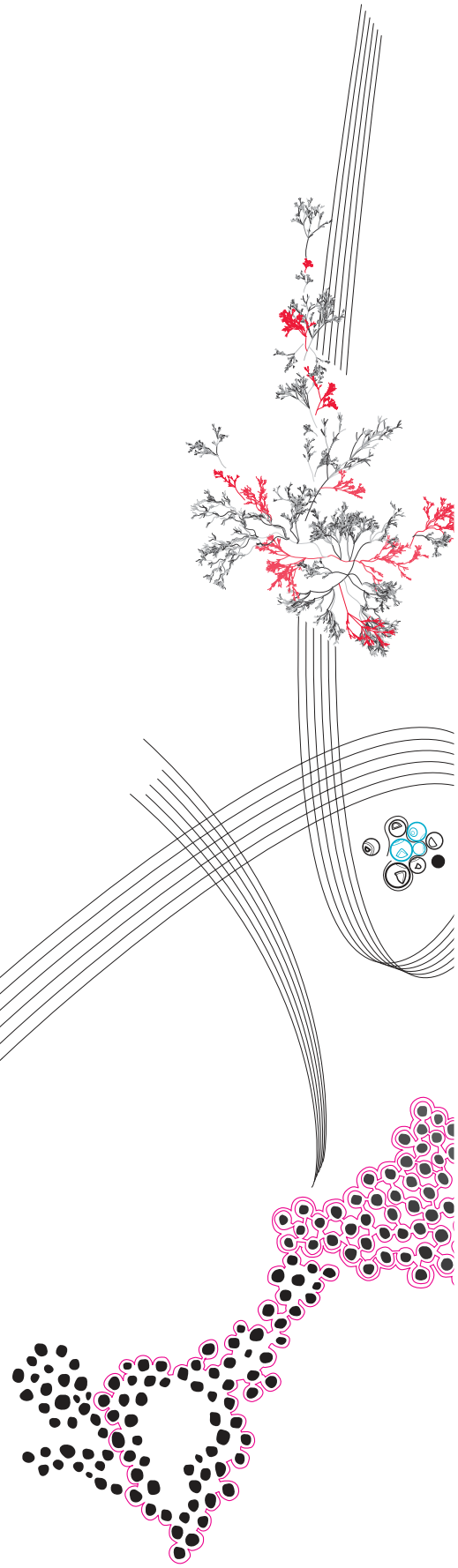


MSc Computer Science
Final Project

# Procedural generation and validation of fixed-viewpoint digital nature landscapes

Lars van Arkel

Supervisors:
V. Zaytsev
M. Gerhold
M. Gómez Maureira

March, 2025

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

# Abstract

In the landscape of procedural content generation, there is a large focus on the generation of large-scale landscapes. As part of a research project into the psychological effect of digital nature, there is a need for small-scale environments visible from a single point of view on the ground. This paper proposes a framework to generate a landscape with rivers and vegetation, as well as how the different parameters in each generation step influence the overall terrain. In addition to this, multiple methods are defined to analyse certain aspects of the terrain and used to test the system. These aspects relate to the visibility of different components of the system from the perspective of the camera, such as the presence of rivers and the obstruction of the view by single large objects.

# Contents

# 1 Introduction

The use of digital natural landscapes is common in a number of different applications. For decades, video games have created landscapes wherein the player is immersed with varying levels of realism, but even outside of entertainment digital nature can be used for more serious applications. In previous research, it has been shown that digital nature has a positive effect on mental and social wellbeing [36], and can function as a substitute for real natural environments when these are not accessible. In previous research into the effect of digital nature, most environments have been hand-crafted by a designer. The process of creating these environments can be time consuming and has to be done each time a different environment needs to be created.

Instead of manually designing virtual environments, Procedural Content Generation (PCG) can be used to automatically generate certain aspects of the desired terrain and can reduce or remove the workload necessary for generating realistic virtual environments. When generating landscapes, one important aspect is the trade-off between the control the user has over the generated terrain and the amount of effort required to create the environment. This thesis analyses the parameters regularly used in the generation of procedural environments and what effect these parameters have on the system. To accomplish this, we have designed a system capable of generating environments consisting of a number of terrain features often found in nature environments. This system can generate digital landscapes that will be viewed from a single point of view. In addition to creating these environments, we have looked at multiple criteria of the resulting landscape to find qualitative evaluations of the generated terrain.

We have created an implementation of the system in the Unity game engine[1]. Unity is a widely used game engine for its ease of use, and is already used in other tools and programs designed to use digital nature environments.

# 2 Problem statement

The Growing Roots Research project[2] was a collaboration between the University of Twente, Amsterdam University of applied sciences, the serious gaming company TexTown Games and several health institutions. This research project aimed to reduce loneliness in people who have limited access to the real world by using the digital world [36]. This research into digital nature has focused on how different types of environment influence the mood of someone who experiences digital nature in immersive ways, such as virtual or augmented reality. The project looked at the difference between dense and sparse environments, or what different mood a user has when looking at landscapes with a lot or no human influence. To support the integration of digital nature in people's lives, TexTown Games has developed a device called the "virtual window"[3]. This device consists of a camera and a Kinect sensor that tracks the position of a person in front of a monitor and projects a view of a digital nature landscape on the screen based on where the person is standing. This gives a parallax effect that makes the user feel like they are watching through an actual window. In the research project, only hand-made landscapes were used when analysing the effect of different kinds of environments on the mood of the user. In the future, more research on the effect of different types of environment will require additional landscapes that have visually distinguishable features, such as the density of vegetation or the colour and types of plants used. Creating different landscapes by hand takes a significant amount of time, so the ability to automatically generate terrains based on different requirements and parameters is required.

Previous research on procedural landscape generation has focused mainly on generating large-scale environments. The virtual window shows a small-scale view of the environment, which means that the camera stays in a single location and shows a small part of the environment. This limits the size and amount of terrain that needs to be generated, but instead requires a higher level of detail. This research will focus on the generation of a small-scale environment based on a single point of view.

[1] Unity Real-Time Development Platform. From https://unity.com/

[2] Growing Roots. From https://web.archive.org/web/20240515061910/https://growingroots.nl/, archived at 15-5-2024

[3] Design2Connect - TexTown Games. From https://textowngames.nl/portfolio/design2connect/, retrieved at 10-1-2025

As further research into the effects of digital nature will focus on the effect of different terrain features, the researcher needs control over the terrain they want to create. This control should allow the user to select broader features of the terrain, such as the kind of ecology the terrain represents, and finer features, such as the density of vegetation or ruggedness of the ground.

As the system will be used by people who are not necessarily designers or programmers, the influence of different parameters of the system should be clear. In addition to this, the ability to validate whether the generated terrain meets certain requirements of the study should be included. We will therefore look at different ways to test the generated landscape.

With TexTown Games, an initial set of features was created. From this set of features, we have selected the terrain itself, rivers, and different vegetation that can be placed on the terrain as initial features of the system, and the scope of the research. Instead of needing to generate tree or plant models, TexTown Games has provided a Unity asset package that contains a large number of European vegetation[4]. This thesis will use these assets in all screenshots showing the textured terrain or any placement objects.

The virtual window already uses an environment created in Unity, and because of the large amount of documentation and assets for Unity, the system will also be created in Unity.

# 3 Research questions

To generate distinct environments that can be used in future research in digital nature, the following research questions are established. By answering these research questions, we show how a digital nature landscape can be generated, how a user that generates landscapes can influence the resulting product, and how different metrics can be created to test the resulting environment.

RQ 1: How can we generate each component in a landscape, and how do these components interact to create a landscape suited for a single point of view?

- RQ 1.1: How can we generate each component required for the landscape?
- RQ 1.2: How can we connect all components to create a landscape for a single point of view?

RQ 2: What are the influences of the different parameters on the resulting landscape?

RQ 3: In what ways can we validate the procedurally generated terrain?

- RQ 3.1: How can different requirements be translated into rules to validate the terrain?
- RQ 3.2: How can the terrain generation be influenced by the validation to create terrain conforming to certain rules?

[4]European Vegetation Pack Two, from https://assetstore.unity.com/packages/3d/vegetation/european-vegetation-pack-two-151939, retrieved at 3-3-2025

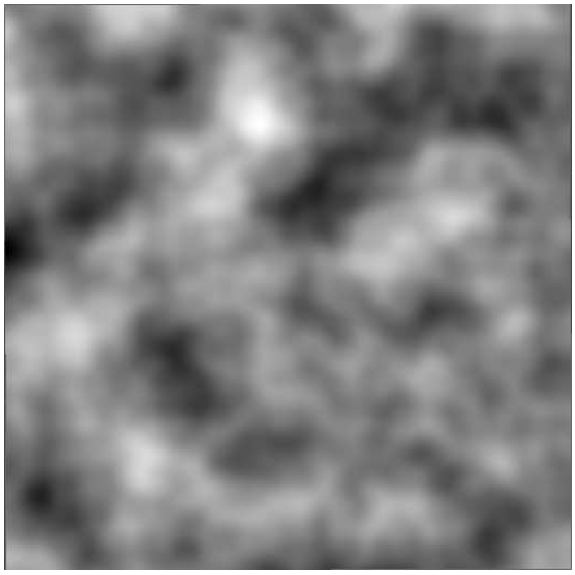


FIGURE 1: A height map generated by fractal Perlin noise.

# 4 Background

The generation of a virtual landscape can be split up into multiple components that are connected to create a complete landscape. Firstly, the terrain itself needs to be created, after which the other components are generated and interact with the terrain to create the complete environment. The following sections describe previous research and techniques for generating individual components, as well as different ways that are used to connect these components.

## 4.1 Generation of terrain

The topic of procedural terrain generation has been researched for several decades. A 2-dimensional height map, represented as a greyscale image, is the most common technique for generating terrain. When converting this height map to a 3-dimensional terrain, a grid of vertices uses the height map to scale each vertex of the terrain to a specified height. This approach is widely used due to its simplicity but has some downsides. Because the height map is projected onto the plane, the terrain cannot have multiple sections of terrain that overlap each other, such as overhangs or caves. Although some solutions support overhangs [11], they usually require solutions tailored to the desired terrain.

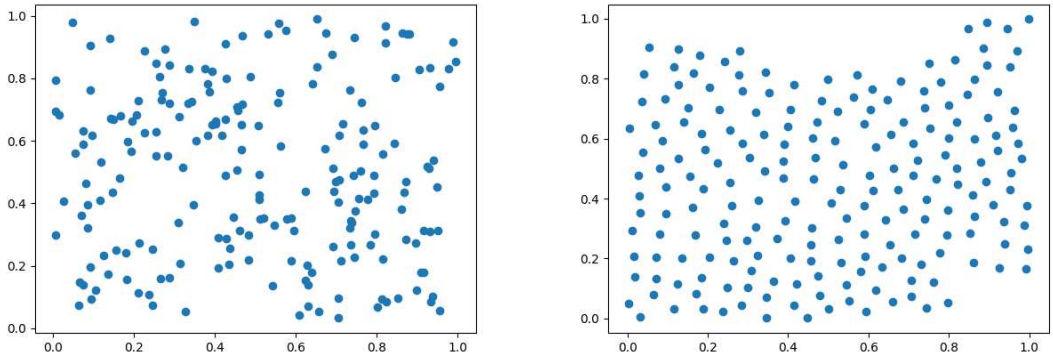
### 4.1.1 Fractal terrain

A popular approach to generate procedural terrain is to use fractal algorithms. In this approach, the height of a terrain is evaluated using multiple noise functions with different frequencies to generate a terrain that contains large-scale differences, as well as small-scale ruggedness [30]. An example of a commonly used noise function is Perlin noise [24]. Because noise functions only depend on the input coordinates, these algorithms can be easily parallelised on GPU-based implementations [29]. An example of a height map generated by fractal noise is shown in Figure 1. The dark colours represent low elevations, and bright colours represent higher parts of the terrain.

### 4.1.2 Rivers

The generation of rivers is directly related to the terrain itself. Rivers have the requirement to be placed in a landscape, such as that the direction of the river should move along a flat part of terrain or go downwards.

One way of generating rivers is to generate the rivers and elevation at the same time. Belhadj and Audibert [1] generate a network of rivers and ridges and use this network combined with a




(A) Random distribution (B) PDD distribution

FIGURE 2: Comparison of Random and Poisson Disk Distributions for 200 samples

modified midpoint displacement algorithm to generate the rest of the terrain. After ridges are randomly generated, points are placed on ridges and simulate gravity to roll down the slope. With this approach, rivers show realistic behaviour connected to the elevation of the ridges.

