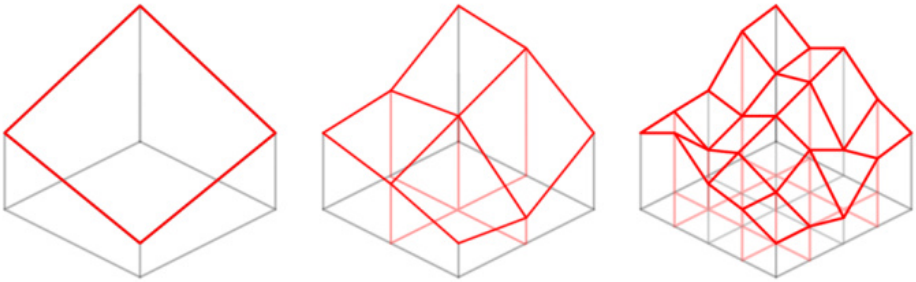


Figure 2.1: Midpoint Displacement algorithm[8].

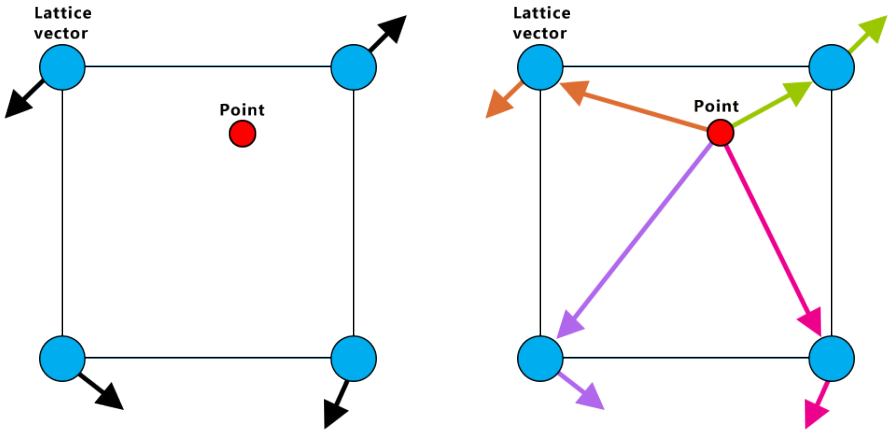


Figure 2.2: Algorithm described by subsection 2.1.3. The first picture shows the base setup with lattice vectors and the point at which we want to generate the height. The second picture shows the pairs of vectors that are used for the dot product.

2.1.2 Random Terrain

One straightforward method is generating vertices' heights randomly [7]. This, however, does not create a believable terrain. It creates random spikes and pits that do not correlate with each other, and even if we interpolate the points, the result is not convincing. However, noise generators are based on this method.

2.1.3 Noise Generators

Noise generators take the approach of generating random numbers, and they use them more coherently. Perlin noise, created by Ken Perlin in 1982 for the movie Tron, is one of the most used noise generators. It uses random numbers to generate a lattice of gradient vectors instead of heights.

A height of a point is calculated by the dot product between a vector pointing from the current location to the lattice point and the gradient vector

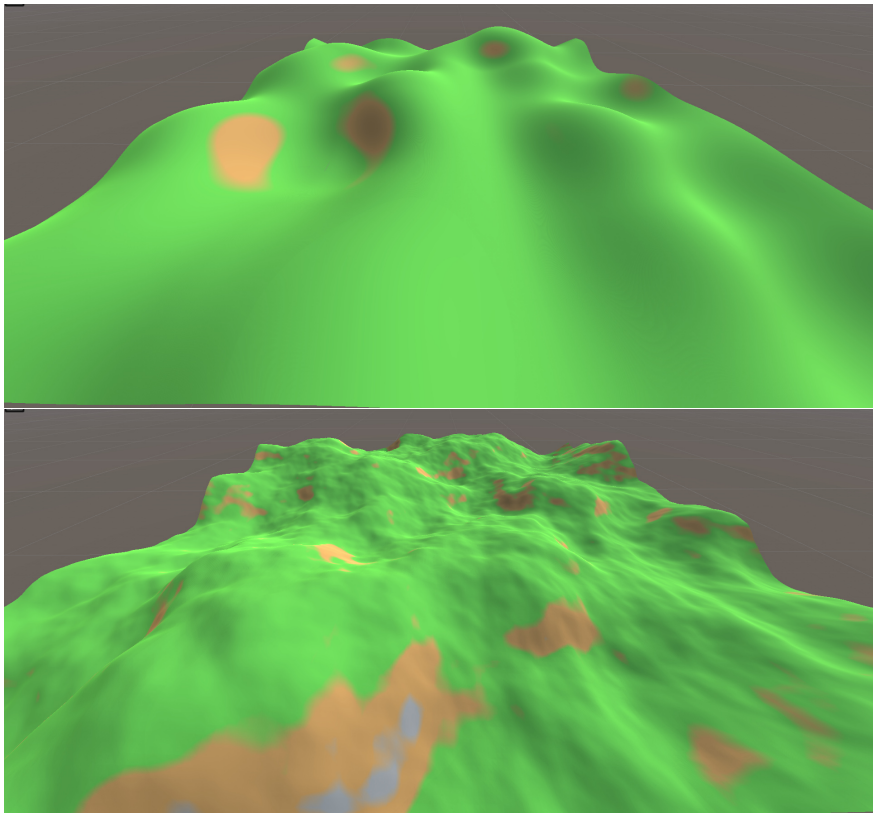


Figure 2.3: Terrain generated with my tool using one octave versus seven octaves.

of the lattice point. We repeat the process for each of the four closest lattice points. Now we have four height values. We interpolate these values based on the distance from the current point to their respective lattice point. This gives us an extra level of smoothness. The process is visualized in Figure 2.2.

One level (octave) of noise creates a terrain that is almost too smooth. This problem can be fixed by adding multiple octaves of noise with diminishing scale and influence [7]. An example using my tool can be seen in Figure 2.3.

Ken Perlin also created Simplex noise, a faster and better version of Perlin noise. It is based on simplexes, shapes with the least possible vertices for a given dimension. Until recently, this method was under a patent [4]. However, I will not be using Simplex noise because I would have to implement it myself. This is not the case with Perlin noise since it is part of the standard mathematical library in Unity.

2.1.4 Physical Processes

Generated landscapes can be further detailed by simulating physical processes such as hydraulic erosion, fractional Brownian motion, or weathering [2]. I would like to focus more on hydraulic erosion since it is the easiest one to implement and, in my opinion, looks the best.

Hydraulic erosion can be simulated as a rain of water droplets, subtracting

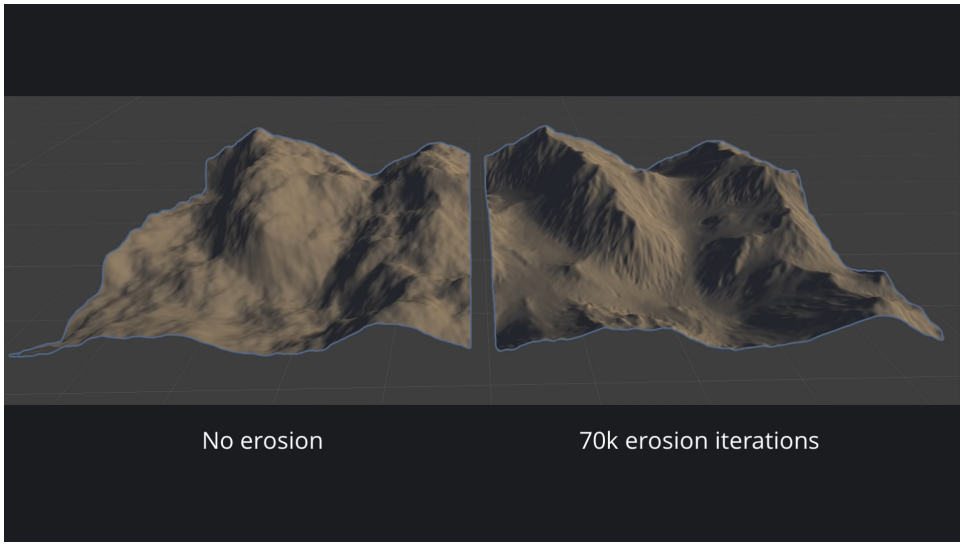


Figure 2.4: The comparison of the effect simulation 70 000 water droplets on a noise generated terrain from Lague's [6] video.

and depositing sediment from the terrain as they travel down the gradient. Lague [6], in his Youtube video about hydraulic erosion, simulates each droplet as it travels down the terrain using several attributes.

Using the inertia parameter, he calculates the droplet's direction as a mix of the previous direction and the current gradient. The simulation also works with the speed of the droplet, which is simulated using gravity and height change. As the droplet slows down or reaches the maximum carrying capacity, it deposits the sediment carried gradually.

The amount of sediment subtracted from the terrain is based on how much the height has changed from the previous step of the droplet. If this were not the case, the droplet would create impossible scenarios for flowing water, which would result in weird pits and spikes on the terrain. The termination state for these droplets is reached when they start flowing upwards, their speed reaches zero, or their water level, which slowly evaporates as they travel, reaches zero.

These simulations tend to be quite computationally demanding. Lague [6] uses compute shaders which are run in parallel on the graphics card, to speed up the calculation immensely. This allows him to simulate tens of thousands of droplets in real-time. I will not be using compute shaders due to the nature of my solution, where it would be quite difficult.

The result of such a simulation can be seen in Figure 2.4 taken from Lague's [6] video about hydraulic erosion.

2.1.5 Agent based approach

Doran and Parberry [10] propose a quite different approach to landmass generation. They use agents that have defined actions, lifetimes, and parameters. The agents traverse the terrain and create its features. The agents

described in Doran and Parberry's paper [10] can perform these tasks, which are executed in the following order:

- Coastline - A single agent is sent out, which defines the terrain's general shape by raising it. The agent can split into multiple agents, each of which works on a separate part of the map for better performance.
- Smoothing - After the coastline agents, smoothing agents are sent out to eliminate rapid elevation changes in the terrain. They move randomly and change the elevation of points by interpolating with their neighbors.
- Beach - These agents create sandy areas around water bodies by traversing the shoreline. They vary the heights of points by sampling from a designer-specified range of values. Using this approach, different kinds of beaches, such as flat or bumpy, can be created.
- Mountain - The beach agents keep regions above a certain threshold untouched. Mountain agents modify these areas. The agents are placed on random map points and move randomly like the smoothing agents. Upon encountering a V-shaped wedge, it is elevated to create a ridge. They also periodically make foothills. After this step, another smoothing step is used.
- Hill - Hill agents work similarly to Mountain agents but on smaller altitudes and with smaller ranges of values. They are also not allowed to create foothills.
- River - This is the last step of generation. Each agent gets assigned two random points. One on the coastline and one on a mountain ridge. The agent moves from the shoreline to the mountain ridge guided by the gradient. Once the mountain ridge is reached, it returns to the coastline while digging a wedge. The wedge gets wider the closer to the sea the agent is.

Most of these agents have many parameters and designer-specified ranges of values to use for more user freedom [7].

This approach is important for my bachelor thesis in the area of generating rivers, hydraulic erosion, and paths because a conceptually similar agent-based approach is used.

Chapter 3

Implementation

I have divided this chapter into sections based on the generated feature and the component responsible for it. The components generate different terrain features, which is why they can be skipped or do not have to be used in a particular order, although a set order is encouraged for best results. The user needs to generate the terrain first using the Generate Terrain component 3.1.2. Then the recommended order of generation is water 3.6, hydraulic erosion 3.5, paths 3.3, and lastly roads 3.4.

3.1 Base Terrain

The terrain is represented as a mesh of evenly spaced vertices that create faces. This terrain is divided into chunks. This is done so that the shape of the terrain can be easily controlled. This approach also proves useful due to Unity's limit of 65 535 vertices per mesh.

However, there is one disadvantage to using this chunk-based approach. On the edges of chunks, the lining vertices are present in both chunks. This means more data is sent to the graphics card than needed. Also, the normal vectors of these lining vertices are calculated incorrectly, causing visible seams. This is due to how normal vectors are calculated. Normal vectors for vertices are calculated as an average of normal vectors of all the faces the vertex is present in.