If the terrain consists of both mountains and coastlines, a river can also be generated by finding a random point on the coast and a random point on the mountain, and generating a random path between these points [10] [27].

Alternatively, rivers can be generated by placing several control points on the map and interpolating between these control points to generate a path [32]. To place the river path onto the terrain, a 2D cross section can be used to extrapolate a shape for the entire path [14]. In this paper, the cross section is also used to add vegetation and textures to the river.

## 4.2 Placement of vegetation

When generating a landscape representing nature, an important step is placing vegetation in ways that are realistic and visually appealing. Adding vegetation to a landscape consists of different kinds of elements, from small vegetation such as grass and small bushes to large structures such as trees. The following sections describe multiple ways of placing and distributing vegetation.

### 4.2.1 Distributed placement

An easy way to place vegetation is by planting each plant randomly. As each plant or tree is a physical object, trees cannot be placed too close to each other or they would overlap.

One way to solve this is to specify a boundary radius and place vegetation randomly in locations where the closest plant is further away than that distance. Although this works, Casey Muratori showed in a blog post that this naive approach is inefficient when the density of plants increases [23]. A more efficient way to generate random positions is to use Poisson Distribution Disks (PDDs). A Poisson distribution disk allows patterns to be generated where each point has a minimal distance to another point, as shown in Figure 2. Efficient algorithms use Voronoi diagrams to create these patterns with a time complexity of $O(N \log N)$ [16], while another algorithm generates points randomly around already generated points to create the same effect, but with a time complexity of $O(N)$ [3]. Another algorithms, based on fluid simulations, uses a smoothing kernel which moves an initial set of points so that the standard deviation of the distances between points is lowered [15].

Another way of distributing points is to have an initial set of points, for instance, points on a grid, and to randomly move each point around its initial location. Although simple, this technique has been used to create points that look seemingly random [13] [37].

## 4.3 Connecting components

The previous sections have elaborated on different components used in procedural terrain generation. To generate terrain that resembles a landscape, these features must be combined. The following section describes different ways that these components can be connected, and how the requirements of each component interact with other components.

### 4.3.1 Level of detail

When placing vegetation, there is a difference between placing large trees and smaller bushes and grass. Trees are larger and the branches prevent other trees from being placed too close to each other. For bushes and grass, the minimum distance is much smaller, as they can also grow under the branches. In their paper, Hammes defines eight different layers that each cover a different type of vegetation on a different scale [13]. For a large scene, this can be used to reduce the complexity of areas that are further away and increase the detail for closer sections of terrain. To resolve placement conflicts between layers, do Nascimento added a function to calculate the zone of influence of objects on a higher layer, which acts as a filter for objects on a lower layer [9].

### 4.3.2 Probabilities and masking

When placing vegetation on a terrain, objects cannot be placed on certain areas. For instance in water, or on slopes that are too steep. There have been multiple ways proposed to resolve these issues. When categorising the terrain into ecotypes, as done by Hammes [13], parts of the terrain that are unsuited for vegetation simply will not include any vegetation at all. Alternatively, Olsen assigns placement scores based on flatness and connected areas that allows placing buildings aimed at an Real-Time Strategy (RTS) game [26]. Larger structures can therefore only be placed on areas where the slope is similar for all tiles under the structure.

Alternatively, when the terrain is generated using a 2D height map, other 2D maps can be used to mask the placement of vegetation. In the video game Horizon Zero Dawn maps that represent water, roads and trees are combined to provide masks that can be used to generate other types of vegetation [37]. Instead of using height maps, combining different probability functions can yield a similar result [39].

# 5 Related work

## 5.1 Terrain generation

### 5.1.1 Volumetric terrain

Instead of representing the terrain as a 2-dimensional height map, it can be generated using a volumetric data structure, such as a 3-dimensional grid [8]. With this approach, the generated terrain depends on whether the volumes of each location in the grid are filled. This technique is popularly referred to as voxels, with most famous example that uses a voxel-based terrain being the video game Minecraft [5], which uses voxels to create a crude, blocky terrain. Voxel-based terrains can also be transformed into smooth terrain using techniques such as the surface nets method [12]. Although volumetric data structures allow for complex structures including caves [5] and overhangs, the limitation of this approach is the larger storage size of terrain data.

### 5.1.2 Alternate terrain generation methods

In section 4.1 the method of using fractal noise to generate terrains. Instead of noise functions, a midpoint displacement algorithm is another way to generate terrain. With this algorithm, a grid gets subdivided into smaller primitives, and the height of each midpoint is updated with a random value. This value grows smaller with each iteration, resulting in small changes in local areas, and large changes in global areas [28]. One type of midpoint displacement algorithm, the

[5] Minecraft. From https://minecraft.net, retrieved on 17-2-2025

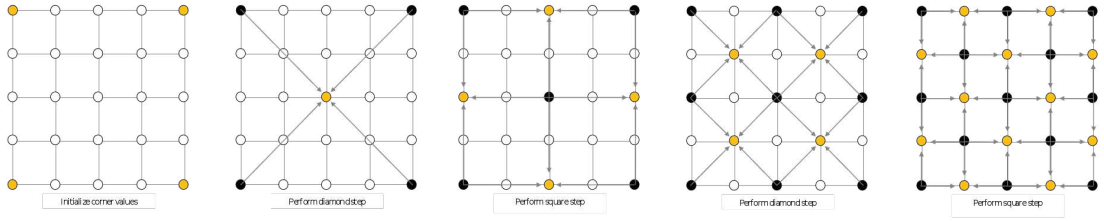



FIGURE 3: Two iterations of the diamond-square algorithm. The yellow dots represent the vertices manipulated in each operation, and the black dots represent the values used in interpolation[6].

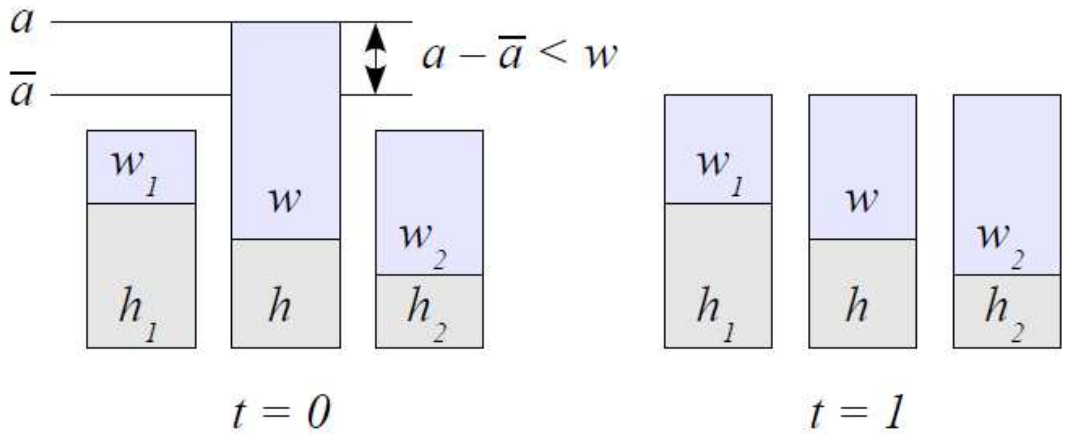



FIGURE 4: An example of water transport in a hydraulic erosion simulation, from [26].

diamond-square algorithm recursively subdivides a grid of points to approximate Fractal Brownian motion [22]. This algorithm is visualised in Figure 3

Instead of using a single approach, the combination of multiple functions, such as adding Voronoi noise to a midpoint displacement map, can also result in different types of terrains [26].

Although fractal methods work well to generate a landscape, the self-similar nature of these techniques do not result in realistic terrains. Because of this, a physical simulation is often used to approximate more realistic terrain. Most techniques follow processes that resemble erosion, which are divided into hydraulic erosion and thermal weathering [24]. These types of simulation are mainly used to enhance the quality of a terrain generated by fractal Brownian motion, instead of being used to fully generate a terrain.

Hydraulic erosion models the use of water to transport sediment from higher elevations to lower locations. This can be used in conjunction with rivers, or by simulating rainfall and water transport. There are several techniques for this simulation, implemented on both the CPU and GPU [2] [40]. Hydraulic erosion works by discretizing the surface, where each chunk contains the height of the terrain, as well as the amount of water contained in the chunk. Updating the state of the simulation is done using Cellular automata, with each iteration interchanging water and sediment between neighbours [6]. An example of a step in this algorithm can be seen in Figure 4.

Thermal erosion describes the way that material is eroded by several processes that break down the terrain, from which the material moves down a slope to accumulate at the bottom [24]. Similar to hydraulic erosion, it works by using cellular automata to transport soil to neighbouring locations. Instead of using water, the difference in height of neighbours is compared, and if it exceeds a threshold a portion of the terrain is moved to the lower level.

[6]By Christopher Ewin - Own work. Retrieved from https://commons.wikimedia.org/w/index.php?curid=42510593

### 5.1.3 Agent-based algorithms

The previous algorithms manipulate the terrain globally. Another option would be to use software agents to manipulate the terrain on a local scale. In their paper [10], Doran and Parberry have used multiple types of software agents that create coastlines, mountains, and rivers. Each agent has a location on the map and perceives their direct environment. They can modify the terrain using a single objective, such as generating a coastline that connects two randomly generated points, or creating a river connecting the top of a mountain and the coast. In more recent work, software agents were expanded to offer more control to the designer [31]. In other work, software agents were used to generate a road network and place buildings to form a city [21].

## 5.2 Vegetation placement

### 5.2.1 Tiling

Instead of randomly generating positions for the entire terrain, we can place vegetation on smaller tiles, and repeat these tiles on the terrain. To reduce repetitions between tiles, Wang tiles can be used [4]. Wang tiles use adjacency rules to result in an aperiodic layout, which allows the same distribution to be used multiple times without looking like the same repetition.

### 5.2.2 Ecosystem simulation

Another way to place vegetation is to perform an ecosystem simulation. This technique simulates the growth of plants, as well as the competition for resources. It uses L-systems to grow plants based on the self-thinning phenomenon [7] [20]. This suggests that when the density of plants is low, all plants grow without competition. When the population has reached a certain density, plants need to compete for resources, and larger plants dominate smaller, weaker plants. This type of simulation can be done on a single type of plant or on multiple different kinds of vegetation of a similar type, such as different kinds of trees or grass.

## 5.3 Connecting components

### 5.3.1 Agent-based simulation

In a previous section, the use of software agents has been used to create certain parts of the terrain. Software agents work by manipulating the terrain locally, and they can be used to work with the terrain generated by previous agents [10]. For instance, the location that a mountain agent visited to create a mountain can be used by a river agent as a starting location for a river that travels down the mountain. Because agents are executed in sequence, the order of agents allows any constraints from agents to be solved linearly, such as vegetation agents being executed after river agents to avoid placing vegetation in the water.

### 5.3.2 Frameworks

There have been multiple attempts at combining different generation features in a single framework. These frameworks combine different types of terrain generation.

To create landscapes for strategy games, Olsen combined height map generation using noise functions and fractal terrain with an erosion simulation [26]. The system uses an improved algorithm for erosion to improve the execution speed and validates the terrain to check whether it is applicable for use in the strategy game. Some metrics that are used are the presence of flat areas where buildings can be placed, and the connectedness of areas where units can move around. Another framework that generates terrain is the one created by Kahoun [17]. This framework provides a library that creates landscapes with oceans and rivers. This library can generate worlds on a flat plane, or complete planets on a sphere. The world generation contains different kinds of fractal noise generators, as well as a large number of parameters directly influencing the generator.

In another paper, Newlands and Zauner have created a framework that generates terrain with forests [25]. The framework generates vegetation dynamically based on L-systems for trees and 2D billboards for grass. The terrain is generated using 2D simplex noise. Placing the vegetation

in the environment uses an ecosystem simulation, as mentioned in section 5.2.2. The amount of simulation steps and properties of each tree can be configured as a system parameter.

The system of do Nascimento combines terrain, rivers and vegetation [9]. The terrain is generated by providing height and river maps. From these maps, certain features such as the slope and moisture of the ground are calculated. These maps act as a probability distribution for selecting the vegetation to be placed in the area, with each plant having a preference for elevation, slope and moisture. The plants themselves are placed by using a PDD which uses the distribution maps to filter which plants can be added. To place different kinds of vegetation, different levels of detail are used. To prevent overlapping plants of different layers, each layer adds a zone of influence to the distribution maps, lowering the chance of plants being placed too close to other plants.