I fixed this by not using Unity's built-in function to recalculate the normals of a mesh and implementing my own function. The function temporarily embalms the mesh in a border layer of vertices that are used to calculate the normals. The vertices are generated using the functions of the surrounding chunks. This is needed because of the height-changing textures discussed in the subsection Texture-based Approach 3.1.1. Figure 3.1 shows a visualization of the process. This problem was discussed in an episode of Lagues [3] Youtube series about Landmass generation, and his approach inspired my implementation.

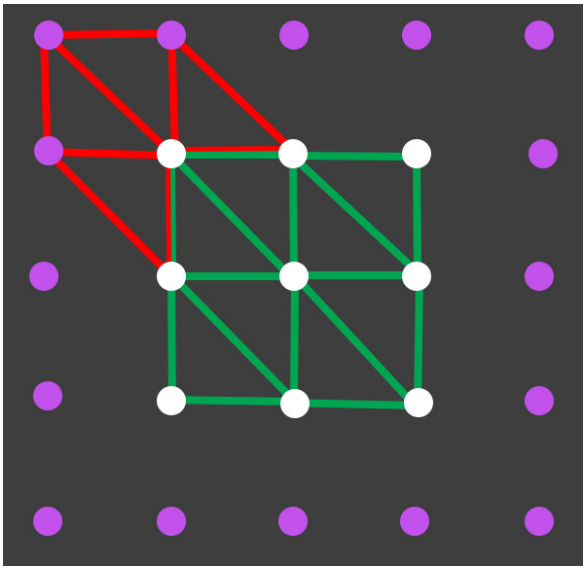


Figure 3.1: In this Figure, the green triangles and white points represent the terrain, the purple points are the embalming points used to calculate the normals. The red triangles are all the triangles whose normals contribute to the normal vector of the top left point of the terrain.

3.1.1 Texture-based Approach

Since I wanted the different features of the terrain to be easily created, edited, and deleted, I chose a texture-based approach. For each chunk, every feature has its texture. The colors in the texture represent the changes to be made to the terrain at a specific point on the chunk. Each feature uses the three color channels and one alpha channel differently. An example can be seen in Figure 3.2.

This makes it very easy to control each terrain feature made by a generator and even change its influence, which adds an extra level of freedom. Another advantage is that the terrain's level of detail, determined by the number of vertices, can be changed without losing the generated features. However, some features are still generated using the current detail, such as hydraulic erosion, so the user should set the highest detail that will be used in their application before generating.

The texture-based approach has a few disadvantages. One of them is the number of textures created for a larger terrain containing a lot of chunks. These textures, at the high-resolutions, start taking up a lot of storage and processing power in the drawing phase. I experimented with chunk sizes and texture resolutions for each generator. I decided that a chunk size of 30 Unity units, corresponding one to one with real meters, and two resolutions, which are used based on the generator type, of 128x128 pixels and 256x256 pixels work well.

All generated textures are stored in the folder "Assets/ChunkTextures/", where each scene has its folder. There are four more folders for each generated texture type in these scene folders.

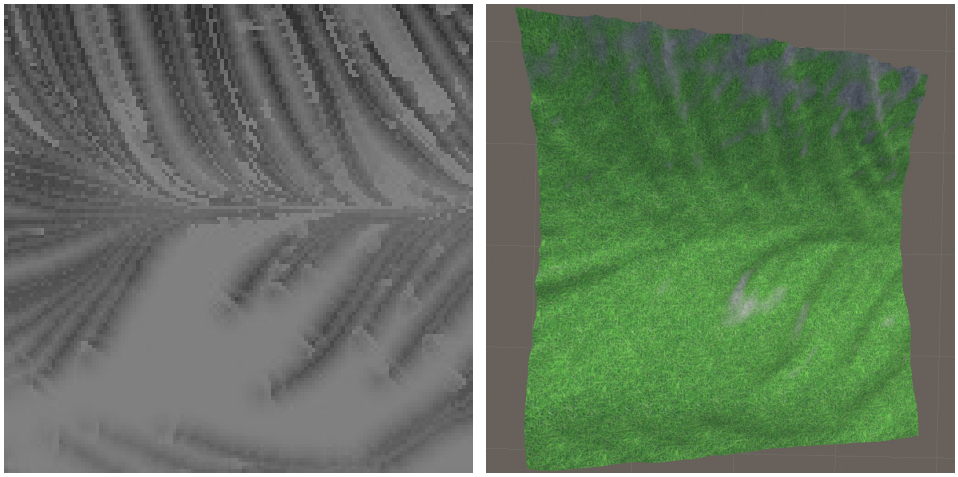


Figure 3.2: The left image is a hydraulic erosion texture of a chunk, where gray means no change in height, darker color means the terrain is subtracted, and lighter color means the terrain is heightened. The right image is the texture applied on a chunk.

3.1.2 Generate Terrain Component

The terrain is generated using the Generate Terrain component. The component has many parameters that can be changed to generate different terrain. However, the user can still change these even after generation. Some example terrain can be seen in the second picture of Figure 2.3 and in Figure 3.3. All of these settings have a tooltip when the user hovers over them. These include:

- Seed - A seed used for the random number generation that controls the offset from the origin of the Perlin noise.
- Parent - GameObject, which will act as a container for the chunk manager, all the generated chunks, and other generators. A new GameObject called "Terrain" will be used if none is given.
- Perlin scale - The scale that will be applied to all the levels of Perlin noise (more details in 3.1.5).
- Amplitude - The amplitude used for the Perlin noise function. This can be thought of as the maximum height of the terrain.
- Number of Octaves of Perlin noise - Number of Perlin noise octaves that are used to recursively calculate the height of vertices (more details in 3.1.5).
- Wrapping - Number of vertices per row of a chunk. For example, if the wrapping is 15, there will be 225 vertices in the chunks.
- Width and Height - The number of rows and columns of chunks that will be generated.

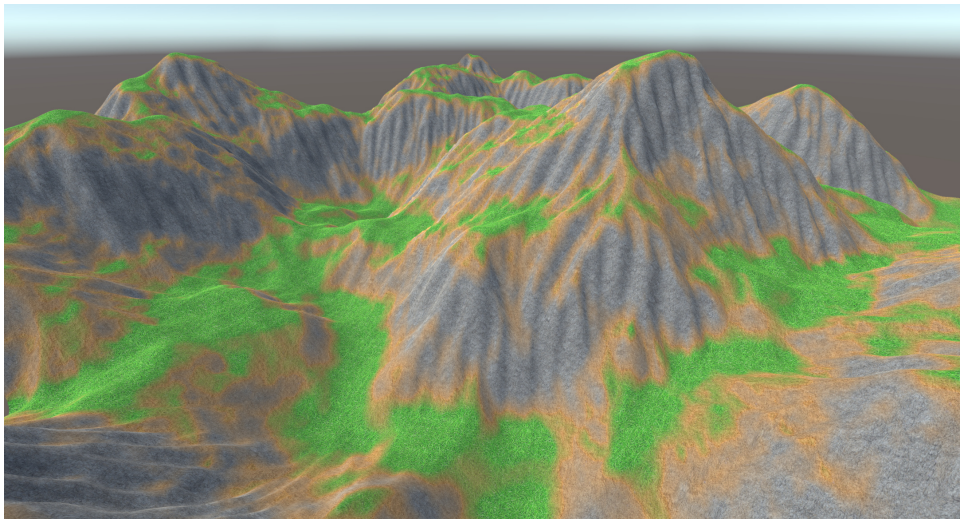


Figure 3.3: A mountainous terrain generated using my tool with hydraulic erosion applied.

- Material - The material that will be applied to chunks.

3.1.3 Chunk Manager Component

A chunk manager is created upon terrain generation. In the scene hierarchy, the chunk manager has all the chunks as children. Its primary purpose is to control all the chunks and their parameters. Some of these parameters were discussed in section 3.1, such as the Number of Octaves, Amplitude, Material, and Wrapping. The new parameters that can be edited in Chunk Manager include the following:

- Offset from Seed - The seed from 3.1.2 was used as a seed in Unity's random number generator, which calculated this offset that is applied to the x and y coordinates in Perlin noise calculation.
- Is Hilly - Toggle parameter which switches between two different terrain types. The difference can be seen in Figure 3.4.
- Lacunarity - Controls the size of pattern gaps. Determines how quickly each octave increases its frequency [1].
- Persistence - Determines how quickly each successive octave's amplitude diminishes [1].
- Edit of Octaves - This is a list of 3D vectors. There are as many vectors as octaves of Perlin noise. The X and Y components are added as offsets for coordinates used for the Perlin noise calculation of the current octave. The Z component is used as a scale of the Perlin noise octave.

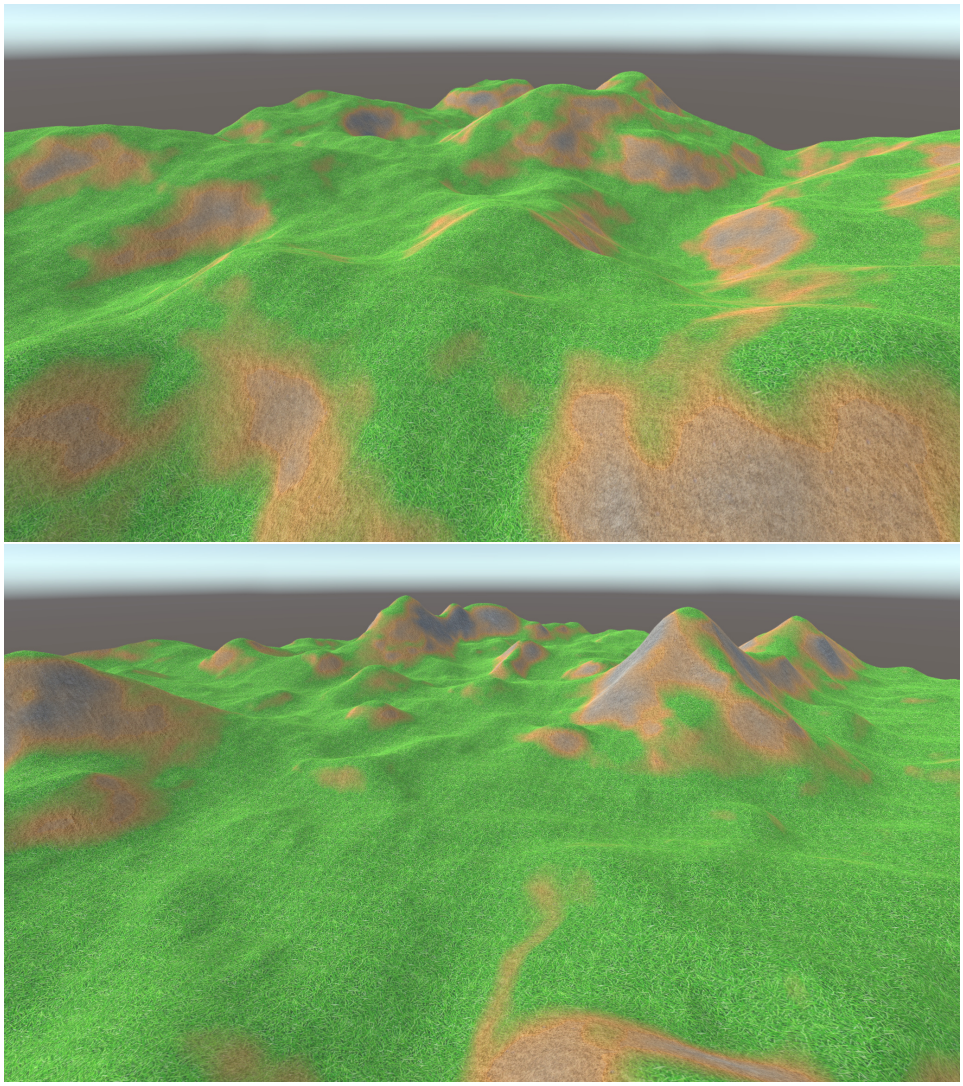


Figure 3.4: The first image is a terrain with many hills and no flat regions. The second image is a flatter terrain with occasional mountains.

3.1.4 Chunk Component

A chunk is a square part of the terrain measuring 30 meters on each side, as mentioned in the subsection Texture-based Approach 3.1.1. In the inspector window for this component, there are four buttons. Each of them creates a chunk in one of the four cardinal directions. This is useful when users need more chunks for their landscape. These buttons can even be used in multi-editing (selecting more than one chunk simultaneously). Upon generating a new chunk this way, the new chunks are automatically selected so the user can easily continue generating in the same direction.