### 5.3.3 Commercial products

Procedural generation of virtual worlds has been widely used in the video game industry. In large open-world games, the semi-automated generation of environments is necessary for developers to create massive environments. Because of this, there are several commercial products that generate environments that can be used in video games. These tools offer a lot of support for developers to create any environment they want.

One of these tools is World Machine. World Machine is a popular tool for generating large procedural terrains. Users can create environments using a flow chart, by selecting and combining different nodes that modify the height maps of the terrain. These nodes range from nodes that use noise functions to nodes that use erosion simulations. World Machine also allows for specific control of the terrain. Users can draw features such as rivers, mountains, or ridges.

While World Machine only generates terrain and rivers, there are also tools that procedurally place vegetation on the terrain. One of these tools is developed by Guerilla Games and is used in the video game Horizon Zero Dawn[37]. This tool allows artists to combine different feature maps, such as elevation and rivers, to paint a distribution of vegetation on the terrain. With this distribution, pre-made models of vegetation are placed onto the terrain.

## 5.4 Validation

### 5.4.1 Validating terrain

When generating terrain, the combination of different features can have constraints. Some constraints have been mentioned before, such as the overlap between different components, but other constraints, such as the availability of flatness in the terrain, can be a demand of the system as well.

The terrain generation method from Olsen [26] contains several criteria for creating maps usable in RTS games. The two main criteria are areas where units can move around should largely be connected, while the other is that the terrain should be flat enough for buildings to be placed. The model checks this by analysing the slope of the terrain and giving a score for different features of the terrain. The system does not modify the terrain to get a better score, but is only used to verify the quality of a specific terrain.

In the framework created by Smelik e.a. [32], the landscape is created by combining different layers. Each layer represents a different part of the system, such as the vegetation layer containing all plants and trees, and the water layer consists of rivers and lakes. To resolve any conflicts in the constraints of each layer, each feature interacts with the system by either claiming a piece of terrain to use exclusively, or by requesting to modify part of the landscape to solve its constraint. Conflicts arise when multiple features try to claim the same area. When this happens, the system can combine the different claims into a connecting structure, such as creating a bridge when a river and a road attempt to claim the same location. When a connecting structure is not possible, each feature has different priorities with which the system can decide to give the claim to one of the features.

### 5.4.2 Search-based Procedural Content generation

A category of procedural content generation, Search-based procedural content generation (SBPCG), takes a special approach to the generation and validation of terrain. While regular PCG usually contains a test to validate whether the generated content adheres to some properties, SBPCG rates the quality based on a fitness function [34]. To improve the generation, the result of the evaluation is used to improve further iterations, which is usually done using an evolutionary algorithm. This requires a piece of content to be generated multiple times, but it allows designers to provide a qualitative way of validating the generation process.

# 6 System Architecture

To create digital landscapes, we have created a framework that can generate small-scale procedural natural environments that can be used in further research of digital nature. This system currently generates environments with rivers and vegetation. To generate these environments, it can be configured with a number of parameters and pre-generated assets, such as textures for the terrain and models for vegetation. The system therefore does not generate any models for trees or plants, but instead relies on already generated models.

The framework, named Magrathea, is named after the fictional planet-building planet from the novel The Hitchhiker's Guide to the Galaxy, and is implemented in the Unity game engine. It consists of a number of different scripts that create separate components of the environment or combine multiple parts. The system has been created from scratch, without using Unity's Terrain system. The code of the system and a sample scene with all features is available on Github[7].

For each component of the system, different parameters are used to influence the components such that a desired environment is generated. Magrathea is designed to be modular when generating the different components, and is configurable to skip the generation of some components when it is not necessary to generate.

The following sections describe how the system will generate each of the components and how they are combined to create a complete landscape.

## 6.1 Generating components

The proposed system consists of three different components. The first is the terrain. This terrain forms the basis of the landscape on which different features can be placed. The second component consists of rivers that are generated based on a number of control points which are planned onto the terrain. Thirdly, the last component of the system is the placement of vegetation. Vegetation consists of different types of plants, ranging from small bushes to large trees. The generation of each component can be described with a number of parameters. While some parameters would be dependent on the initial settings of the terrain, others can wildly influence the type of landscape created.

In this section, the generation of the different components is described in isolation from each other, and the parameters required for each component. For each parameter, a description is given what the parameter does in the generation of the component and how the system can interact with the parameter.

In the set of parameters for each component, we make a distinction between parameters that are set by the system because they are either constant or implicit based on other systems, and parameters that can be influenced by a designer. In the tables containing all parameters, these are separated by a horizontal line, with the top section containing constant parameters while the bottom section contain parameters that a designer can change to create different types of environment.

### 6.1.1 Terrain

In our design, we have chosen to generate terrain using fractal Perlin noise [24]. The advantage of using noise functions is that the calculation of each separate point can be performed in parallel. Each chunk can be generated independent of other chunks and in any order. The terrain is generated on a two dimensional grid of vertices. For each octave, a separate random 2D offset is generated, and scaled and translated by the scale and position of the vertex in the grid. These coordinates are evaluated using Perlin noise and scaled by the amplitude. The sum of these values is the resulting value in the height map. The terrain is split into different chunks. Each chunk is a smaller piece of the total terrain and has its own height map and mesh. To properly overlap different chunks, the last value of each heightmap uses the same coordinates as the first value of the neighbouring chunk. The result of the generated height maps are shown in Figure 5.

The parameters that influence the algorithm are shown in Table 1. The width and depth of the system are fixed since the resolution of the vertices in a chunk is deemed to be constant. The offset

[7] Magrathea, repository available at https://github.com/LvanArkel/Magrathea

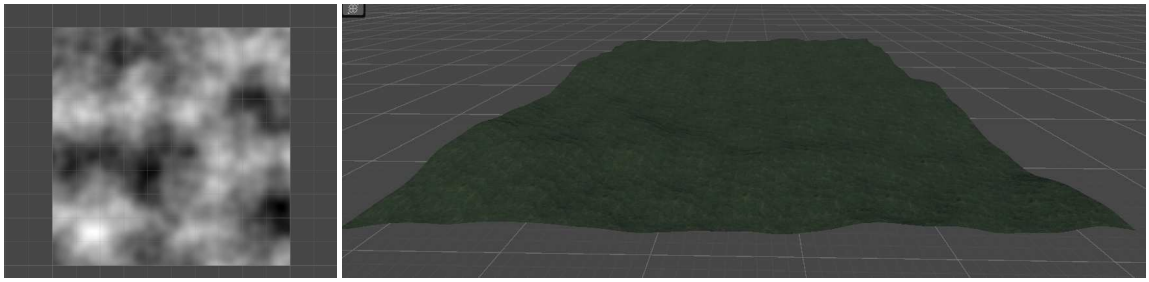


FIGURE 5: Result of the terrain generation as a heightmap (left) and mesh (right)

| Name | Type | Bounds | Description |
|---|---|---|---|
| Width | Int | $\geq 1$ | The width of the heightmap in vertices |
| Depth | Int | $\geq 1$ | The depth of the heightmap in vertices |
| Coordinates | Vec2Int | $\mathbb{Z}^2$ | The coordinates of the generated chunk in the grid |
| Scale | Float | $> 0$ | The initial frequency of the noise on the terrain |
| Octaves | Int | $\geq 1$ | How many iterations of the fractal function is applied |
| Persistance | Float | $> 0$ | How much the amplitude of the noise is reduced multiplicatively each octave |
| Lacunarity | Float | $> 0$ | How much the frequency of the noise is increased multiplicatively each octave |

TABLE 1: The parameters used in the fractal noise algorithm.

and seed are implicit in the execution of the program, since the offset is defined by which chunk will be generated and the seed is constant per generation of the system, to provide the same base noise offsets for all chunks. The current set of parameters is very technical and is based on the mathematical noise functions. These parameters are therefore not intuitive for users not familiar with these noise functions.

### 6.1.2 Rivers

The generation of rivers consist of 2 separate tasks. The first step is to plan the shape of the river. After that, the shape can be combined with a profile to generate the water. To plan the river, we use the same system as used by Teoh [33]. In this approach, a random point is selected as a starting point, and another point on a different side of the terrain gives the initial direction. We then plan a route by placing points a certain distance in a direction that is randomly changed each point. To create a curving, meandering shape for the river, an additional point is placed between each two points, such that the river meanders along the route. To generate the shape of the river, we transform the curve into a fat curve as shown in the system proposed by Huijser [14]. For each point of the curve, the width of the river is used to construct a line perpendicular to the shape of the curve. Connecting the lines between the points of the curve, we create a triangle mesh between the curve points. To create a smoother curve, we interpolate the points using a bézier curve through the curve points. With the fat curve, we can check whether a point is inside the curve, and by using the barycentric coordinates of a point, we can calculate at what position along the cross section of the curve a point belongs. An example of a planned river with its fat curve is shown in Figure 6. The red points are points placed by the random offsets and the cyan points are placed between each regular river point to create the meandering shape. The fat curve follows the width of each control point, which are represented as a dark blue line perpendicular to the curve.

The parameters used to plan the path of the curve are shown in Table 2. The parameters used to create the fat curve are shown in Table 3. The parameters to plan the river can all be modified, while the parameters to create the fat curve can all be kept constant.

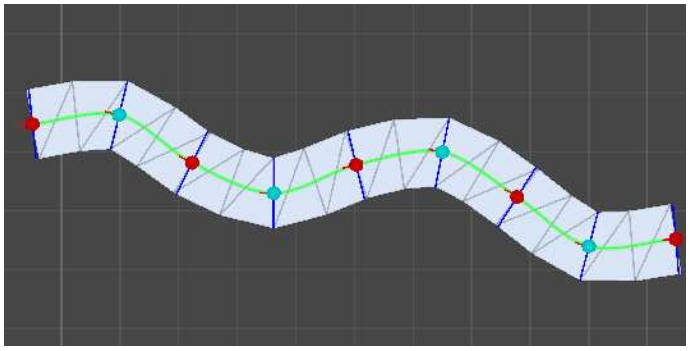


FIGURE 6: Wireframe view of the fat curve of a generated river.

| Name | Type | Bounds | Description |
|---|---|---|---|
| count | int | $\geq 0$ | The amount of rivers to plan. |
| width | float | $> 0$ | The width of the river. |
| segment length | float | $> 0$ | The distance between each river point. |
| max segment rotation | float | $\geq 0$ | The maximum angle that the river curves at each point. |
| meander amplitude | float | $\geq 0$ | The offset of the meandering points. |

TABLE 2: The parameters used for river planning.

### 6.1.3 Placing vegetation

In the system, we introduce two different algorithms of placing vegetation. The first algorithm ensures that any two points are always placed a minimum distance from each other. The second algorithm allows different kind of probability distributions to place objects, but does not always ensure that objects are placed with a minimum distance.

The first method uses the Poisson Disk Distribution (PDD) to generate points that have a minimum distance from each other. To generate the points, the method of Bridson [3] is used. In this algorithm, new points are generated by randomly picking locations around existing points. Points are rejected if another point already exists within a certain distance. We have expanded this initial algorithm so that it can be used with different types of vegetation, where each type has a different minimal distance to objects of another type. This approach can be used to generate multiple different types of vegetation. An example of objects placed using the PDD algorithm is shown in Figure 2b.

To generate different types of vegetation, we separate the types of vegetation into different layers. Similar to the solution provided by do Nascimento et al. [9], each layer represent different types of vegetation, such as trees, bushes or grass. These layers have different rules for how far objects can be placed from another, or to objects in a different layer. For each separate layer, points are randomly generated. Based on the paper of Bridson, we use a 2 dimensional grid to store the points, with the size of each grid cell being equal to $d/\sqrt{2}$, where $d$ is the minimum distance of two points, which is twice the radius of the object on the layer. Using this grid, we can check the existence of nearby points by looking at the neighbours of a cell in the grid. Given an area where the points should be generated, the algorithm places points around already generated points until there is no point that can be placed.

To combine multiple layers generated by the PDD, we generate all layers from large to small,

| Name | Type | Bounds | Description |
|---|---|---|---|
| tangentLength | float | $> 0$ | The size of the tangents of the bezier curve. |
| interpolationPoints | int | $\geq 0$ | The amount of interpolation points on the curve. |

TABLE 3: The parameters used for generating a fat curve

| Name | Type | Bounds | Description |
|---|---|---|---|
| retries | int | $> 0$ | The amount of attempts to place a point. |
| layer radius | float | $> 0$ | The PDD radius used for objects in the same layer. Defined for each layer |
| inner layer radius | float | $0 < r \leq$ layer radius | The PDD radius used for objects in lower layers. Defined for each layer. |

TABLE 4: The parameters used for generating multiple PDD layers

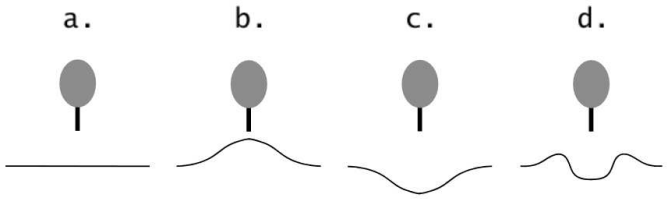



FIGURE 7: Examples of kernels that can be used in the deformation kernel algorithm, from [20].

and when each layer is generated the points are filtered by checking if there is overlap with a higher layer. When placing trees in an environment, their canopies cannot overlap but smaller objects, such as bushes, can be placed within the radius of the canopy. These plants are still blocked by the size of the trunk. Because of this, we can assign a different radius for objects in the same layer as objects in a lower layer. The parameters used in the Multi-layer PDD algorithm are shown in Table 4.

Our second approach follows the method proposed by Lane et al. [20], which uses a probability density function to randomly place objects. Each object placed modifies the probability function with a kernel, which updates an area of the probability function around the placed location. Examples of kernels are shown in Figure 7. The first kernel has no influence on the function, but kernel B increases the probability that another object is placed near the point. Kernel C decreases the chance of points placed near each other, but kernel D has a more complex behaviour. Points placed with this kernel have a low likelihood of being placed close to other points, but have a higher chance of appearing a bit further away. The deformation grid which is a result of generating 50 samples using kernel C is shown in Figure 8. The parameters used in the deformation kernel algorithm are found in Table 5. The bounds and size of the field are kept as constant, and the number of iterations can be changed to change the density of plants placed.

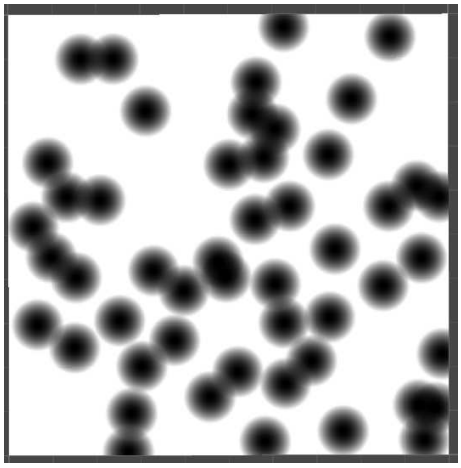


FIGURE 8: Example of a deformation grid after placing 50 objects.

| Name | Type | Bounds | Description |
|---|---|---|---|
| iterations | int | $\geq 0$ | The amount of objects to be placed. |
| deformation kernel curve | AnimationCurve | $f(x) \in [0, \rightarrow)$ | The function used by the kernel from the center to the edge of kernel. |
| deformation kernel radius | float | $> 0$ | The size of the kernel. |

TABLE 5: The parameters used for placing objects with a deformation kernel

## 6.2 Connecting components

In the system, the different components interact with each other, and the generation of some components depends on the result of others. Because of this, an architecture is created that can combine creation of all components. The architecture separates the different features of the landscape in order to keep a clear view of the generation and connection of all parts. When more features are added in the future, the interaction between all components of the system should still be clear.

### 6.2.1 Architecture of the system

We have separated the tasks of generating the terrain into multiple components. Some of the components use the techniques described in the previous chapter to generate a single component of the system, while other components combine these components. The architecture of the system is shown in Figure 9. In this diagram, each different component of the system is shown in a rectangle, while each data type is shown in an ellipse.

In this system, we first generate the surface terrain and after this we plan each river. With the path of the river, we generate a fat curve. To improve the performance of the system, we calculate which triangles of the fat curve intersect each chunk. This allows us to skip triangles which do not pass a given chunk. This is done by component (4). With this information, the terrain is modified with the fat curve to add a river bed to the terrain, and the river is used to generate the initial deformation fields that is used to generate the object placements with the deformation kernel.

The components labelled with the numbers (1), (2), (3), (7.1) and (7.2) are described in section 6.1. The rest of this section describes the other components of the system.

### 6.2.2 Combining terrain and curves

To combine the river with the terrain we need to update each point of the terrain, which is done by component (5). Using the method described in [14] we can check whether a point is located on the river, and at what part of the cross section the point is located. With the cross section profile provided, the change in elevation is calculated. The height of the river is subtracted from the base height of the terrain. To flatten the riverbed, the base height is interpolated between the heights of the side of the river.

### 6.2.3 Masking object placement based on curves

To prevent the placement of objects on places such as water, we need to mask the placement of objects with rivers. The system uses two separate methods of placing objects, and therefore requires different kinds of masking.

For the PDD placement, we add an additional check to the multi-level PDD algorithm as described in section 6.1.3. When a set of objects is generated, they are filtered by their distance to a river to avoid placing objects on rivers. The size of the object is based on the inner radius, such that the trunks of the tree never intersect with a river, but their canopy can be above a river.

When generating objects using the deformation kernel method, we create an initial deformation field. To change the location where objects can be placed, the initial deformation field is modified with the fat curve of the rivers, and the previously placed PDD objects. In the system, this is done by component (6). The deformation field generator creates a deformation field based on the

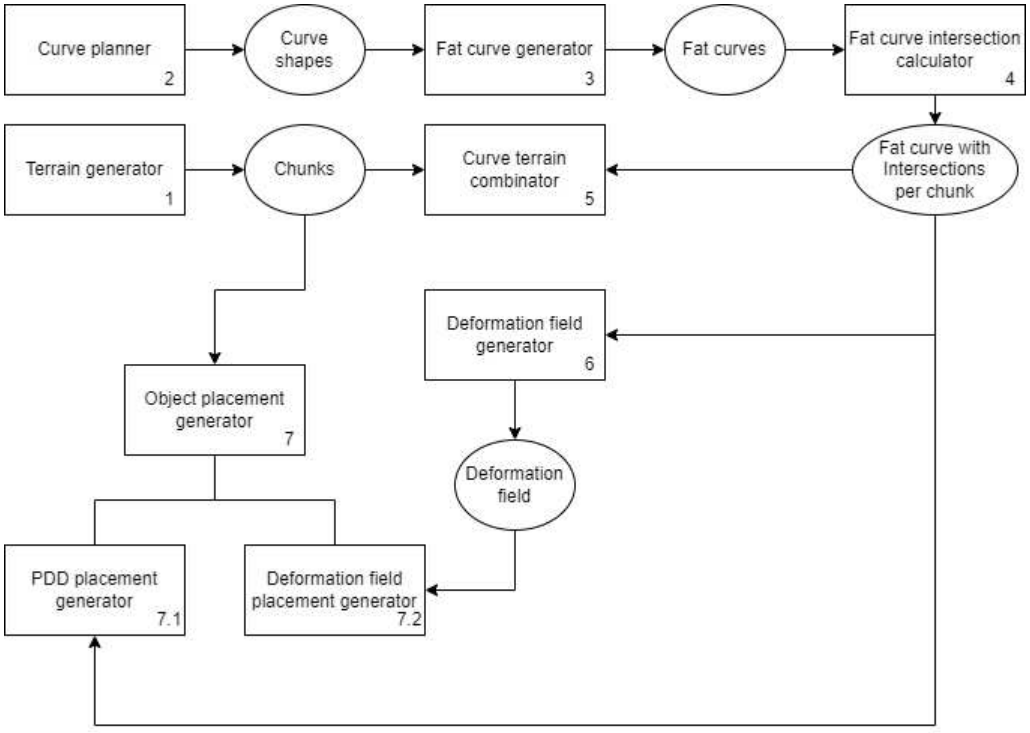



FIGURE 9: A block diagram of the system's architecture

terrain bounds and the rivers. The parameters used by the deformation field generator can be found in Table 6. An example of an initial deformation field is shown in Figure 10. By separating the initial probability of a point placed outside and inside of a river, this method allows for objects that should be placed near rivers, such as reeds.

| Name | Type | Bounds | Description |
|---|---|---|---|
| initial probability | float | $\geq 0$ | The initial probability of a point not on a river. |
| river probability | function | $f(x) \in [0, )$ | The initial probability of a point on a river, evaluated on the cross-section of the river. |

TABLE 6: The parameters used for generating an initial deformation field

### 6.2.4 Combining object placement methods

The system uses two techniques for placing vegetation. For larger vegetation the PDD placement is used as it gives the guarantee of objects being spaced a certain distance from each other. For smaller vegetation the deformation kernel approach is used as it results in objects being placed more randomly, which results in a more realistic look.

To combine both methods, the Object placement generator, which is component (7) in the architecture, is used. This component uses components (7.1) and (7.2), which are described in chapter 6.1.3. Firstly, the larger vegetation is generated using the PDD. To prevent smaller vegetation being placed within these objects, all placements are added to the deformation field using the inner radius. The probability on those locations is set to 0.

### 6.2.5 Finalising terrain

When all components are generated and combined, some final steps are performed to transform our data structures into a terrain. Firstly, the heightmap of the terrain is rasterized into a mesh. Based on the fat curves of the river, meshes are generated for the river and placed in the landscape.

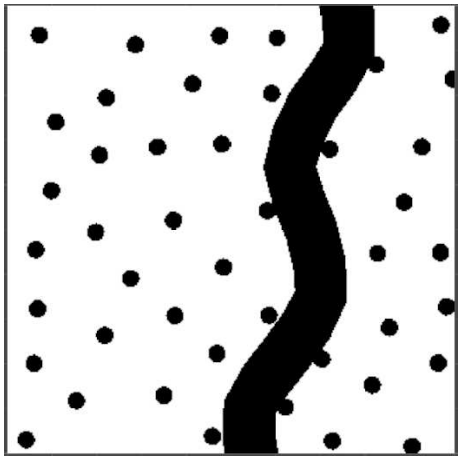


FIGURE 10: An example of an initial deformation field based on a river and PDD objects.

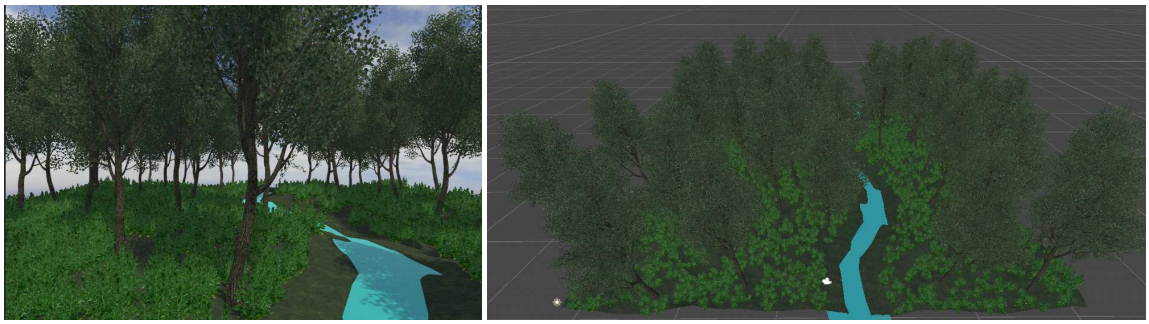


FIGURE 11: A complete landscape generated by the system.

After this, the placement objects are added to the terrain based on predefined models. The height map of the terrain is used to place the objects, as well as the camera, on the correct height of the terrain. The result of generating a complete landscape is shown in Figure 11, as both a view from the camera and a top-down perspective.

# 7 Validation

When generating the terrain, a user might have certain requirements that the terrain has to adhere to. For instance, when the user needs a terrain with a river, the generated river should be visible from the camera's perspective. Additionally, placed objects should not block large parts of the view or other large sections of the environment. To show how the environment can be analysed and validated, we have chosen three metrics that we want to analyse. We will look at the density of forest and how this is influenced by certain parameters, as well as the visibility of rivers and trees from the perspective of the camera.

## 7.1 PDD placement and object density

In research of digital nature, one of the variables in the type of environment is spaciousness, or the difference between dense and spacious environments[35], which is related to the density of the objects placed. Instead of generating environments and calculating the density of the objects from the result, it is more efficient to derive the density of the environment from the parameters.