New water chunks 3.6.1 will also be generated with the normal chunks if the user has used the water generator component 3.6 and generated water chunks.

Additionally, each chunk can act as a height effector. These height effectors have two configurable parameters height and range. As the name suggests, these chunks offset the height of vertices in a set range by a set height value. The operation is most potent in the center of the effector chunk and falls off with distance. This is done by running the final value through the bicubic interpolation function f(x)=$-2x^3 + 3x^2$, often used to smooth out terrain changes [7].

3.1.5 Height Calculation

The height of each vertex is calculated by the following process. Firstly, the coordinates for Perlin noise are calculated. Next, these coordinates are input into Unity's built-in Perlin noise function. Then the resulting number, ranging between 0 and 1, is multiplied by two and subtracted by one to change the range of values to -1 and 1. Now if the parameter Is Hilly is toggled off, the number is run through the MakeCubic() function. Finally, the number is multiplied by the amplitude and divided by the number 2 or 5, based on terrain type, to make the height range of the terrain better correspond to the amplitude parameter. Then the current frequency is multiplied by lacunarity and the current amplitude by persistence. This is done once per octave of Perlin noise in use.

Once the height is calculated, an additional height from all the height-affecting chunks, which are in range, is run through MakeCubic() and added. Then the height from all textures generated by the different generators is added. This is done by sampling the texture and using the pixel color based on parameters and rules for a given generator. A 3x3 block of pixels are sampled and averaged for road and erosion textures achieving better smoothness. These textures also use cubic ease in and out function [17].

Here is an excerpt from the height calculation function:

```
private float GetHeightOfPoint(float x, float z)
{
    Vector3 chunkPosition = transform.position;
    float perlinX = point.x + chunkPosition.x
                  + _offsetFromSeed;
    float perlinZ = point.z + chunkPosition.z
                  + _offsetFromSeed;

    float height = 0;

    float tempFrequency = 1;
    float tempAmplitude = _amplitude;

    // Recursive detail
    for (int i = 0; i < _numberOfOctaves; i++)
    {
        // Get the right edit of octave
```

```
            ...

            if (_isHilly)
            {
                height +=
                    (Mathf.PerlinNoise(
                    tempX + editOfOctave.x,
                    tempZ + editOfOctave.y)
                    * 2 - 1) * tempAmplitude / 2;
            }
            else
            {
                height += MakeCubic(
                    Mathf.PerlinNoise(
                    tempX + editOfOctave.x,
                    tempZ + editOfOctave.y)
                    * 2 - 1) * tempAmplitude / 5;
            }

            tempAmplitude *= _persistence;
            tempFrequency *= _lacunarity;
        }

        // Height addition from textures
        ...
}
```

Where _amplitude, _persistance, _lacunarity, _editOfOctave, and _isHilly are the parameters set in chunk manager 3.1.3. The function MakeCubic() is a function for bicubic interpolation described in 3.1.4.

3.2 Material and Shaders

I have created a special material and a shader that colors the terrain based on the steepness, altitude, and features generated. All textures used in the images in this bachelor thesis were generated using a web-hosted artificial intelligence generator [16].

3.2.1 Textures and Color

Users can adjust three steepness textures with parameters that decide the thresholds and the amount of blending between them. There are two altitude-based textures for terrain above and below two settable thresholds. These can be used for snow on mountaintops and sand in underwater terrain. Users can also select the textures for the generated features, such as paths, roads, etc. Each texture mentioned has a settable scale.

All of the textures are used with Triplanar mapping [22]. In normal texturing, the texture is projected onto the terrain from one direction, usually downwards. This causes the textures to stretch on steep surfaces. Triplanar mapping projects the texture from all three main directions, the X, Y, and Z axis. Then all three projected textures are weighed using the direction of the normal vector. This process not only removes the stretching but also makes the textures more variable, causing patterns to be less visible.

3.2.2 Grass Shader

The last feature I implemented in the shader was grass rendering. There are a lot of techniques used for grass rendering. Several of them were described in a video by Daniel Ilett [13].

Methods

One of them is billboarding, where instead of the grass having a complex mesh, it is represented by two or more intersecting quads with a clump of grass texture. This method is not particularly performance-heavy but does not generate convincing results [13].

Another method is the use of compute shaders and procedural rendering. The compute shader generates transform matrices containing the grass blade meshes' positions and rotations. Then only a single grass blade mesh is sent to the GPU, and using a special shader, the grass blade is rendered many times using all the transform matrices. This method can render many grass blades very efficiently [13].

The last interesting method I will mention, and the one I chose, is using a tessellation and geometry shader. This method uses the geometry of the terrain and adds a triangle, in this case representing grass blades, above each face. To control the amount of grass, even if the terrain does not have many vertices, tessellation is used. This method is closely detailed in the section below.

Implementation

I used Roystan's [11] article, which contains an excessive tutorial on implementing a grass shader using this method. However, I still needed to use my old shader for drawing the terrain. Luckily, shaders in Unity can have multiple passes. So the shader does a first pass where the terrain is rendered and colored based on principles mentioned in the subsection Texture and Color 3.2.1. Then another pass using a modified implementation of Roystan's [11] grass shader is done, which renders the grass blades.

The grass shader works as follows. After the Vertex shader, the vertices and their properties are passed to the Hull shader. Its job is to prepare all the data needed for the tessellation stage and pass it. Next, the tessellation stage happens in the Domain shader, for which we need to specify how much each triangle should be split using Tessellation factors [14]. Once this division

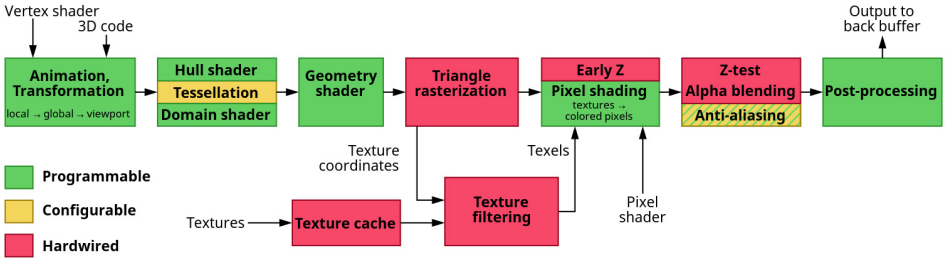


Figure 3.5: Modified visualization of the rendering pipeline used in modern applications with color-distinguished stages based on programmability.[12]

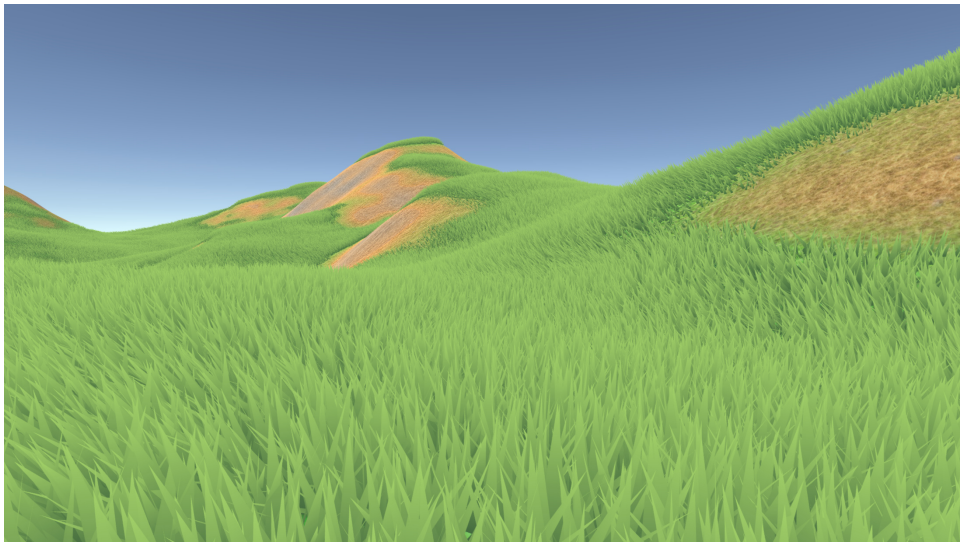


Figure 3.6: Image showing the grass shader in action.

is done, the Geometry shader is run on each triangle. This stage creates one triangle representing a single grass blade. It has random height, width, bend, and rotation on the Y-axis. An extra bend from a wind texture is added, which changes with time, giving the grass believable animation. In the last stage of the grass shader, the grass blades are colored and drawn to the screen. The order of the rendering pipeline can be seen in Figure 3.5

While creating the shader, I used the Hull and Domain shaders implemented by Jasper Flick [14] in his article about Tessellation. I also did not make it possible to have more triangles per grass blade as it was not as important to this bachelor thesis, and I was already satisfied with the result. The tessellation of the grass is controlled by the grass texture, meaning it does not generate on steep hills. While on the edges between the grass and dirt texture, the grass blades become less dense and smaller in size. For the wind displacement texture, I used the water distortion texture provided by Roystan [11] in his tutorial. The grass is very customizable and includes many parameters to help with performance and general look, such as:

- Use Grass - Toggle to turn on or off the grass rendering.

- Grass Density - The user can change how much the triangles tessellate, thus controlling the performance impact.
- Minimal Grass Density - The smallest amount of tessellation allowed. It is used on the edges of grass textures and in reduced tessellation at a distance, described in the parameters below.
- Cull At Maximum Distance - Controls if grass should be culled at a maximum distance, a sum of High Detail Distance and Detail Interpolate Distance.
- High Detail Distance - Distance from the camera to which the tessellation set by Grass Density will be used before interpolating to Minimal Grass Density.
- Detail Interpolate Distance - The distance from the edge of High Detail Distance where the tessellation will be gradually interpolated from Grass Density to Minimal Grass Density.
- Blade Bottom Color - The color applied to the bottom two vertices of each grass blade.
- Blade Top Color - The color applied to the top vertex of each grass blade.
- Blade Width - The distance between the bottom two vertices of each grass blade.
- Random Offset of Width - Sets to maximum width by which a blade can be smaller or larger than the default Blade Width. The offset in width of each grass blade is randomly [15] generated in the shader based on world position. This makes the grass look less uniform.
- Blade Height - The distance from the ground to the top vertex of each grass blade.
- Random Offset of Height - Sets to maximum height by which a blade larger than the default Blade Height. The offset in the height of each grass blade is randomly [15] generated in the shader based on the world position. This also makes the grass look less uniform.
- Blade Bend Amount - Controls each grass blade's maximum default forward bend amount. The actual amount for each grass blade is randomly [15] generated in the shader.
- Wind Map (RG) - A texture whose red and green channels will be used as the rotation amount in the X-axis and Z-axis for the top vertex of each grass blade. The texture moves with time to simulate wind movement.
- Wind Scale X-axis - The speed of scrolling through the texture on the X-axis.

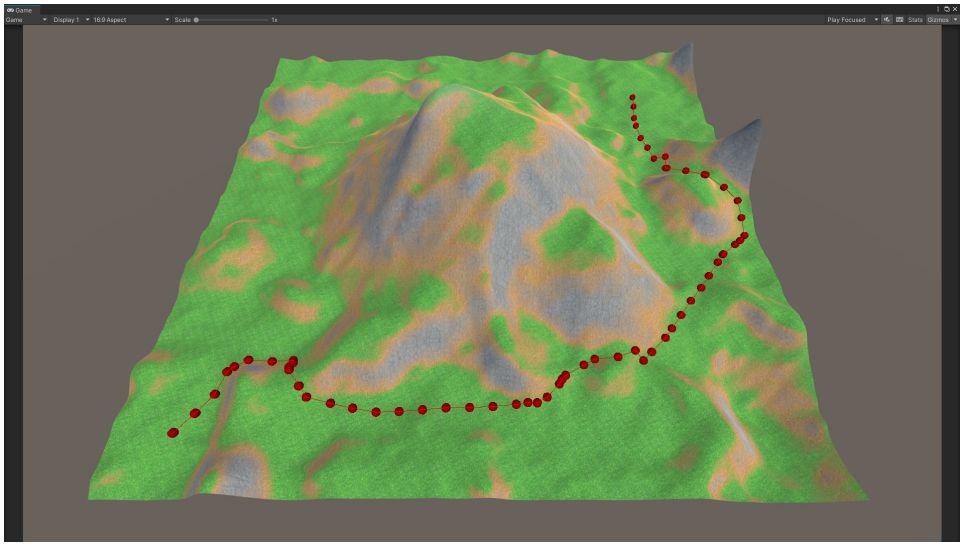


Figure 3.7: A control points of a path generated by the combined efforts of the global and local agent.