In the PDD placement algorithm, the placement of the objects is dependent on the radius of each layer. Changing the radius of an object changes the minimum distance between two objects. The density of the objects placed can therefore be changed by increasing the radius. To show the effect of the radius on the density, we take an initial radius $r$ that represents the bounds of the

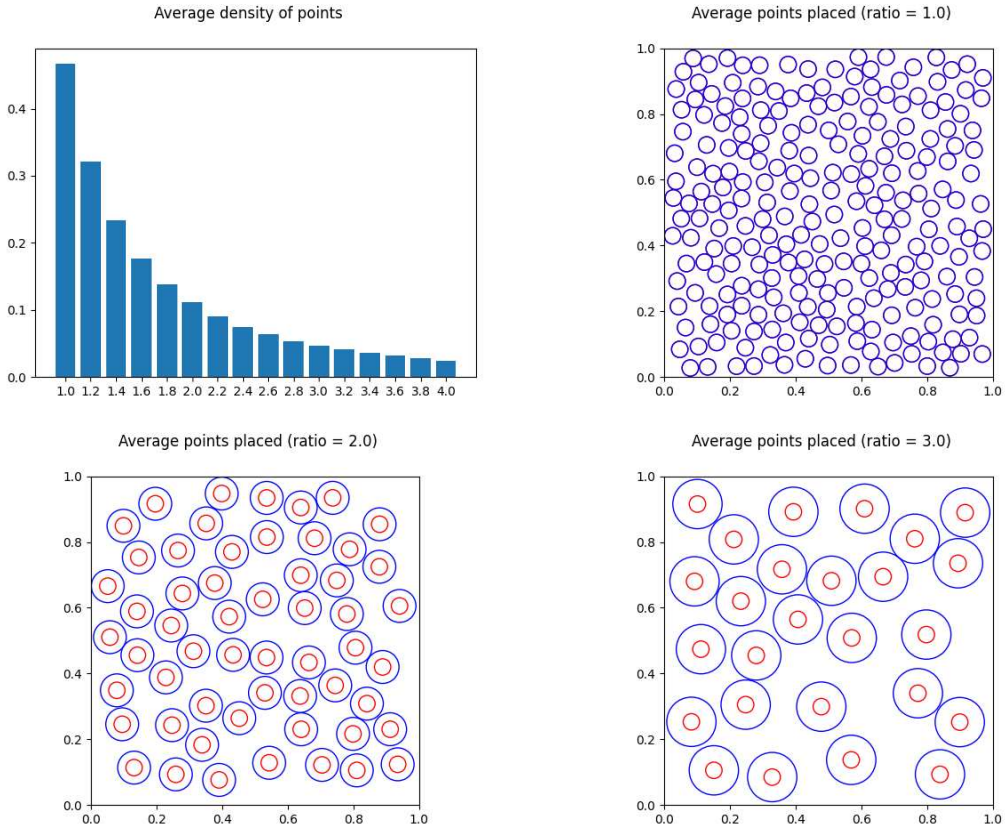



FIGURE 12: Density of objects placed with the PDD using increasing radii, along with a few examples of objects placed.

object itself. We increase this radius by a factor $f$ and run the algorithm. From the number of points placed we calculate the area that the points occupy with the original radius and calculate the density. To see the effect of increasing the radius of an object, we take a ratio from 1 to 4 with increments of 0.2. For each ratio, we run the algorithm 500 times and calculate the average density by using the area of the initial radius. The results of this example are shown in Figure 12, along with a few examples showing the change of radius. From the figure, we see that the average density of points decreases as the minimum radius increases.

## 7.2 Object segmentation

When validating the terrain, we want to make a distinction between different types of objects, such as the difference between the ground and rivers, or between the sky and placement objects. To solve this we use object segmentation, which is regularly used in computer vision [19]. In object segmentation we make a distinction between semantic segmentation, where all objects of the same type have the same colour, and panoptic segmentation, where each object has a different colour, regardless of its type.

In our implementation, we use a custom shader to colour the individual objects. When analysing the resulting image, we use pixel counting to find the relative size of the objects in the view.

An example of applying image segmentation to an image of a landscape is shown in Figure 13. In this image, we can see the original landscape on the top-left, and the segmented images on the right. The top image shows the object segmentation where the different kinds of placement object have a shared colour. The bottom image shows panoptic segmentation, where each object has an individual colour.

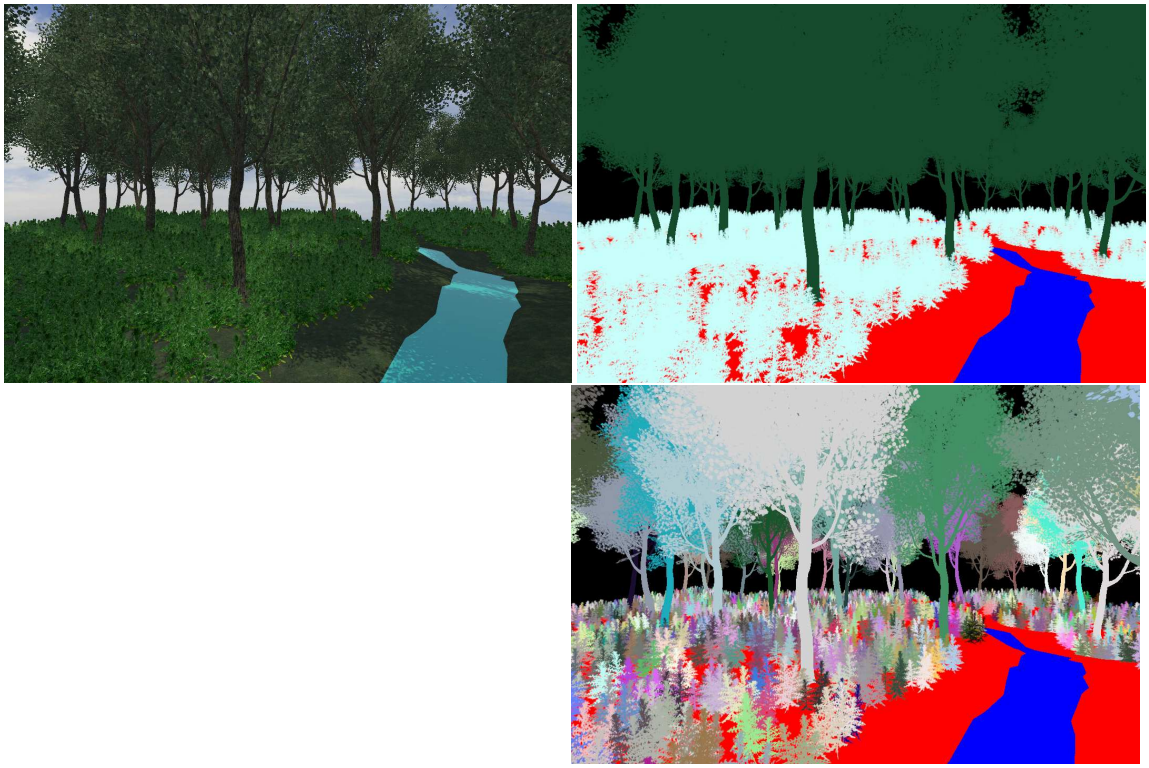


FIGURE 13: Example of object segmentation (top-right) and panoptic segmentation (bottom-right) on a generated terrain (left)

## 7.3 Presence and visibility of rivers

When generating an environment with a river, it is important that the river is visible from the camera's perspective. If a river is not visible, or only a very small part is visible, the generated landscape does not meet the user's requirements and cannot be used.

To show the visibility of the river in the landscape, we analyse the proportion of the river in the terrain. As placement objects can obscure both rivers and terrain, this analysis is performed without the generation of placement objects.

To calculate the visibility of the river, the view of the camera is rendered with object segmentation, and the proportion of river pixels against the entire terrain is calculated.

### 7.3.1 Testing the visibility of rivers

To test the visibility of the river, we take 500 samples of generated terrain, of which we calculate the percentage of terrain covered by the river. The result of this experiment is shown in Figure 14. In these results, we see that the visibility of the river is very low, with more than 60% of the samples containing a river where the visibility is less than 5% of the total terrain. One of the experiments where there is no river visible is shown in Figure 15a. A top down perspective of this terrain is shown in Figure 15b. As seen, the river is generated near the edge of the terrain, outside of the perspective of the camera. The experiment with the largest visible river is shown in Figure 15c. As seen in the top down perspective in Figure 15d, this river runs through the middle of the terrain, and bends close to the camera.

### 7.3.2 Improving the visibility of rivers

In the previous tests, we kept the camera in a fixed position, namely on the centre of one side of the terrain. Instead of generating the terrain multiple times, which takes additional, the camera can be placed on any of the four sides of the terrain. This results in a different perspective on the terrain and therefore a different view of the river.

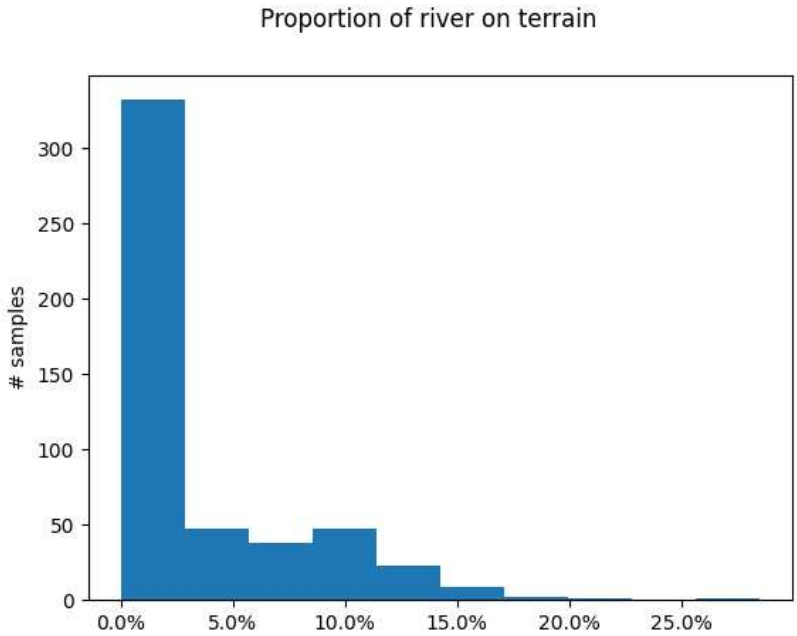



FIGURE 14: Histogram of the results of the river visibility test

To test this improvement, we repeat the previous experiment where we calculate the proportion of the river, but for each generated terrain we calculate the visibility for each camera position, and take the camera position with the largest visibility. The histogram with the results of the experiments can be seen in Figure 16. From the results it shows that checking different camera positions does result in terrains where the river is more visible. Instead of more than 60% of the experiments having no visible or nearly invisible river, only 40% of the samples show this property when multiple camera angles are used. Most of the samples have a visible river proportion between 5 and 20%.

## 7.4 Single-object occlusion

As trees are placed randomly on the terrain, it is important to ensure that a large part of the view is not obscured by a single tree or a few objects. Because of this, it is important to evaluate the proportion of an individual object to the scene.

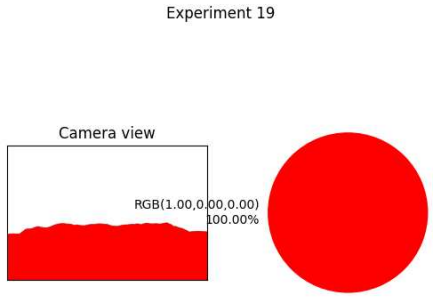
To calculate the individual proportion, we use panoptic segmentation to separate the different trees in the image. For each tree in the view, we calculate the size of the object in the view in pixels, as well as the position of the object relative to the camera.

As a test, we generate the terrain 500 times and calculate the panoptic segmented image from the terrain. We calculate the distance from the object to the camera and the size of the object in pixels as a fraction of the total size of the screen. The result of these experiments can be found in Figure 17. In Table 7 a breakdown of the objects and their sizes is shown.

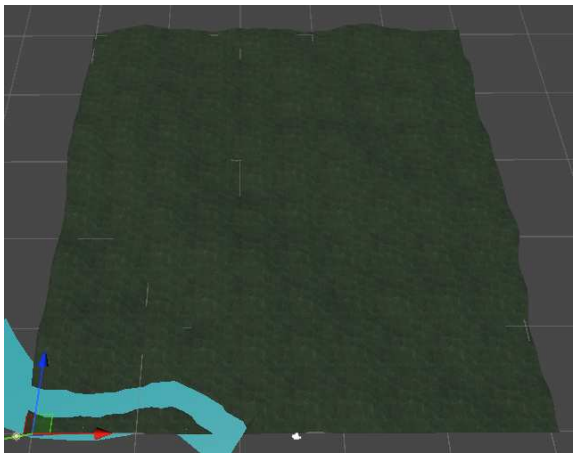
In these results we can see that most of the objects only cover a small proportion of the screen. Almost all objects that are visible in the frame take up less than 10% of the screen. Only 3.7% of the generated objects are larger than 10%, but some of these objects occupy a very large section of the screen. In Figure 18 an example of a terrain is shown where a single object occupies a large section of the window. The tree is placed close to the camera, only 6.6 metres away from the camera, and obstructs a large part of the view of the camera.

### 7.4.1 Improving single-object occlusion

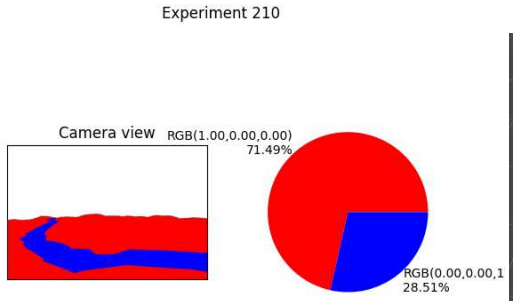
As seen in Figure 17, most objects with a very large visibility are placed within 15 units of the camera. All objects that cover more than 30% of the screen are placed within 11 metres of the



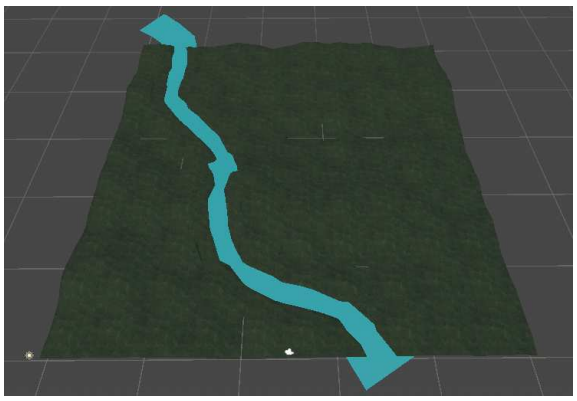

(A) Segmented view of a terrain without visible river



(B) Terrain of the view without a visible river




(C) Segmented view of terrain with the most visible river



(D) Terrain of the view with the most visible river

| Proportion of object | # of objects | % of Objects |
|---|---|---|
| >10% | 633 | 3.71% |
| >20% | 195 | 1.14% |
| >30% | 94 | 0.55% |
| >40% | 48 | 0.28% |
| >50% | 27 | 0.16% |
| >60% | 17 | 0.10% |
| >70% | 7 | 0.04% |
| >80% | 3 | 0.02% |
| >90% | 1 | <0.01% |

TABLE 7: Amount of objects with a minimum size, from 500 experiments.

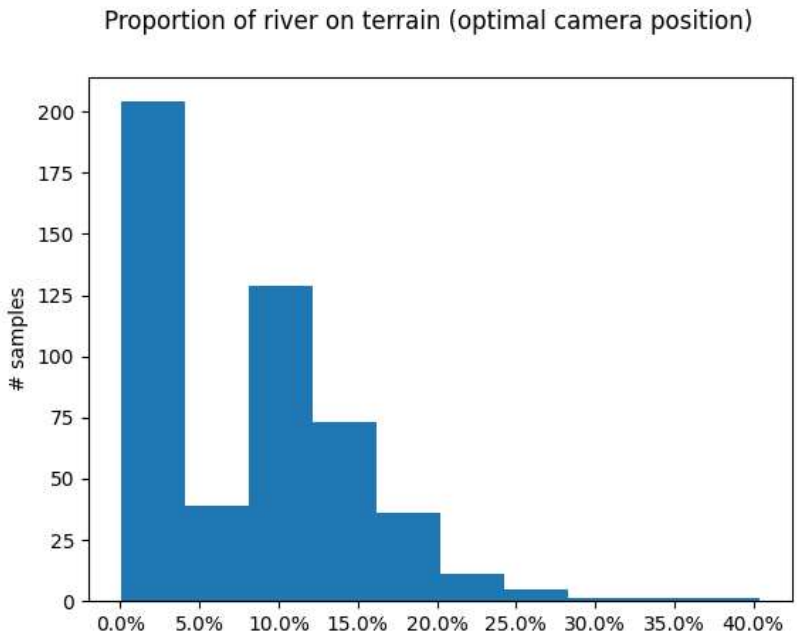



FIGURE 16: Histogram of the results of the river visibility test with the best camera position.

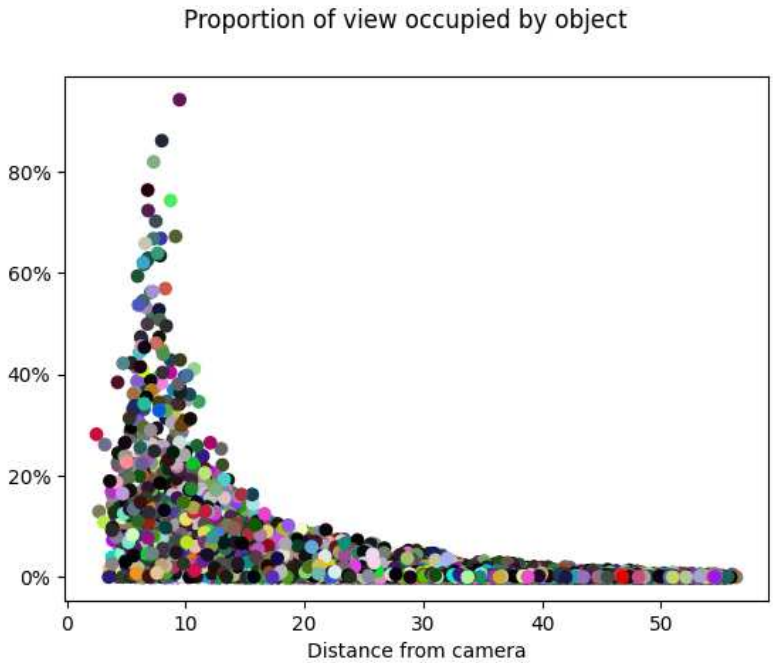



FIGURE 17: A plot of the distance and size of each object in the experiments.

FIGURE 18: Example of a view obstructed by a single object, with a normal render and using panoptic segmentation

camera. To prevent occlusion by a single object, objects within 11 metres of the camera are removed. The second round of experiments can be seen in Figure 19. This approach significantly reduces the amount of objects with a very large visibility on the screen.

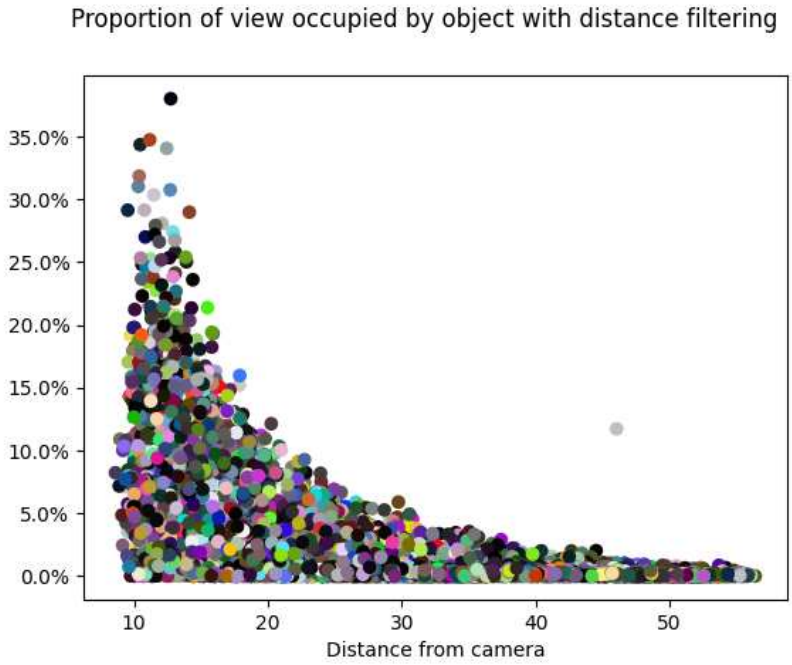



FIGURE 19: A plot of the distances and sizes of each object in the experiments with camera distance filtering of 11 meters.

# 8 Discussion

With the architecture we have created, Magrathea is capable of generating digital landscapes with rivers, smaller vegetation, and larger trees. The different components are connected to each other without overlap, but they still create a whole environment. In the environments created during testing, which represents a forested area, the generation terrain does not require large inclines, but still results in a landscape that is not fully flat. The generation of the river results in rivers that curve throughout the terrain, but the planning of the rivers does not work very well. Randomly placing points often results in rivers that generate mostly near the corners of the terrain and are not visible to the camera. The placement of vegetation allows different types of plants to be placed on the terrain without overlap with each other or with rivers. The distribution can be configured to modify how plants spread or clutter in the environment.

The system is able to generate environments quickly enough. To test the system, we created a preset with a single river, an environment with 10 by 10 chunks of 65 by 65 vertices each, a single PDD object, and a single deformation kernel object. Running the testing setup, the system takes an average of 2.23 seconds to generate an environment. Further optimisations can improve this, but when only a small number of environments need to be generated, the current runtime is good enough.

## 8.1 Architecture

In the architecture that we have defined, the different tasks of generating each terrain feature are separated into one or multiple different components. Each of the components generates, transforms, or combines specific parts of the system. Some of the components are isolated from other parts of the system, such as the curve planner and fat curve generator, while others are used by a lot of different components, like the cached intersections between fat curves and chunks.

The different components of the system can be divided into 3 categories, based on the different terrain features. The division of these categories is shown in Figure 20. The first category, category A, is related to the generation of the terrain and is the smallest category. Only the generation of the chunks is required to create the terrain. The second category, category B, is responsible for the

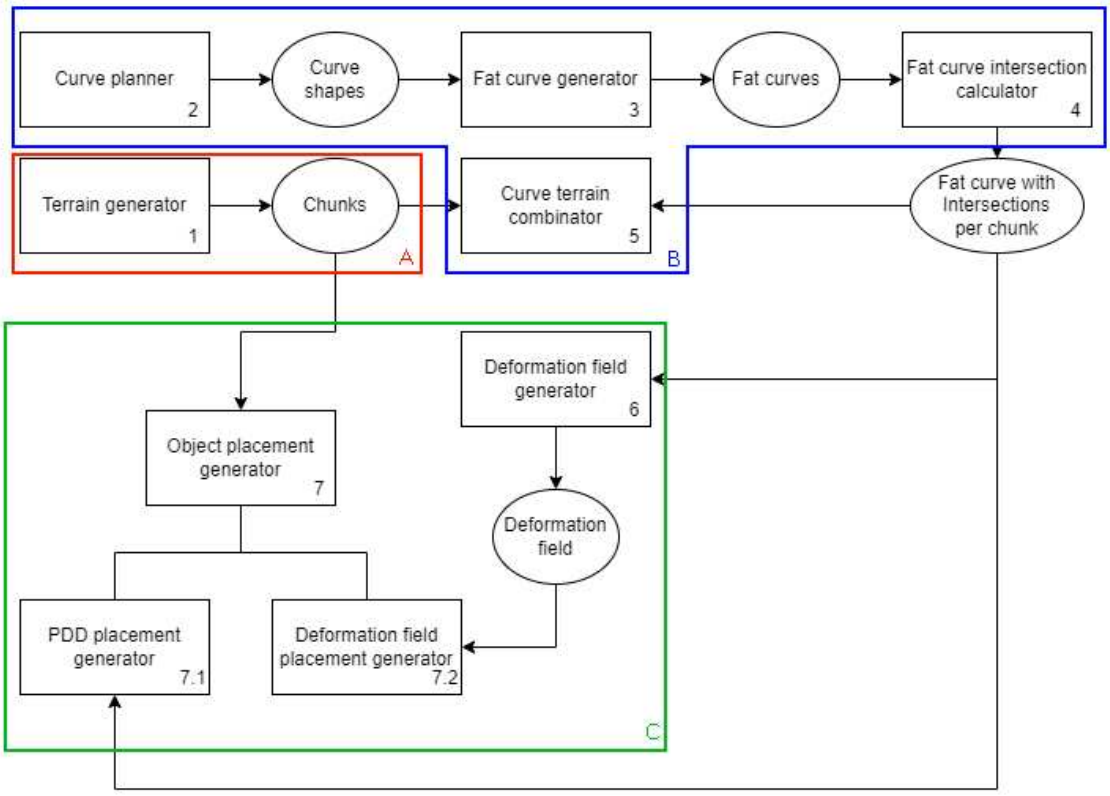



FIGURE 20: Architecture of the system with components categorised for each environment feature

planning and generating of the rivers, as well as combining the rivers onto the terrain. Category C is used to calculate the positions of objects to be placed onto the terrain using both the PDD and deformation field placement methods. This category depends on the rest of the system, as it requires the height map of the chunks to place objects and the rivers to filter object placements. In Figure 20, the intersections between fat curves and chunks are not divided into a category, as it is a separate data structure used by all parts of the system.