- Wind Scale Z-axis - The speed of scrolling through the texture on the Z-axis.
- Wind Power - Controls the maximum amount each grass blade can bend from the wind.

3.3 Generate Path Component

This component is created upon the initial terrain generation by the Generate Terrain Component. It is responsible for human-made path generation. These paths can be made winding, and they can shape the landscape. The texture which will be used for the path can be set in the material of the terrain 3.2.1.

The component is split into three parts. The first one is the approximation of the path using control points by manual point addition and path-finding. The second one is the generation of textures, which offsets the terrain. The last one is the option to generate random paths on the terrain.

3.3.1 Path Control Points

The main idea is to create an approximation of the path using control points. These control points can be manually added by toggling the button mentioned below and clicking on the terrain. These points are visually represented as small red spheres connected by lines. After adding at least two points, the user can send an agent to achieve a good density of control points.

The agent consists of two path-finding agents working together to connect the path. One is a global agent that uses the Unity NavMesh System, which generates the shortest path using control points. However, the density of the created control points is low, and the distances are usually larger than the

Travel Distance parameter. The other is a local agent that I have implemented, which fills the gaps between the control points left by the global agent. The result of the global agent can be seen in Figure 3.8, the result of the local agent in Figure 3.9, and path-finding using both agents can be seen in Figure 3.7.

This component has several parameters and buttons:

- "Start Manually Adding Points" button - When active, all mouse clicks on the terrain call a raycast, and a control point is placed on the intersection with the terrain.
- Control Points - An array of points used as control points for the curve of a path. They can all be edited in the inspector window.
- "Reset points" button - Deletes all points from the Control Points array.
- "Reset except 1st and last" button - Deletes all but the first and last point from the Control Points array.
- Show Control Points - A toggle that changes the visibility of the control points.
- Travel Distance - How far apart the control points should be. Used by the local path-finding agent as the distance between each step.
- "Send Agent To Generate Control Points" button - Connects the manually added control points using the global and local agent.

Global Path-finding Agent

The global path-finding agent works similarly to humans with an aerial view of the terrain. It always finds at least a partial path to the goal. The paths created are straightforward and the shortest possible while abiding by set constraints.

My first idea of how to implement this kind of agent was to create my own A* algorithm. This proved quite tricky, especially when generating my own graph. So I tried a different approach using Unity's built-in NavMesh system. A NavMesh is a mesh generated by Unity to approximate walkable areas in an environment for path finding and AI-controlled navigation (NavMesh agent) [5]. This also proved difficult because you cannot control the NavMesh system using scripts. I had to add an official experimental Unity package called "NavMesh Building Components".

Using this package, I was able to generate a NavMesh using a script. However, I was still unable to control the parameters of the NavMesh agent using a script due to problems with the package. I solved this problem by attaching a NavMeshSurface component to the GameObject, where the user can change the parameters of agents, and by displaying a warning in the inspector that explains the situation.

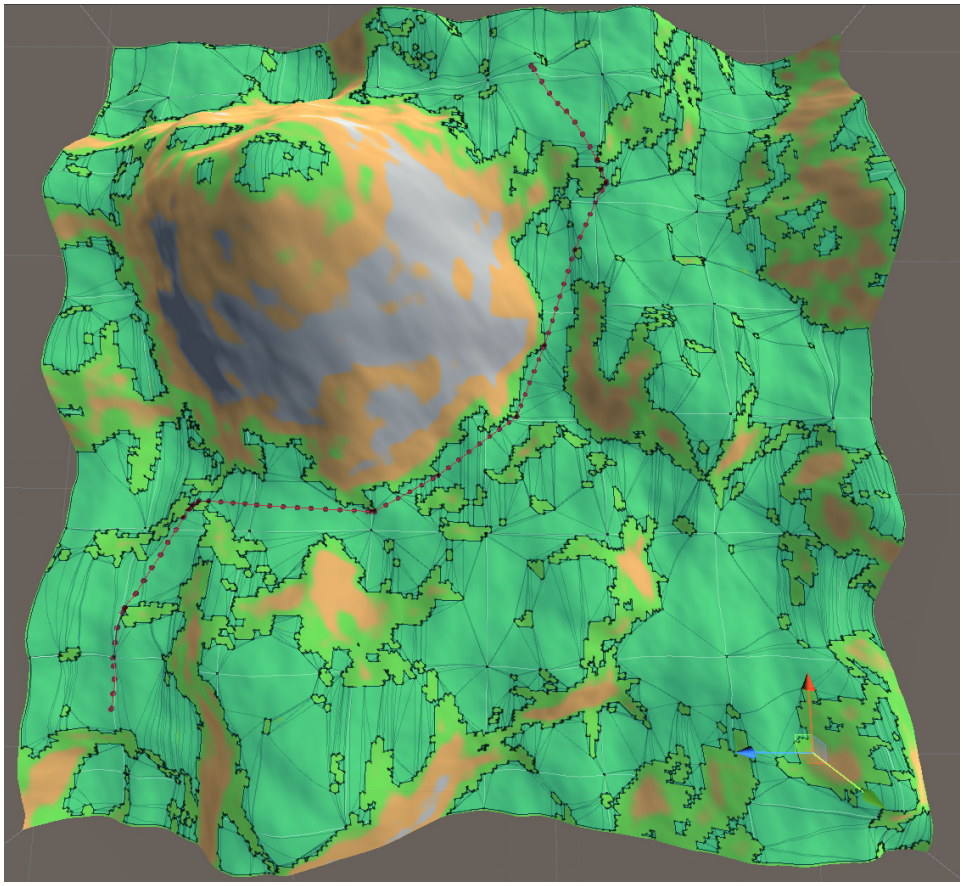


Figure 3.8: A straightforward path generated using global path-finding with a NavMesh (turquoise areas). Additionally, for better visibility, the corner points have been connected by more points using linear interpolation.

The NavMesh agent has several parameters that influence the generation of the whole NavMesh. The shape of the agent is a cylinder, and many of these parameters are related to it. The parameters are:

- Radius - The radius of the cylinder that represents the agent.
- Height - The height of the cylinder that represents the agent.
- Step Height - The maximum height of a step that the agent can take (does not influence the result in my use case).
- Max Slope - Controls the maximum steepness the agent can walk on.

The path is created by generating a NavMesh for the current terrain configuration. Then a path is calculated using NavMeshSurface's CalculatePath() function, which represents the path as the starting point, all the corner points, and the target point. Corner points are points on the edges of the NavMesh, which the agent needs to go around to stay inside the NavMesh. The next step is to add the corner points to the control points array. An example of a

path generated using only global path-finding with NavMesh can be seen in Figure 3.8.

Local Path-finding Agent

The local path-finding agent works similarly to humans without a map of the terrain. It is not as reliable as the Global path-finding agent. Sometimes it can get stuck circling around the goal or not being able to find a valid path. However, in most cases, it helps to create more natural paths.

The agent starts at the first control point and searches for the next one. At every search step, the agent turns 90 degrees left and samples a point Travel distance away in front of it. This sampling is done via raycasting several meters above the terrain in the downward direction. The process produces a single point, which is evaluated by the following reward function (a smaller value is better):

$$value = steepness + slope + differenceToLastDirection \\ + directionToGoal/distanceToGoal$$

Here is a quick explanation of all the parts of the value:

- Steepness - How steep the terrain is in the direction of the sampled point. It is calculated using the difference in the height of the current location and the sampled point.
- Slope - Steepness of the terrain at the sampled point. Calculated by the angle difference between the normal vector of the terrain and a vector pointing upwards.
- Direction To Goal - Angle difference between a vector pointing to the goal and a vector pointing to the sampled point.
- Difference To Last Direction - Angle difference between a vector pointing from the last step to the current step and a vector pointing to the sampled point.

Once the point with the smallest reward value is found, it is added to the control points list, and the process of searching starts again from this new point. This is done until the agent reaches the last point.

Connecting the control points using the local path-finding agent has multiple parameters. These include:

- Number of Steps Until Failing - The maximum number of control points added between two control points. Used as a terminal state so that the agent does not get stuck in an infinite loop.
- Max Walkable Steepness - The maximum steepness the agent can directly walk on.

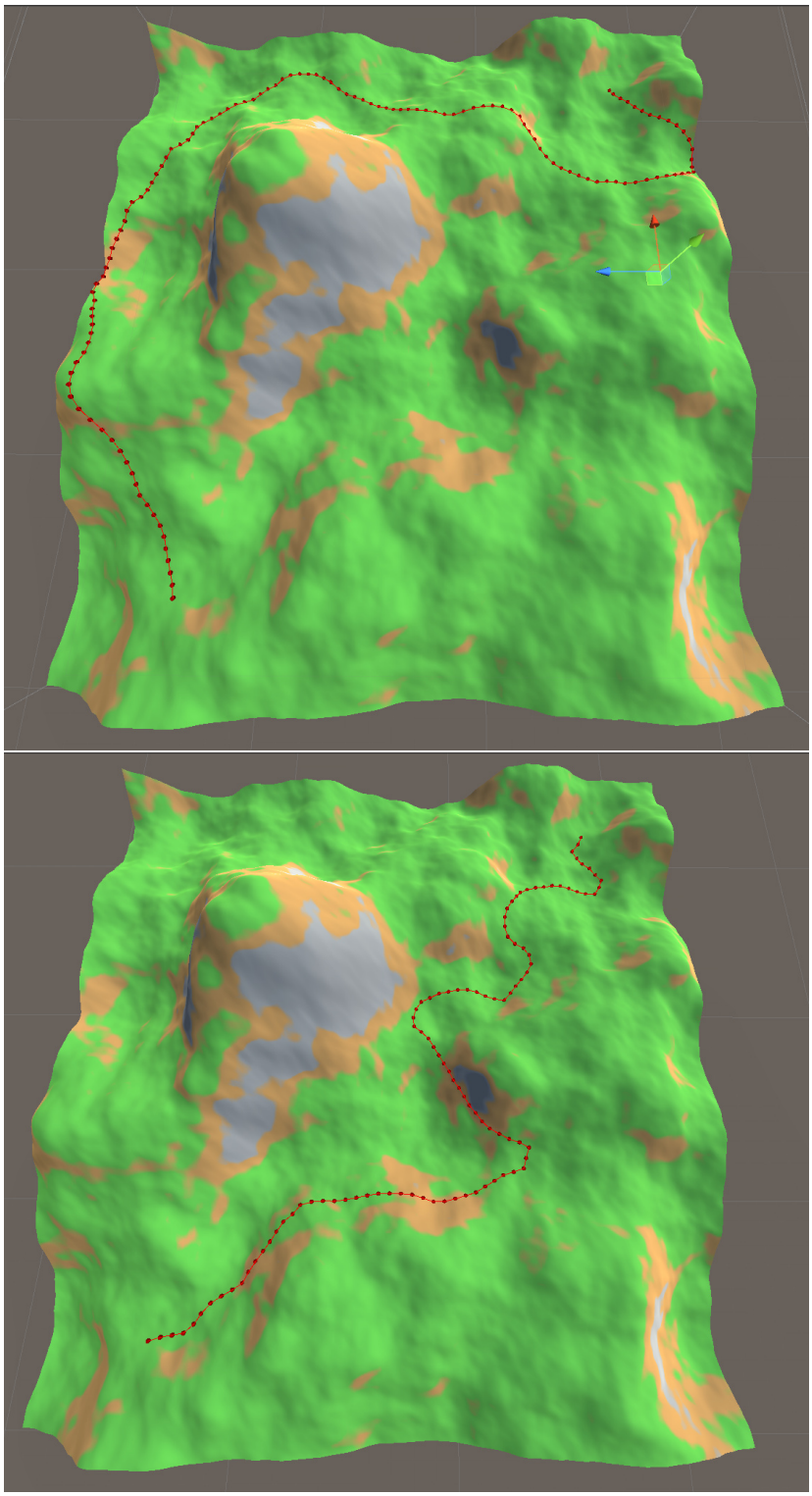


Figure 3.9: Two different winding paths generated using local path-finding. The first one had default weights. The second one had the Direction To Goal weight lowered to create a less straightforward path.

- Max Walkable Slope - The maximum steepness of terrain the agent can walk on without considering the direction.