## 8.2 Parameters of the system

For each of the components in the system, we have established a set of parameters that influence the generated terrain. Some of these parameters, such as the size and resolution of the height maps of the terrain, can be kept constant for any generated terrain. With the other parameters, the user is able to influence the generated terrain.

For the parameters related to the generation of the terrain, most of the parameters are very technical. The parameters of the fractal noise algorithm apply to the mathematical nature of noise algorithms and are not accessible for users without knowledge of noise functions. Due to this, when a designer wants a specific feature on the terrain, more experimentation is required to generate a terrain that is acceptable to the user.

To plan the generation of the river, the parameters have a clearer meaning on how they influence the system. The planned rivers generate in a specific shape, namely a meandering curve that can randomly curves around the environment, and the parameters can influence the shape the river takes. Changes in meandering amplitude or random rotation can result in different characteristics of rivers, as shown in the research by Teoh [33]. In the current version of the system, there are no direct parameters that contribute to the initial position and direction of the river, so this is still a completely random process. This results in rivers that only occupy a very small part of the terrain

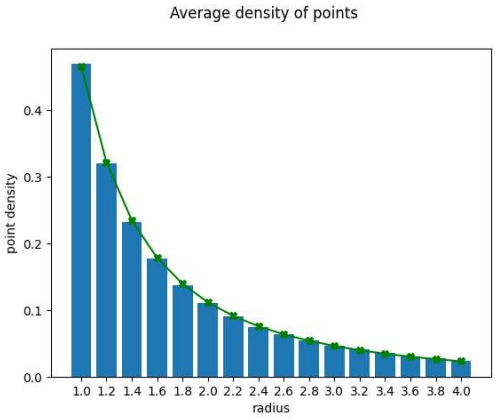



FIGURE 21: The density of points with varying ratios, with an exponential and inverse quadratic fit

| Function | MSE |
|---|---|
| Exponential | $3.97 * 10^{-5}$ |
| Quadratic | $9.60 * 10^{-7}$ |

TABLE 8: Mean squared error of different relations.

and are barely visible. To transform the river path into a fat curve, the process mainly contains constant parameters. Since the interpolation of the path uses Bezier curves, a parameter has been added to derive the tangents of each control point. As this is mainly to control the smoothing of the curve, this parameter can be kept as a constant.

The parameters related to placing objects are closely related to the type of vegetation used. For the PDD algorithm, the parameters are related to the physical size of the objects that are placed. For trees, these are the outer radius which equates to the size of the canopy, which prevents overlap between trees, and the inner radius, which is to prevent overlap with smaller objects. The multi-level PDD placement has been used in testing and allows smaller objects to be placed between larger objects, but realistic placement patterns such as clumps do not appear when using PDD.

For the deformation kernel the initial probability field is based on the path of the river, with a parameter specifying the initial probability of an object placed on or near a river. For the placing of objects, there are parameters related to the size of the object to be placed, as well as how the probability function changes when an object is placed. With these parameters, different kinds of distributions can be achieved. The amount of objects placed.

## 8.3 Validating system components

To validate the generated terrain, we have used a number of different metrics to check and control the environment. Each of these methods analyses a specific component and how it affects the generated environment. Currently, these methods are not directly involved in the final terrain generation, but can instead be used after a terrain has been generated to improve the result.

For the placement of PDD objects, we have experimented with increasing the radius to reduce the density of trees. From this, we see an inverse relation between the radius of the object and the density of the objects within the area. To find this relation, we fitted two functions on the dataset, the exponential function and an inverse quadratic relation. These fits are shown in Figure 21. For both fits, we calculated the mean squared error, which is shown in Table 8. From this we can see that the quadratic relation has a better fit. This is expected, since increasing the radius linearly results in the area quadratically. With this relation, the user can calculate the expected density of the generated objects.

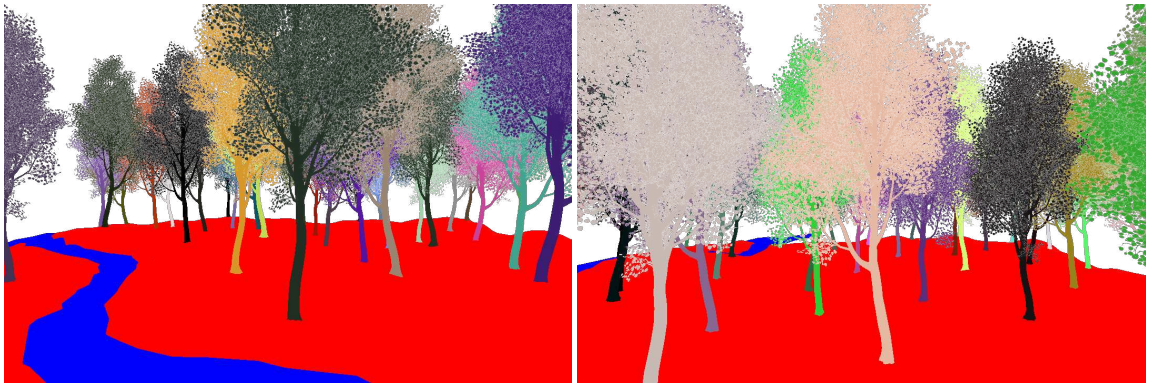


FIGURE 22: Examples of environments generated with objects closer than 11 units removed

### 8.3.1 River visibility

In the current implementation of the system, generating a river visible from the camera is very unreliable. Since the start point and direction of the rivers are selected by random, rivers often only traverse for a small distance before moving out of bounds, or they can travel near a border that is not visible from the perspective of the camera. In the initial version, more than 80% of all rivers have a visibility lower than 5% of the total terrain surface, which results in a large number of terrains to be generated before an acceptable terrain appears. Improving this method by taking multiple camera positions into account has improved the system somewhat, although there is still a large amount of generated rivers that still do not contain a large section of the river. If a river is generated near the corner of the terrain, none of the camera positions will be able to view the river.

### 8.3.2 Single object occlusion

To validate whether the view is obscured by a single large object, we have used panoptic segmentation to calculate the size of each object in the view. With this, we found that the system sometimes generates objects close to the camera that take up almost all the space in the view, as shown in Figure 18.

The solution to remove all objects close to the camera has resulted in removing most large objects in the view. Some examples of landscapes generated using this method are shown in Figure 22. Although the second round of experiments still contains objects that cover more than 30% of the view, the majority of views do not contain a large object in front of the camera. Although this approach seems to work well when looking at the statistics alone, the environments generated do look more sparse because of the empty space between the camera and the closest trees. When a dense environment is required, filtering nearby trees removes a large part of the density.

# 9 Future work

In its current state, the system is able to generate environments with vegetation and rivers, which can be manipulated using a set of parameters. However, there are still some changes and improvements that can be made to the system to enhance the features and capabilities of the system. Some of these improvements are described in the following section.

## 9.1 Improving performance

In the initial version of the system, there has been no significant focus on improving the performance of the system. One improvement that has been made is caching the intersection between fat curve triangles and chunks, but the performance of the system can be further improved by taking advantage of Unity's optimisation techniques. One example would be to use parallelisation and SIMD by using Unity's job system in generating the terrain or operations with fat curves. Research

done by Wei [38] proposed a way to parallelise the Poison Disk Distribution on the GPU, which can be applied to the PDD object placement component.

Although the current system can generate landscapes in a reasonable time, faster landscape generation can allow the user to generate more terrains in the same timespan, allowing for more options of the resulting environment.

## 9.2 Multiple environments

While the system is able to generate environments with multiple types of vegetation, is not able to generate terrains with multiple environments within the same landscape. For example, the system cannot generate the edge of forests where a section of the terrain contains trees and the other section only contains grass or bushes. Interesting environments often contain different types of landscape, so therefore, the addition of multiple types of environment should be added. Some research has already gone into the generation of multiple ecotope. Hammes [13] uses terrain features such as the height and slope of the terrain to determine which ecotope the section of terrain belongs to. Based on the ecotope, different plants are selected for placement. [9] and [37] take a similar approach, but both use water as well as terrain, so rivers also influence the selected ecotype.

## 9.3 Generating roads

In digital nature research, the presence of artificial objects is a desired component of the system [35]. Because of this, one of the next features that should be added is the presence of roads or road networks. Roads can be planned on the terrain similar to rivers, and the addition will change the architecture. The planning and generation of roads has been researched before, most times in the procedural generation of cities [18] or in the procedural generation system by Smelik [32]. In addition to the placement of the roads itself, certain objects can be placed adjacent to roads, such as roads, sign posts or bins. With these objects, tended natural environments can be tested, like in the research of van Houwelingen-Snippe [35].

## 9.4 Generating from point of view

Although the aim of the system is to generate an environment for a single point of view, the generation techniques used are the same that are used to create large-scale environments. New techniques that involve the position of the camera directly to generate each component can generate the environment more effectively. As mentioned in the discussion, generated rivers are often not generated visible to the camera. When the position of the camera is taken into account, rivers can be generated such that they will appear in a certain position of the view. Additionally, the generation of objects can be changed by including the position of the camera to distinguish between foreground and background objects, which can result in interesting environments, or to prevent occlusion by a single large object.

# 10 Conclusion

## 10.1 RQ 1: Terrain generation

**How can we generate each component in a landscape, and how do these components interact to create a landscape suited for a single point of view?**

The system we have created currently supports 3 features: The terrain, rivers and the placement of vegetation.

### RQ 1.1: How can we generate each component required for the landscape?

The surface terrain is generated using fractal Perlin noise as a two-dimensional height map, and split into separate chunks. For each height map, a mesh is generated. This method is able to generate simple terrain with some ruggedness, which is enough for simple environments, such as forests.

To generate rivers, we plan the path of a meandering river that randomly moves around the terrain. This path is transformed into a fat curve, which is used in other steps of the program. To reduce the amount of calculations in other parts of the program, the intersections between chunks and fat curves are cached.

For the generation of vegetation we have selected two methods. The first method randomly places objects on a Poisson Disk Distribution, where all objects have a minimum distance to another object. With the second approach, a probability distribution is used to randomly place objects with each object modifying the distribution. Using different kernels changes the behaviour of objects placed, where certain kernels produce a prohibiting effect for other objects, other kernels can attract objects to clutter near each other. Using both methods of generation, different types of behaviour can be produced when generating the environment.

**RQ 1.2: How can we connect all components to create a landscape for a single point of view?**

Using the cross section of the river, the fat curve modifies the height map of the terrain to imprint the river bed and creates a mesh that represents the river surface. Although the generation of the river itself works, the river planning is very simplistic and often generates rivers that are not visible from the camera's perspective. From our experiments more than half of the terrains we have generated does not include a significant portion of the river in the view.

To prevent overlap with rivers, the fat curve of the river acts as a filter for PDD objects, where objects are not generated if they are placed on a river. For the deformation kernels, the fat curve modifies the initial probability distribution to change the probability of an object placed on a certain part of the river. To place the objects onto the terrain, the height map is used to find the vertical position of the object on the terrain and the update is placed on the surface.

## 10.2 RQ 2: Parameters of the system

**What are the influences of the different parameters on the resulting landscape?**

In the architecture we have created, each of the different components has a number of parameters that influence the generation or transformation of the specific feature. These parameters are categorised into two categories, where parameters can be considered as constant in most of the generated terrains, and parameters that change the terrain in order to get a desired effect.

For generating the terrain, most parameters are related to the mathematical noise functions, and changing the parameters requires knowledge of these functions. The generation of rivers uses parameters that describe the general shape and direction of the river. For the generation of objects, the parameters are inherent to the objects to be placed on the terrain. In the Poisson Disk Distribution, objects specify separate radii for objects of the same type or objects in a lower layer. In the deformation field, the objects placed specify how the probability field is modified in an area around the object. The amount of objects placed using the PDD is limited by the bounds of the terrain and the minimum distance, while the deformation uses a parameter to control the amount of objects generated.

## 10.3 RQ 3: Validating terrain

**In what ways can we validate the procedurally generated terrain?**

To validate the generated environments, we have selected three metrics to test the landscape. These metrics are as follows:

- The density of trees in the environment.
- The visibility of the generated rivers.
- Whether objects obstruct a large part of the view.