Furthermore, there are settable weights for all the reward function parts in the inspector window of the component. In Figure 3.9, there are some examples of the same configuration of control points (Start top right, Goal bottom left) generated with different weights of these parameters. The paths were generated on the same terrain using only the local agent to better show the reward parameters' effects.

3.3.2 Path Mesh Alteration

Once the user is satisfied with the path approximation using control points, they can generate the path, thus creating the path texture for all affected chunks.

Before generating, the user should choose the size of the path to generate. This can be done using two parameters which can be changed between generating to create differently sized paths. The first one, Width Of Paths, controls the threshold from which the path will start blending into the surrounding terrain. The second one, Path Smooth Width, controls the distance to which the blending will occur.

Two buttons are located at the bottom of the inspector's section. One is called "Generate Path", which takes the control points and generates a path on the terrain between them. The other is called "Reset Path", which resets all the path textures of the terrain.

During the generation, three rays are cast for each step along the path approximated by control points. One is in the middle of the path, one on the left edge of the path, and one on the right edge. This is done to get the height of the terrain at those points. The height is used to level the terrain, which makes the paths look more realistic. The result of this leveling can be seen in Figure 3.10.

The leveling can be controlled by an interactive parameter called Path Incline Balance. This affects all the paths generated. Another interactive parameter is Path Y Offset, which controls the paths' height offset.

As mentioned in 3.1.1, the final texture uses the color channels in specific ways. The alpha channel is used for the path influence over the terrain. If the alpha channel is 0, no terrain change is made. If it is 1, the terrain is changed by Path Y Offset. Lastly, if it is between 0 and 1, the path starts blending with the surrounding terrain. The color channels are used for leveling the terrain. Gray color means no change in height, when a darker color is present, the terrain is carved into, and when a lighter color is present, the terrain is heightened.

The result of terrain with generated paths can be seen in Figure 3.11.

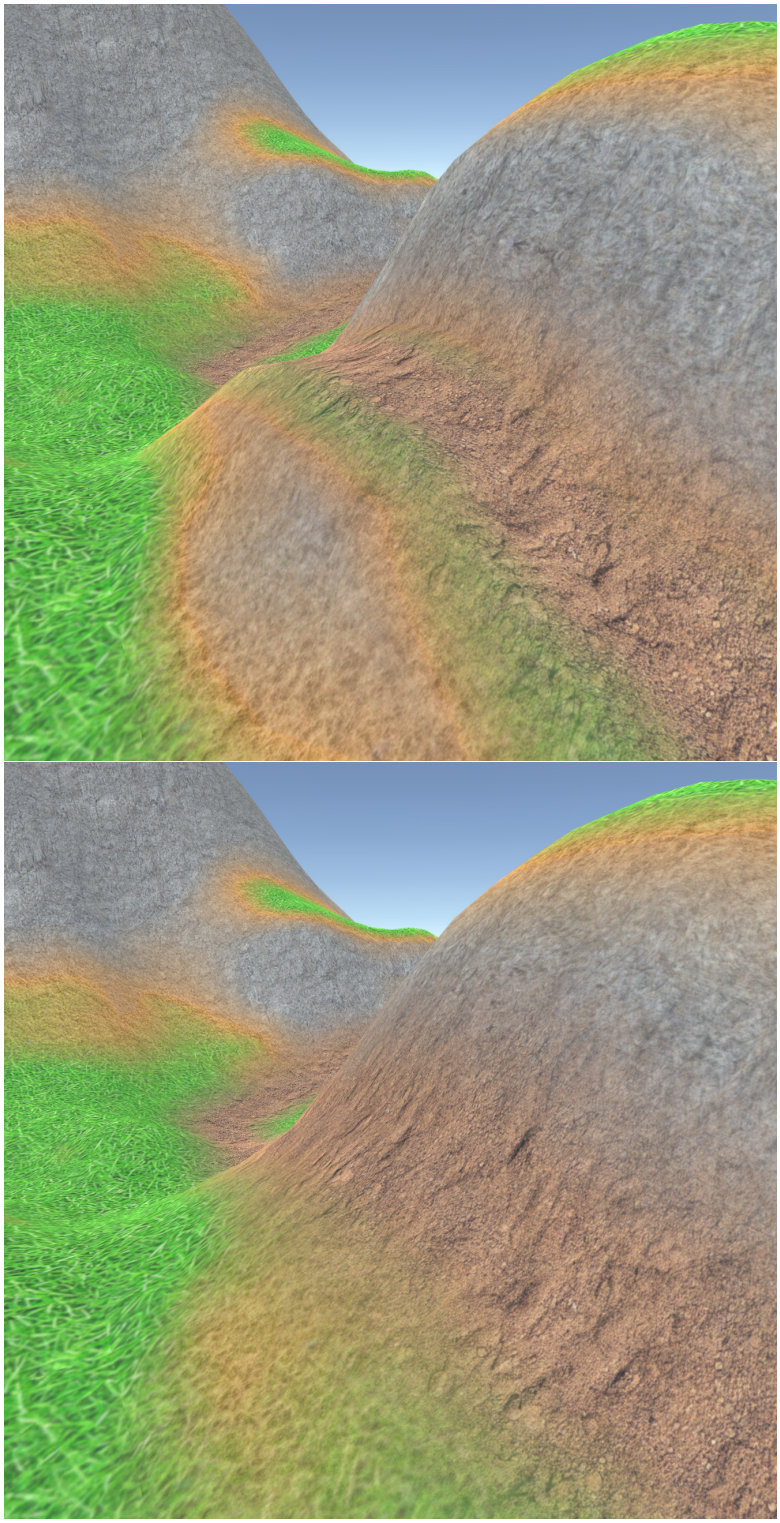


Figure 3.10: The first picture shows the Path Incline Balance parameter set to -15. In comparison with the second picture, where it is set to 0.

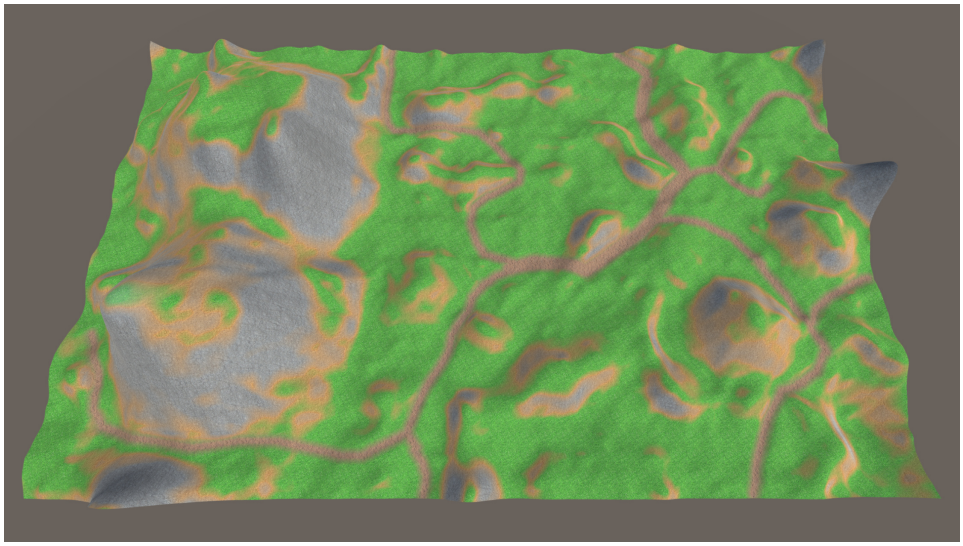


Figure 3.11: Terrain with generated paths using the Generate Paths Component.

3.3.3 Random Path Generation

This component's third and final part is the option to generate random paths. The user can specify the number of paths they want to create, and the generator tries to generate them. It is done by choosing two random chunks and two random points in these chunks. With these control points, the global and local agents are called. They try to connect the points using the parameters set in the component above. If the agents manage to create a valid path, it is generated using the parameters from the Path Mesh Alternation described in section 3.3.2. Depending on the terrain and parameters, this feature can create anywhere between zero and the specified number of paths.

3.4 Generate Road Component

This component is also generated with the initial generation of terrain using Generate Terrain Component. It is responsible for generating roads on the terrain. Roads are generated in a similar way as paths so I will cover this generator briefly. However, roads do not have any agents to connect the control points. Users can manually add control points to the terrain, set their desired width, and generate. The texture of the road as well as the color of the road lines can be set in the material of the terrain 3.2.1.

The component has the same parameters for creating and editing control points as paths described in section 3.3.1.

During each step of the road generation, a target height is set by interpolating between the heights of the closest two control points. For each texture pixel, the terrain is sampled using a raycast at the world location of the pixel. Thanks to this, we know how much the terrain mesh needs to be altered in this location to achieve a flat road. This method is precise and surprisingly

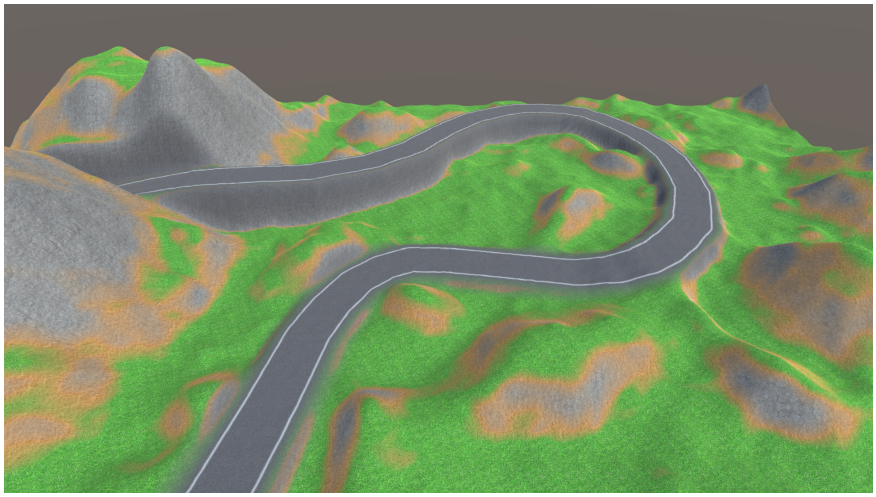


Figure 3.12: Terrain with a road generated. The road can heighten the terrain or even cut through it, as seen on the left side of the image.

not very performance-heavy.

The texture uses the green channel to save the height adjustment, the red channel to draw road lines, the blue channel to indicate the distance from the middle of the road, and the alpha channel to smooth the edges of the road, similar to paths.

For the mesh alteration, users can change the following parameters:

- Generate Road Lines - Toggles if the road to be generated should feature road lines.
- Width Of Roads - The threshold at which the road generates road lines and starts to blend with the terrain.
- Road Smooth Width - The distance from road lines, where the road will be blended with the terrain. A high number is recommended for big height differences.
- Road Max Y Offset - The maximum height change the road can make to the terrain. It will be used as the divisor of the height change. This is done because textures store the color as values between 0 and 1. The user should set it before road generation and not change it.

Below these parameters, there are two buttons. The first one, "Generate Road," uses the control points and draws a road on the corresponding textures. The second one, "Reset Road," resets all the road textures of all chunks.

An example of a road can be seen in Figure 3.12.

3.5 Generate Hydraulic Erosion Component

This is another component created upon the generation of the terrain. Using this component, the user is able to deform the landscape using the natural

hydraulic processes of water as detailed in the subsection Physical Processes 2.1.4. The water droplets are placed on the terrain, and in each step, they slide down toward the most significant descent. They take some sediment with them down, displacing the height of vertices. Due to this component dramatically changing vertex height, it should be used and configured before generating roads, paths, and rivers.

This component has tools to manually drop water droplets on the terrain so that the user can test the behavior of the droplets, which can be changed by the parameters listed below. These include:

- Drop Type - Switches the type of water droplet generation between Grid and Random. Grid drops the water droplets in a grid pattern with equal offsets. Random generates the droplet positions randomly in each chunk.
- Maximum Angle - If the angle between the current direction and the previous is larger than the maximum angle, the droplet is stopped.
- Droplet Lifetime - Maximum number of steps a water droplet can take.
- Inertia Of Droplets - A value from 0 to 1 symbolizing how closely the direction from the previous step is kept.
- Droplets Per Chunk Row - Determines how many droplets are dropped on terrain per chunk row. For example, if set to 9, 81 droplets will be dropped on each chunk.
- Droplets Width Of Effect - Determines how far each droplet influences the terrain. A default value of 3 or 4 works well.