**RQ 3.1: How can different requirements be translated into rules to validate the terrain?**

For the first metric, we analysed the density of the generated trees. With the PDD algorithm, a minimum distance between each object is ensured but by increasing this minimum distance, the density of objects decreases. By testing the density for different radii, we found an inverse quadratic relation between the radius used in the PDD algorithm and the density of objects.

The second metric we have analysed is the visibility of the generated rivers in the view of the camera. To analyse this, the generated terrain is evaluated by the proportion of river visible on the terrain from the camera's perspective. This method only analyses the surface without any vegetation placed. From this terrain, object segmentation is used to calculate the size of the river in pixels. From this the ratio between the surface and the rivers is calculated.

For the placement objects, we analysed the visibility of each object, and looked at whether some objects take up a significant portion of the screen. Instead of object segmentation, panoptic segmentation is used to separate each object in the image. For each terrain, the pixels per object are counted and calculated as the proportion of the total screen size. From this we have seen that some objects occupy a significant portion of the screen.

**RQ 3.2: How can the terrain generation be influenced by the validation to create terrain conforming to certain rules?**

The relation between the minimum radius and the density of objects generated by the PDD algorithm can be used by a designer to estimate the resulting density in the generated environment. When a certain density of objects is required, it can be calculated back to a minimum radius and then validated when the terrain is generated by counting each object.

The visibility of rivers can be used by a designer to automatically find a suitable terrain. If a designer requires a river to be visible, they can generate the terrain until the visibility is larger than a set amount. With multiple camera positions, this increases the chance that a terrain has a river that is visible enough.

With the size of each placement object on the screen, objects that block a large part of the view can be removed. We found that most objects that occupy a large part of the view are placed within a distance of 11 metres of the camera. To prevent these objects from blocking a large part of the view, they can be removed on the basis of the distance of the object to the view.

# References


[1] Farès Belhadj and Pierre Audibert. Modeling landscapes with ridges and rivers: bottom up approach. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '05, pages 447–450, New York, NY, USA, November 2005. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/1101389.1101479, doi:10.1145/1101389.1101479.

[2] Bedřich Beneš and Rafael Forsbach. Visual simulation of hydraulic erosion. 2002. Accepted: 2013-07-15T13:50:11Z Publisher: UNION Agency. URL: http://dspace5.zcu.cz/handle/11025/5963.

[3] Robert Bridson. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 sketches*, page 22, San Diego California, August 2007. ACM. URL: https://dl.acm.org/doi/10.1145/1278780.1278807, doi:10.1145/1278780.1278807.

[4] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang Tiles for image and texture generation. *ACM Transactions on Graphics*, 22(3):287–294, July 2003. URL: https://dl.acm.org/doi/10.1145/882262.882265, doi:10.1145/882262.882265.

[5] Juncheng Cui, Yang-Wai Chow, and Minjie Zhang. A voxel-based octree construction approach for procedural cave generation. *Faculty of Informatics - Papers (Archive)*, January 2011. URL: https://ro.uow.edu.au/infopapers/3612.

[6] D. D'Ambrosio, S. Di Gregorio, S. Gabriele, and R. Gaudio. A Cellular Automata model for soil erosion by water. *Physics and Chemistry of the Earth, Part B: Hydrology, Oceans and Atmosphere*, 26(1):33–39, January 2001. URL: https://linkinghub.elsevier.com/retrieve/pii/S1464190901850115, doi:10.1016/S1464-1909(01)85011-5.

[7] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomír Měch, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 275–286, New York, NY, USA, July 1998. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/280814.280898, doi:10.1145/280814.280898.

[8] Rahul Dey, Jason G. Doig, and Christos Gatzidis. Procedural feature generation for volumetric terrains using voxel grammars. *Entertainment Computing*, 27:128–136, August 2018. URL: https://linkinghub.elsevier.com/retrieve/pii/S1875952117301349, doi:10.1016/j.entcom.2018.04.003.

[9] Bruno Torres do Nascimento, Flavio Paulus Franzin, and Cesar Tadeu Pozzer. GPU-Based Real-Time Procedural Distribution of Vegetation on Large-Scale Virtual Terrains. In *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 157–15709, October 2018. ISSN: 2159-6662. URL: https://ieeexplore.ieee.org/abstract/document/8636903?casa_token=IoZQcW6PpB8AAAAA:7gvMS7_v60kDTLNDIcNa3XBuwLQAengSbAZs2V2URLTRxSy4ayn6ibXbF2riyOd8N66va6BGnw, doi:10.1109/SBGAMES.2018.00027.

[10] Jonathon Doran and Ian Parberry. Controlled Procedural Terrain Generation Using Software Agents. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):111–119, June 2010. Conference Name: IEEE Transactions on Computational Intelligence and AI in Games. URL: https://ieeexplore.ieee.org/abstract/document/5454273?casa_token=GPHytrYBv10AAAAA:etKTf-MqS36Xpc6fRwIksbRZ9ZpWJ68gTeFZAwrtKjoGYlA2ShUIT_mC-hFqX03nDag2fz4SGg, doi:10.1109/TCIAIG.2010.2049020.

[11] Manuel Gamito and F. Musgrave. Procedural Landscapes with Overhangs. December 2003. URL: https://www.researchgate.net/publication/2948853_Procedural_Landscapes_with_Overhangs.

[12] Sarah F. F. Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In William M. Wells, Alan Colchester, and Scott Delp, editors, *Medical Image Computing and Computer-Assisted Intervention — MICCAI'98*, Lecture Notes in Computer Science, pages 888–898, Berlin, Heidelberg, 1998. Springer. doi:10.1007/BFb0056277.

[13] Johan Hammes. Modeling of Ecosystems as a Data Source for Real-Time Terrain Rendering. In Caroline Y. Westort, editor, *Digital Earth Moving*, Lecture Notes in Computer Science, pages 98–111, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-44818-7_14.

[14] Remco Huijser, Jeroen Dobbe, Willem F. Bronsvoort, and Rafael Bidarra. Procedural Natural Systems for Game Level Design. In *2010 Brazilian Symposium on Games and Digital Entertainment*, pages 189–198, Florianpolis, Santa Catarina, TBD, Brazil, November 2010. IEEE. URL: http://ieeexplore.ieee.org/document/5772287/, doi:10.1109/SBGAMES.2010.31.

[15] Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. Blue noise sampling using an SPH-based method. *ACM Transactions on Graphics*, 34(6):211:1–211:11, November 2015. URL: https://dl.acm.org/doi/10.1145/2816795.2818102, doi:10.1145/2816795.2818102.

[16] Thouis R. Jones. Efficient Generation of Poisson-Disk Sampling Patterns. *Journal of Graphics Tools*, 11(2):27–36, January 2006. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/2151237X.2006.10129217. doi:10.1080/2151237X.2006.10129217.

[17] Martin Kahoun. Realtime library for procedural generation and rendering of terrains. September 2013. Accepted: 2017-05-15T13:04:09Z Publisher: Univerzita Karlova, Matematicko-fyzikální fakulta. URL: https://dspace.cuni.cz/handle/20.500.11956/51668.

[18] George Kelly and Hugh McCabe. Citygen: An Interactive System for Procedural City Generation. URL: https://www.researchgate.net/publication/357658334_Citygen_An_Interactive_System_for_Procedural_City_Generation.

[19] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollar. Panoptic Segmentation. pages 9404–9413, 2019. URL: https://openaccess.thecvf.com/content_CVPR_2019/html/Kirillov_Panoptic_Segmentation_CVPR_2019_paper.html.

[20] Brendan Lane and Przemyslaw Prusinkiewicz. Generating spatial distributions for multilevel models of plant communities. 2002. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5eb032f25a186124371de7c537b13fb0ad49113a.

[21] Thomas Lechner, Benjamin Watson, U. Wilensky, and M. Felsen. Procedural City Modeling. 2003. URL: https://www.semanticscholar.org/paper/Procedural-City-Modeling-Lechner-Watson/dc8a76e03a9139f24c2a4f03279d8a6980e86692.

[22] Gavin S P Miller. The definition and rendering of terrain maps. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 39–48, New York, NY, USA, August 1986. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/15922.15890, doi:10.1145/15922.15890.

[23] Casey Muratori. The Color of Noise, May 2014. URL: https://caseymuratori.com/blog_0010.

[24] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *ACM SIGGRAPH Computer Graphics*, 23(3):41–50, July 1989. URL: https://dl.acm.org/doi/10.1145/74334.74337, doi:10.1145/74334.74337.

[25] Callum Newlands and Klaus-Peter Zauner. Procedural Generation and Rendering of Realistic, Navigable Forest Environments: An Open-Source Tool, August 2022. arXiv:2208.01471 [cs]. URL: http://arxiv.org/abs/2208.01471, doi:10.48550/arXiv.2208.01471.

[26] Jacob Olsen. Realtime Procedural Terrain Generation. 2004. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5961c577478f21707dad53905362e0ec4e6ec644.

[27] Przemyslaw Prusinkiewicz and Mark Hammel. A Fractal Model of Mountains with Rivers. 1993. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0af95fb26380dae62a6ec46dfd7eba9fcaeccdc3.

[28] Nicoletta Sala. Mathematics, Territory and Landscape. 2002. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ee6f5c2e6b279ef9d81e1249bb07ebf7dcaab750.

[29] Jens Schneider, Tobias Boldte, and Rudiger Westermann. Real-Time Editing, Synthesis, and Rendering of Infinite Landscapes on GPUs. 2006. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2000fd31703293aaa08fe622a3b032f21ba707e6.

[30] Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Springer International Publishing, Cham, 2016. URL: http://link.springer.com/10.1007/978-3-319-42716-4, doi:10.1007/978-3-319-42716-4.

[31] Marcus Sköld. *Generating terrain features using software agents*. 2023. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:mau:diva-62688.

[32] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. A declarative approach to procedural modeling of virtual worlds. *Computers & Graphics*, 35(2):352–363, April 2011. URL: https://www.sciencedirect.com/science/article/pii/S0097849310001809, doi:10.1016/j.cag.2010.11.011.

[33] Soon Tee Teoh. RiverLand: An Efficient Procedural Modeling System for Creating Realistic-Looking Terrains. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Yoshinori Kuno, Junxian Wang, Jun-Xuan Wang, Junxian Wang, Renato Pajarola, Peter Lindstrom, André Hinkenjann, Miguel L. Encarnação, Cláudio T. Silva, and Daniel Coming, editors, *Advances in Visual Computing*, Lecture Notes in Computer Science, pages 468–479, Berlin, Heidelberg, 2009. Springer. doi:10.1007/978-3-642-10331-5_44.

[34] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-Based Procedural Content Generation. In Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Anikó Ekárt, Anna I. Esparcia-Alcazar, Chi-Keong Goh, Juan J. Merelo, Ferrante Neri, Mike Preuß, Julian Togelius, and Georgios N. Yannakakis, editors, *Applications of Evolutionary Computation*, Lecture Notes in Computer Science, pages 141–150, Berlin, Heidelberg, 2010. Springer. doi:10.1007/978-3-642-12239-2_15.

[35] Josca van Houwelingen-Snippe, Somaya Ben Allouch, and Thomas J. L. van Rompay. Designing digital nature for older adults: A mixed method approach. *DIGITAL HEALTH*, 9:20552076231218504, January 2023. Publisher: SAGE Publications Ltd. doi:10.1177/20552076231218504.

[36] Josca van Houwelingen-Snippe, Thomas J. L. van Rompay, and Somaya Ben Allouch. Feeling Connected after Experiencing Digital Nature: A Survey Study. *International Journal of Environmental Research and Public Health*, 17(18):6879, January 2020. Number: 18 Publisher: Multidisciplinary Digital Publishing Institute. URL: https://www.mdpi.com/1660-4601/17/18/6879, doi:10.3390/ijerph17186879.

[37] J van Muijden. GPU-based procedural placement in Horizon Zero Dawn, 2017. URL: https://www.youtube.com/watch?v=ToCozpl1sYY.

[38] Li-Yi Wei. Parallel Poisson disk sampling. *ACM Transactions on Graphics*, 27(3):1–9, August 2008. URL: https://dl.acm.org/doi/10.1145/1360612.1360619, doi:10.1145/1360612.1360619.

[39] Martin Weier. Generating and Rendering Large Scale Tiled Plant Populations. 10(1), 2013. doi:10.20385/1860-2037/10.2013.1.

[40] Ondřej Šťava, Bedřich Beneš, Matthew Brisbin, and Jaroslav Křivánek. Interactive terrain modeling using hydraulic erosion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 201–210, Goslar, DEU, July 2008. Eurographics Association.