At the bottom of the component, two buttons are located. One, "Generate Erosion," starts simulating the water droplets and creates erosion textures. The other "Reset Erosion" resets all the textures of all chunks.

Once the texture is generated, users can influence the effect of hydraulic erosion on the terrain using an interactive parameter, Erosion Strength. The result of hydraulic erosion can be seen in Figure 3.3 or in Figure 3.13

3.5.1 Droplet Simulation

For each droplet, two processes are run. The first one simulates the droplets' path down the terrain using the gradient and inertia while calculating the sediment carried. If the water droplet still has the capacity, the sediment is increased by a part of the height difference between the current and last points. If the droplet reaches full capacity, the sediment slowly decreases. This is also the case if the droplet starts flowing upwards in flatter regions. During the droplet's descent, its speed, which is decreased with gravity and drag, is simulated.

The lower the speed, the smaller the capacity, and when it reaches zero, the droplet stops simulating. The Maximum Angle parameter can be used to create another terminal state. When the angle between the direction from

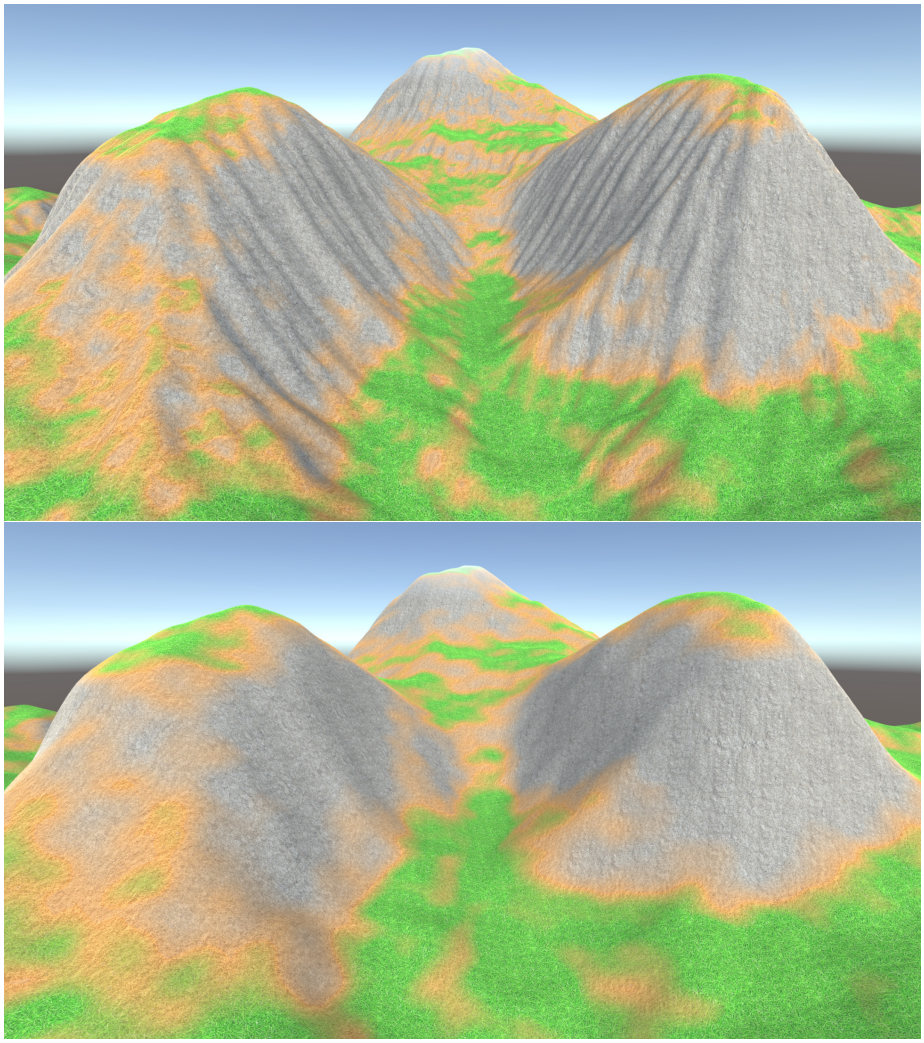


Figure 3.13: The first image shows a terrain deformed by my hydraulic erosion generator, while the second one shows the same terrain without the effect.

the last position to the current position and from the current position to the next position exceeds this parameter, the droplet is also terminated. This is done so that the droplet does not circle around the local minima of the terrain. This process produces an array of sample points. Each point has its position and the change in sediment that occurred since the last point.

3.5.2 Texture Generation

The points generated by the first process are used by the second process for drawing the hydraulic erosion texture for all affected chunks. The alpha channel once again marks the areas where the erosion is active. All three color channels adjust the terrain height in a similar way to paths. The algorithm interpolates from one point to another and draws in the chunk textures based on the sediment change in an area defined by the Droplets Width Of Effect

parameter. An example of a hydraulic erosion texture can be seen in Figure 3.2.

3.6 Generate Water Component

This component, generated with the initial generation of the terrain, is responsible for creating meshes representing water. I chose the approach of generating individual chunks for water because I wanted the terrain to feature underwater areas. The component can modify the water chunks' parameters, create rivers, and carve terrain. There are a few parameters that can be changed even after generating. Those are:

- Mesh Wrapping - The number of vertices per water chunk row. Same as Wrapping in Chunk Manager in 3.1.3.
- Material - A Unity material that will be applied to all generated water chunks.
- Water Level - The height at which the water level will be located.

Below these parameters, there are three buttons. The first one, "Generate Water Chunks," generates a water chunk for each terrain chunk. The water chunks are set as children of their corresponding terrain chunks in the hierarchy. The second one, "Reset Water Data," resets all generated water textures, changing the water and terrain chunks. The final one, "Delete Water Chunks," deletes all generated water chunks from the scene.

3.6.1 Water Chunks

The water chunks are similar to normal terrain chunks except for the height calculation. The height of each vertex is calculated as a sum of the water level height and the offset from the water texture generated by rivers.

3.6.2 River Generation

The river shape can be approximated by manually adding points on the terrain along the whole river or using a river agent, which connects the start and end points. The generated rivers have high levels of freedom, making generating artifacts possible. The user should always check the whole river after generation and regenerate the river in case of unwanted artifacts. Before generating rivers, the scene should already have water chunks generated.

Control Points

As written above, control points of the river can be placed manually or by the river agent. When using the agent, the user can specify a few parameters.

- Travel Distance - Length of each step of the agent in meters.

- Max Steps Of Agent - The maximum number of steps the agent can take between manually added control points.

This agent combines the local path-finding agent and hydraulic erosion droplet simulation. It connects two control points and uses the movement of water droplets to create convincing river shapes. Like the local agent, it looks around 180 degrees and finds the best next step. The same formula evaluates the candidate locations as the local agent. However, the parts of the value whose weights can be modified in the inspector differ. The parts are:

- Direction To Target - Controls how much the candidate's point direction is close to the direction of the target.
- Height Difference - Controls how much the agent prefers bigger height changes.
- Gradient Direction - Controls how much the agent goes along the terrain gradient.
- Same Direction - Same parameter as in the local path-finding agent. Controls how the agent prefers going in the same direction as the last step.

I recommend not setting the Direction To Target lower than Gradient Direction because the agent can get stuck. The agent can be sent out by pressing the "Connect River" button. A result connecting the control points can be seen in Figure 3.14

Mesh Alteration

The same approach as in roads, hydraulic erosion, and paths is taken. A new per-chunk texture is applied upon generating a river or carving the terrain. However, this texture is also applied to the water chunks. Similar to roads, a Water Max Y Offset needs to be set.

The first tool in the component is terrain carving. Before carving, the user can set the beginning and end widths, which influence the width of the mesh alteration and is interpolated between the first and last point. Another settable parameter is the Carve Depth. This number determines the final height of the terrain, which is calculated as a sum of the water level and the Carve Depth.

The second tool is river generation. The river lowers the terrain vertices and heightens the water chunk vertices in the same area. The user can set several parameters that are doubled. One for the start of the river and one for the end. These are:

- River Width - Controls the area of effect of the river generation.
- River Depth - Controls how deep the river will cut into the terrain.

- Water Level - Controls the height of the water from the river bed.

If the river reaches the water level, the parameter Under Waterlevel Depth is used to determine the height of the terrain the same way as in terrain carving.

The texture channels are used as follows. The terrain chunks use the red channel, where a value of 0.5 means no change, a higher value heightens the terrain, and a lower value does the opposite. The green channel is used to offset the water chunks, where a value of 0 means no change. Higher values offset the height of vertices by multiplying the value with Water Max Y Offset. The alpha channel is once again used to mark the areas where the effect of this component is active.

Users can expand the base water level and create mountain rivers using these two simple tools. An example can be seen in Figure 3.15. Suppose the user is not satisfied with the generated result. In that case, they can use a button called "Reset Mesh Alternation Using Control Points", which resets the water generation using the control points in a radius set by River Width.

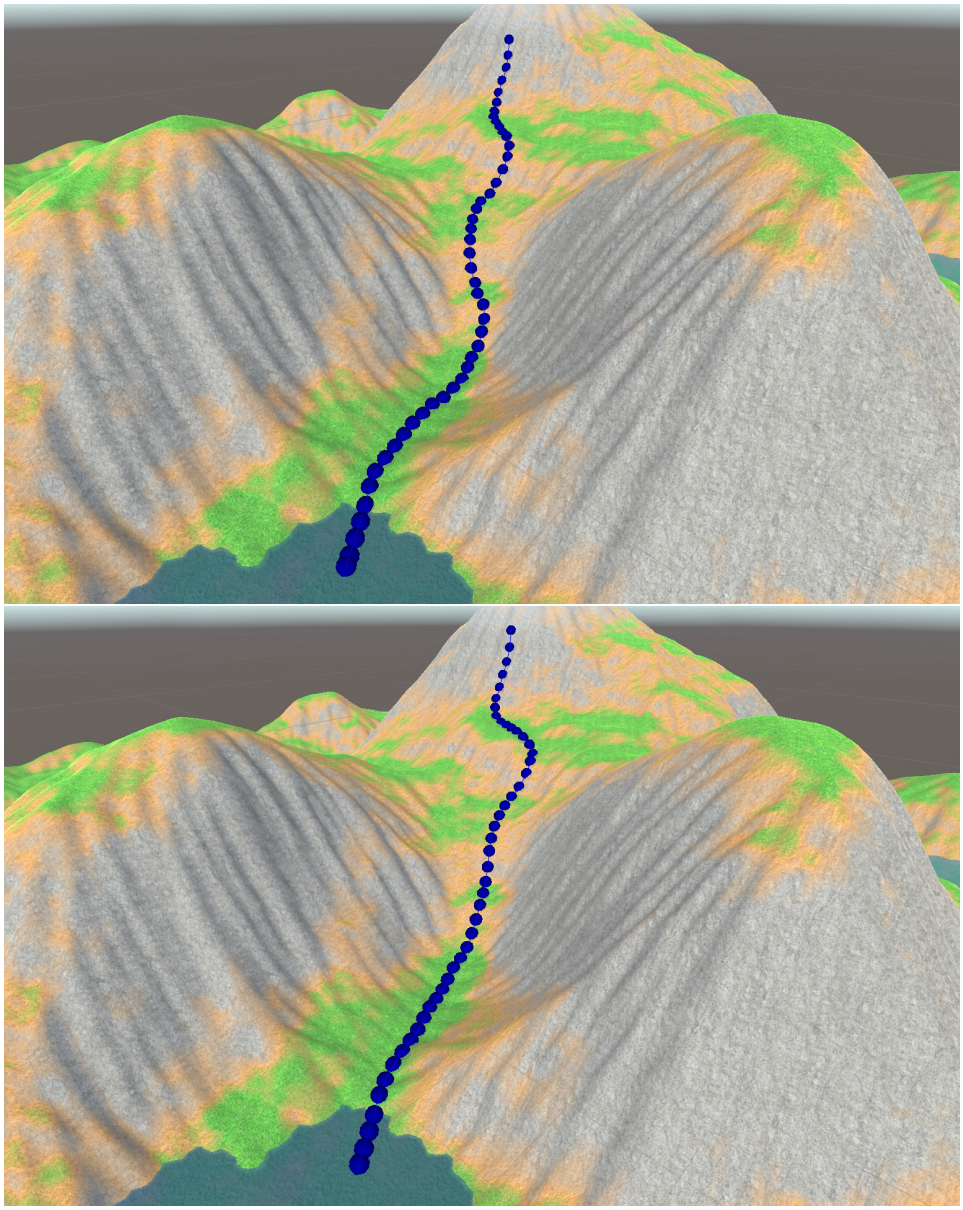


Figure 3.14: The first image shows the river agent's path of control points, which follows the gradient more closely than the one in the second image.

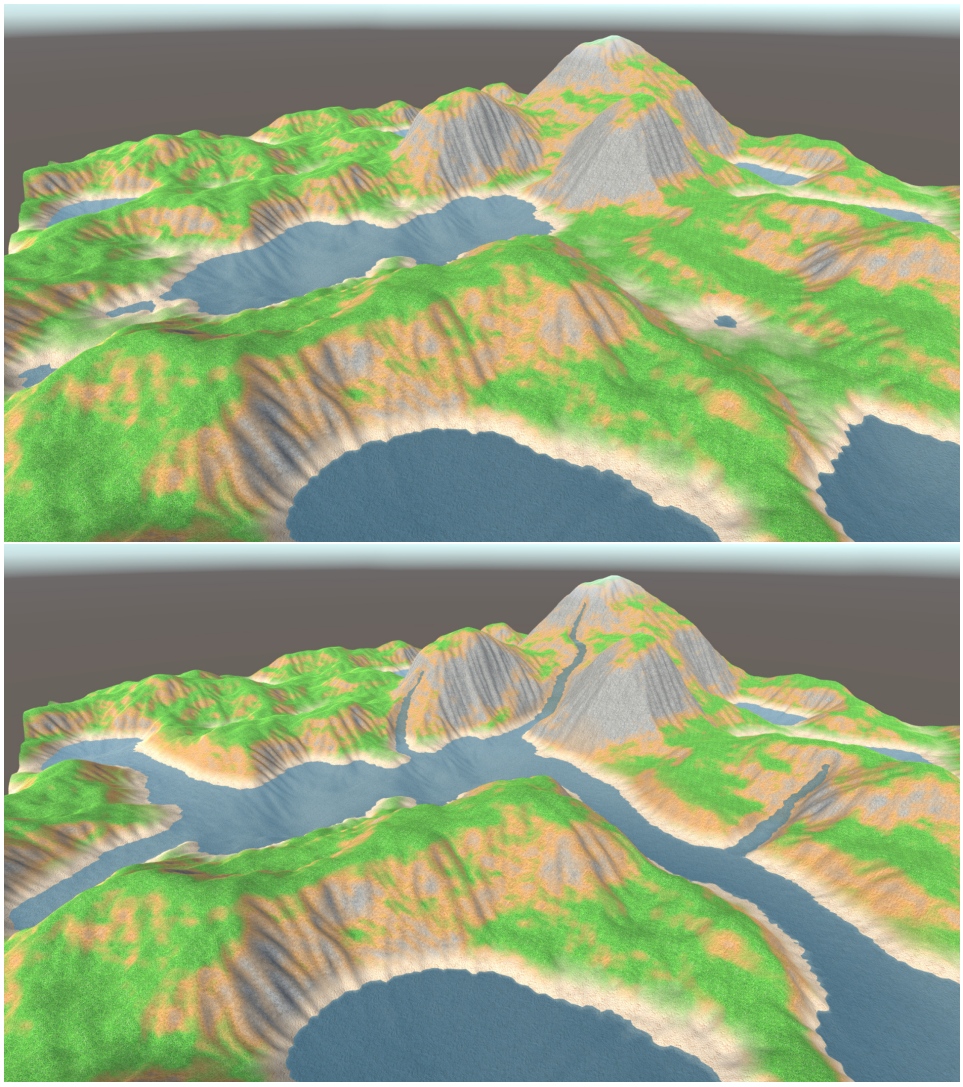


Figure 3.15: The first image shows a terrain with only water chunks generated. The second image shows the same terrain shaped by the carving and river tools.

Chapter 4
Performance

This chapter contains information about the performance of the tool I have created. The measured times mentioned in this chapter were all calculated as the average of 10 measurements and considered only the raw calculation time of the functions I implemented. The real-time it takes to make these changes to the terrain is longer and not reliably measurable due to Unity's scene update and texture post-processing functions. The times were tested on a computer with an Intel(R) Core(TM) i5-8400 CPU, 16 GB of RAM, and NVIDIA GeForce RTX 3060 GPU.

4.1 Base Terrain

The generation of the terrain is pretty optimized. Changing parameters for highly detailed meshes is slow and could be made faster by optimizing the height-calculation function or running it on the GPU. That is why changing the parameters using a low-detailed terrain version is recommended. I measured these times using 100 chunks, which all had the number of octaves set to 6. Using more octaves does not have much influence over the detail of the terrain, and using fewer has only a small influence on the performance. All the measured times can be seen in Table 4.1 and in Figure 4.1.

Wrapping	Vertices	Generation Time [ms]	Update Time [ms]
11	12 100	375	172
25	62 500	797	607
51	260 100	2 191	2 027
75	562 500	4 255	4 107
101	1 020 100	7 269	7 057

Table 4.1: A table showing the time it takes to create and edit a terrain with 100 chunks and 6 octaves of Perlin noise.

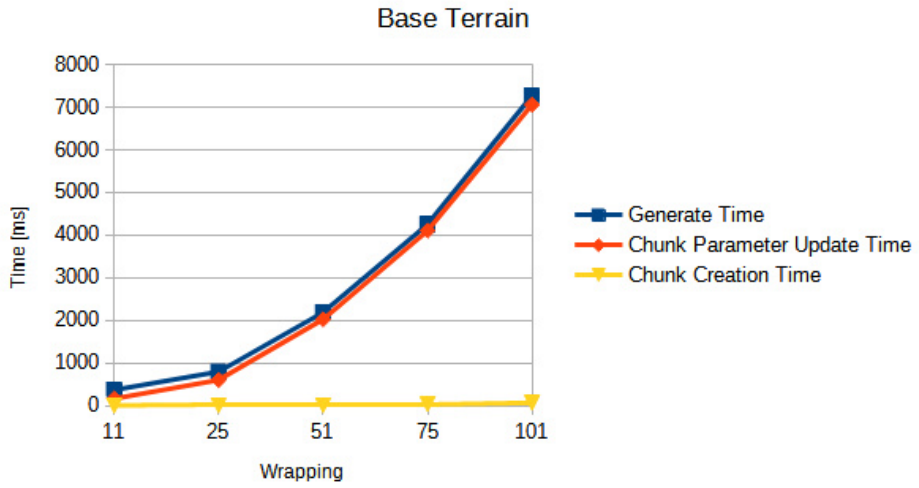


Figure 4.1: Graph showing the generation and parameter change times of the base terrain from Table 4.1.

4.2 Generate Path Component

The connection of paths is fast. I tried measuring the time using different terrain and parameters. However, it was always around 200 milliseconds, which is why I did not put it in a table. The global agent's generation of a NavMesh takes up most of the connecting time. This could only be optimized if I implemented my own global path-finding and did not use the Unity NavMesh system. The time it takes to generate paths of different lengths and the time it takes to generate random paths, which can differ based on the success rate, can be seen in Table 4.2.

Path Length [m]	Generate Time [ms]
50	115
150	235
Number of Random Paths	Generate Time [ms]
5	528
10	989

Table 4.2: A table showing the time it takes to generate a path.

4.3 Generate Road Component

The road generation is good from the performance side. The roads do not feature any agent, so the connection time of control points could not be measured. The process that takes most of the generation time is drawing the textures. It could be further optimized by not setting each pixel of the texture separately using the Texture2D component's function SetPixel(). However, I

could make it work in time for the submission of the thesis. The measured times can be found in Table 4.3 and in Figure 4.2.

Road Length [m]	Total Width [m]	Generate Time [ms]
50	8	990
50	16	2 764
100	8	1 585
100	16	5 149

Table 4.3: A table showing the time it takes to generate roads based on their length and width.

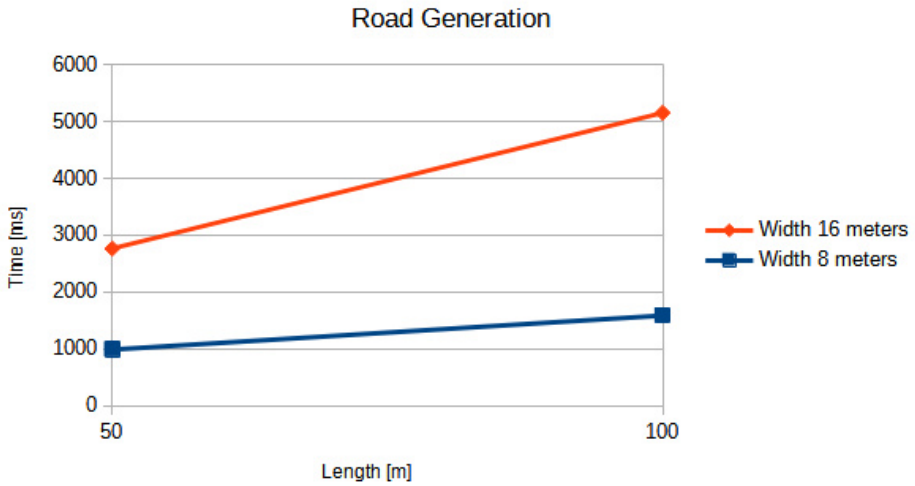


Figure 4.2: Graph showing the generation times of roads from Table 4.3.

4.4 Generate Hydraulic Erosion Component

The generation of hydraulic erosion was tested on a terrain consisting of 25 chunks. The droplets had maximum steps set to 100 and the drawing width to 3 meters. The overall performance of the generation is not good. The first part, which is the simulation of the droplet paths, is fairly fast and can be tested in real-time using manual droplets. However, the second part, texture drawing, could be optimized. It suffers the same problem as roads. The measured generation times can be found in Table 4.4 and in Figure 4.3.

Droplets per Chunk	Generate Time [ms]
81	7 938
169	15 384
225	20 278

Table 4.4: A table showing the time it takes to generate hydraulic erosion based on the number of droplets.

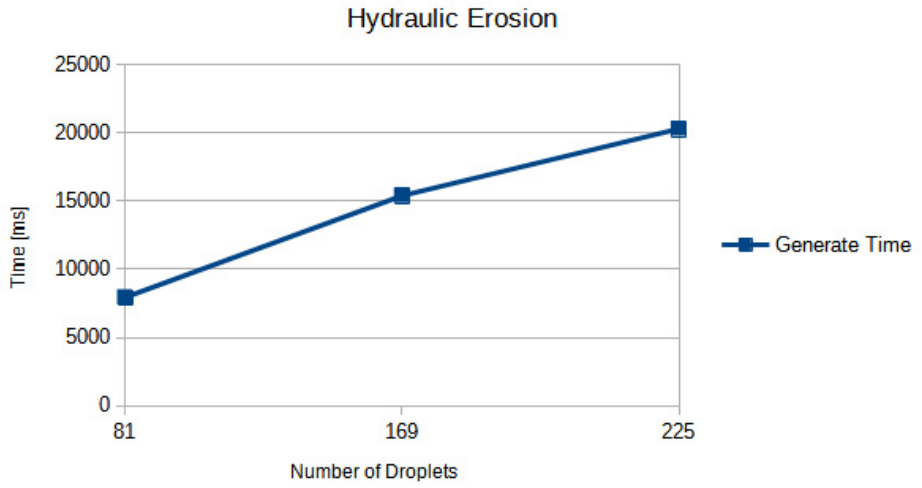


Figure 4.3: Graph showing the generation times of hydraulic erosion from Table 4.4.

4.5 Generate Water Component

This component has several measurable parts. The first one, water chunk generation and editing of their parameters is fast. It was tested on a terrain consisting of 100 chunks. An optimization would be available the same way as normal terrain chunks, by calculating the height on the GPU or using multithreading. The second one, river generation, and terrain carving, are a bit slower but still performant. The times for terrain carving and river generation are the same since the most time-consuming part, texture drawing, is done the same way in both of them. This can be optimized using the same method as mentioned in previous sections. All the measured times for water chunk generation and editing can be found in Table 4.5 and in Figure 4.4. The times for river generation are in Table 4.6 and Figure 4.5.

Wrapping	Generate Time [ms]	Parameter Change Time [ms]
31	239	1
61	690	52
91	1 409	118
121	2 449	211

Table 4.5: A table showing the time it takes to generate water chunks and change their parameters based on the wrapping of the water chunks.

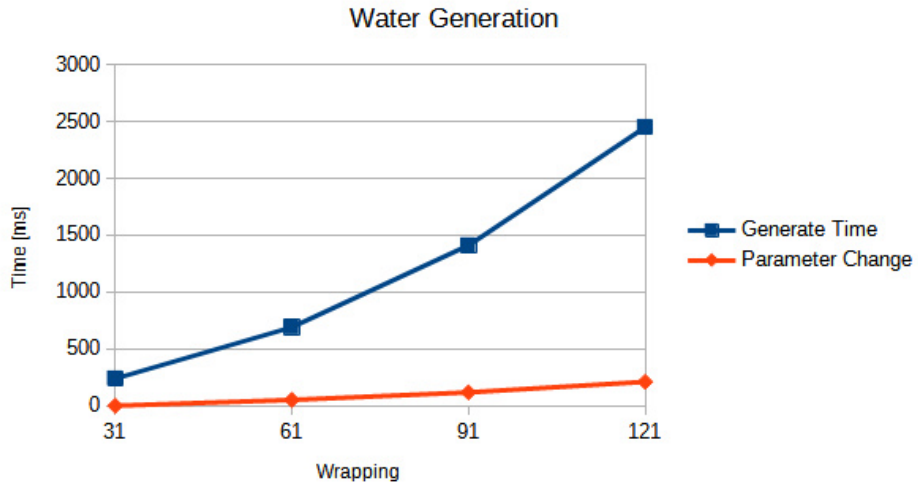


Figure 4.4: Graph showing the generation and parameter change times from Table 4.5.

River Length [m]	Generate Time [ms]
50	1 453
100	1 751
150	2 073

Table 4.6: A table showing the time it takes to generate rivers on the terrain or carve the terrain.

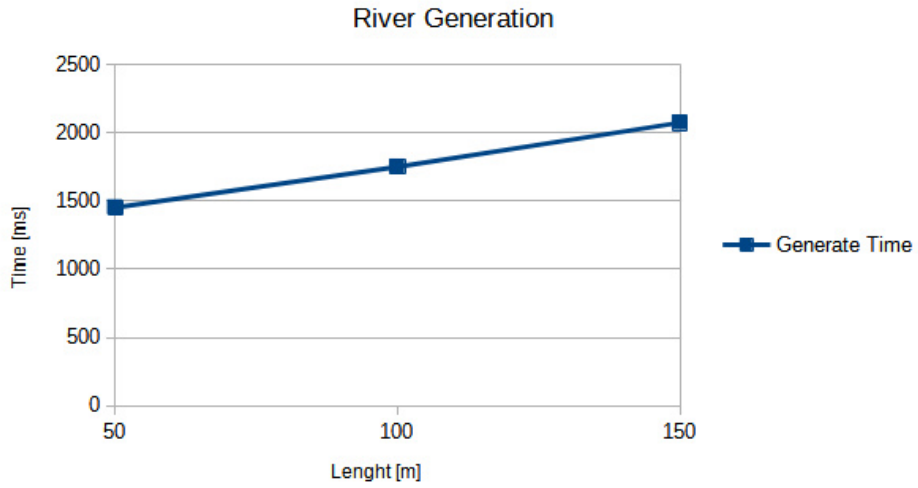


Figure 4.5: Graph showing the generation times of rivers from Table 4.6.

4.6 Discussion

Many areas could be further optimized. However, that would require rewriting a lot of the code. I could not do that due to time constraints. The main area that could be optimized is the texture drawing for each generator. Another optimization would be to calculate the heights of vertices on the GPU. Right now, the tool is not well suited for large landscapes. Considering this, the tool can still be used even on slower computers if the user works with a less detailed terrain in the editing phase.

Chapter 5

Photo Recreation

Three real-life photos need to be recreated using the tool I implemented as part of the bachelor thesis assignment. While photo recreation is not the goal of the tool, it is used here to demonstrate its capabilities. These photos were taken from the Pexels website [18]. I tried to choose photos of landscapes best suited for my tool and its features. However, the tool is not properly suited for photo recreation and lacks important features such as the generation of rocks, trees, etc. It would also be beneficial for photo recreation to have more freedom in terrain editing. However, the results are satisfactory.

The recreation process was as follows. First, a general terrain was created using Generate Terrain component. Then the camera was placed and rotated as in the photo. The Perlin noise of the terrain was set low to make only the small details visible. Then the general shape of the terrain was created by height-effecting chunks. Next, water chunks were generated, and their height was set. After that, the rest of the generators, hydraulic erosion, roads, paths, and rivers, were used to make the features of the terrain.

All of the generated landscapes with their real-life counterparts can be seen in Figure 5.1, Figure 5.2, and Figure 5.3.

Figure 5.1: The first image shows a real-life photo [19]. The second image shows the landscape from the first picture recreated using my tool.

Figure 5.2: The first image shows a real-life photo [20]. The second image shows the landscape from the first picture recreated using my tool.

Figure 5.3: The first image shows a real-life photo [21]. The second image shows the landscape from the first picture recreated using my tool.

Chapter 6

Conclusion

There are many different methods for procedurally generating terrain, which differ in the generated landscape's believability. To produce a believable terrain, multiple approaches ought to be used. One of the most prevalent methods is noise generation, such as Perlin or Simplex noise. Multiple octaves of these noises are combined with diminishing scale and amplitude to create the terrain's general shape as well as small bumps and hills. The final landscape is then deformed by physical processes such as hydraulic erosion or agent-based approaches for creating specific features. Terrains created by this process can look very realistic and natural. However, the amount of control the user has over the features of the landscape is very limited and not intuitive.

My tool implements this terrain generation method chunk-like, aiming to give the user more control. This is done by separating the features into independent generators with many parameters, where each feature can be easily added, altered, or deleted. This is possible thanks to a texture-based approach, where each chunk has a texture for each generatable feature. This texture dictates the height changes for all vertices. Additionally, most generators give users even more freedom by letting them decide where the feature will be generated, using easily-usable control points.

In base terrain generation using Perlin noise, the user can change the number of octaves of Perlin noise with changeable scale and offset, lacunarity, persistence, the amplitude of the landscape, the number of points in a line, etc. More chunks can be generated at any time, may the user need it. Furthermore, height-effector chunks can also influence the terrain to create large mountains or low valleys.

The terrain uses a complex shader to color the terrain based on steepness, altitude, and the textures created by the generators. These textures are interpolated on the edges, which the user can control. This shader also includes the generation of grass, which moves with the wind.

The tool implements a path generator. Using this, the user can create winding, rule-obeying, human-made paths with a high degree of freedom. Two agents are deployed to help connect the paths and enforce the given rules. One uses global path-finding to create the shortest path from the start to the goal. The other one connects these spread-out points using local path-finding.

The resulting paths can flatten the terrain to create the effect of humans using them.

Another generatable feature is roads. These work similarly to paths, except they follow the control points instead of the terrain. The user can easily modify the coordinates of the control points to make the roads carve into the terrain and even create causeways. It is possible to generate road lines with a defined color.

A big part of this thesis was explaining physical processes and how they create a more believable terrain. That is why I implemented a basic hydraulic erosion generator. It creates water droplets that traverse the terrain along its gradient. As they travel, they take a part of the terrain with them and deposit it in the lower regions. The quality of my implementation is not great, but it adds a nice visual effect.

The users can also generate water, represented as a separate mesh similar to normal terrain chunks. The detail, material, and height of the water are settable. This is done using the water generator, which features two tools. One is responsible for terrain carving, which can lower the terrain and enlarge the water bodies at the water level. The other is the generation of rivers, which creates downwards-flowing water streams using an agent. This agent is a mix of the local path-finding agent from the path generator and the droplet simulation from the hydraulic erosion generator.

The tool can be used to create natural landscapes for video games. Photo recreation is also possible with my tool but difficult. Many areas could be improved upon, but all in all, it is a reliable and easy-to-use tool for landscape generation.

Appendix A

Electronic appendix content

The appendix contains the Unity project in the "src/" folder with all the necessary files to open the project. Alternatively, the project can be found on my GitLab [23]. The link for the GitLab is in the file "README.txt" alongside some basic information about the project. The folder "latex/" contains the latex project with the written document, and the folder "images/" contains all the images used in the thesis.

Bibliography

[1] Jason Bevins. *libnoise glossary, https://libnoise.sourceforge.net/glossary/*, 2003

[2] Ruben Michaël Smelik and Tim Tutenel and Rafael Bidarra and Bedrich Benes. *A Survey on Procedural Modeling for Virtual Worlds*, 2014

[3] Sebastian Lague. *Procedural Landmass Generation Youtube Series, https://www.youtube.com/watch?v=wbpMiKiSKm8&list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3*, 2016

[4] Petr Lhota. *Editor Virtuálních Světů*, Bachelor Thesis, 2022

[5] Unity Technologies. *Unity Glossary, https://docs.unity3d.com/Manual/Glossary.html*, 2023

[6] Sebastian Lague. *Coding Adventure: Hydraulic Erosion, https://www.youtube.com/watch?v=eaXk97ujbPQ*, 2019

[7] Shaker, Noor and Togelius, Julian and Nelson, Mark J. *Procedural content generation in games*, Springer International Publishing, 2016

[8] lewellen. *Algorithms for Procedurally Generated Environments, https://antimatroid.wordpress.com/tag/midpoint-displacement/*, 2015

[9] Wikipedia, Navarras. *https://commons.wikimedia.org/wiki/File:Climate_influence_on_terrestrial_biome.svg*, 2017

[10] Jonathon Doran and Ian Parberry. *Controlled Procedural Terrain Generation Using Software Agents*, 2010

[11] Roystan. *Grass Shader Article, https://roystan.net/articles/grass-shader/*, 2019

[12] Martin Wantke. *Flow chart of a 3D graphics rendering pipeline, https://en.wikipedia.org/wiki/Graphics_pipeline#/media/File:3D-Pipeline.svg*, 2021

[13] Daniel Ilett. *Six Grass Rendering Techniques in Unity, https://www.youtube.com/watch?v=uHDmqfdVkak*, 2022

[14] Jasper Flick. *Tessellation Subdividing Triangles, https://catlikecoding.com/unity/tutorials/advanced-rendering/tessellation/*, 2017

[15] Tommy Poulin. *Shader Rand() Function, https://forum.unity.com/threads/am-i-over-complicating-this-random-function.454887/#post-2949326*

[16] Polycam. *Polycam Material Generator, https://poly.cam/material-generator.*, 2022

[17] Andrey Sitnik and Ivan Solovev. *Easing Functions Cheat Sheet, https://easings.net*, 2020

[18] Pexels. *Free Stock Photos, Royalty Free Stock Images & Copyright Free Pictures • Pexels, https://www.pexels.com*, 2014

[19] Daniel Santos. *Photo of Lake and Rocky Mountain Under Cloudy Sky , https://www.pexels.com/photo/photo-of-lake-and-rocky-mountain-under-cloudy-sky-4215909/*, 2019

[20] Nextvoyage. *Green Mountains Under Blue Sky and White Clouds , https://www.pexels.com/photo/green-mountains-under-blue-sky-and-white-clouds-4061011/*, 2018

[21] Ziauddin Refah. *Brown and Green Mountain Range Under Blue Sky, https://www.pexels.com/photo/brown-and-green-mountain-range-under-blue-sky-1461380/*, 2018

[22] Jasper Flick. *Triplanar Mapping, https://catlikecoding.com/unity/tutorials/advanced-rendering/triplanar-mapping/*, 2018

[23] Ondřej Kyzr, *GitLab project of this bachelor thesis, https://gitlab.fel.cvut.cz/kyzrondr/bachelorthesis*, 2023

[24] Hello Games, *No Man's Sky Beyond Development Update, https://www.nomanssky.com/2019/08/beyond-development-update/?cli_action=1684788247.408*, 2019

[25] Mojang Synergies AB, *The Offical Minecraft website, https://www.minecraft.net/en-us*, 2023

[26] JoeBroesLL, *Image of a sunset over a procedurally generated terrain in the game Cube World, https://steamcommunity.com/sharedfiles/filedetails/?id=1871928236*, 2019