Procedurally generated realistic virtual rural worlds

Minor Dissertation presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science

by

Harry Long

Supervised by:

James Gain and Marie-Paul Cani

February 2016

I know the meaning of plagiarism and declare that all of the work in this document, save for that which is properly acknowledged, is my own.

Abstract

Manually creating virtual rural worlds is often a difficult and lengthy task for artists, as plant species selection, plant distributions and water networks must be deduced such that they realistically reflect the environment being modelled.

As virtual worlds grow in size and complexity, climates vary on the terrain itself and a single ecosystem is no longer sufficient to realistically model all vegetation. Consequentially, the task is only becoming more difficult for these artists.

Procedural methods are extensively used in computer graphics to partially or fully automate some tasks and take some of the burden off the user. Input parameters for these procedural algorithms are often unintuitive, however, and their impact on the final results, unclear.

This thesis proposes, implements, and evaluates an approach to procedurally generate vegetation and water networks for realistic virtual rural worlds. Rather than placing these to reflect the environment being modelled, the work-flow is mirrored and the user models the environment directly by specifying the resources available. These intuitive input parameters are subsequently used to configure procedural algorithms and determine suitable vegetation, plant distributions and water networks. By design, the placeable plant species are configurable so any type of environment can be modelled at various levels of detail.

The system has been tested by creating three ecosystems with little effort on the part of the user.

Acknowledgements

Firstly, I would like to thank my primary supervisor, James Gain, for his ongoing support for the past two years. His knowledge, guidance and feedback has been invaluable and essential to this research.

I would like to thank my secondary supervisor, Marie-Paul Cani, for her contributions to this work and enabling me to spend some time meeting her and fellow researchers at the INRIA research center in Grenoble, France.

I would also like to thank all members of the Procedural Interest Group at the University of Cape Town for sharing their knowledge and feedback on a weekly basis.

Also essential to this work were those that helped keep my sanity: My fellow lab-mates for their procrastination potential; my parents for their continued support; my friends for helping me switch off to revitalise and my bicycle for always being at my disposal to pedal out my frustrations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Creating detailed virtual worlds can be a tedious task for artists. Indeed, modelling terrain, vegetation, rivers, water reserves, soil, rocks, buildings and road networks for large virtual worlds manually can be extremely burdensome. This is especially true when realism is a key requirement. The increase in size and complexity of these virtual worlds mirror that of the processing capabilities of computing hardware. As a consequence, the task is only becoming more time consuming.

A popular technique that overcomes the burden of repetitive tasks is to have them automated. This involves algorithms which, given a set of input parameters, generate the required content automatically. This is called *procedural content generation* and has been successfully applied in different areas of computer graphics, including: the generation of non-repetitive textures [EL99, LLX+01, WLKT09], modelling plants [BPC+12, FZS+08, GFJ+11, Lew99], creating terrains [SKG+09, GMS09, DP10], generating river networks [DGGK11, EVC+15] and producing city landscapes [GMN14, KM07, PM01] (Figure 1.1)
A common difficulty with these methods, however, is finding the appropriate input parameters for the procedural algorithms. The correlation between the parameters and the resulting content is often unintuitive and, as a consequence, often requires iterative trial-and-error until a sufficiently acceptable result is found. To overcome this, interactive techniques are often used in an attempt to make generating the input parameters more intuitive. These range from simple paint tools such as lassos and brushes [EVC+15] to sketch-based algorithms [GMS09].

The intent of this thesis is to develop procedural algorithms to automate the generation of virtual rural worlds. More specifically, this work focuses on procedurally generating two core properties of rural worlds: vegetation and water networks. The input parameters for the procedural algorithms developed must be interactive or self-explanatory.

1.1 Research Goals

This research aims to answer whether or not procedural algorithms can be developed to automate the generation of realistic vegetation and water networks for virtual worlds. It also tries to answer if making these algorithms configurable permits the generation of a wide spectrum of virtual worlds. Finally, by optimizing the core algorithms, we will explore whether real-time interactions can be achieved.

Rephrased as research goals, this work attempts to:

- Develop procedural methods to automate the generation of realistic virtual rural worlds.
- When possible, provide real-time interaction.

One of the most important aspect of rural landscapes is vegetation. As such, our *first goal* must strongly focus on the placement of plants. The automation provided should not limit user control and the flexibility of the system. For example, it must be possible to generate worlds with varying elevations, river networks, water sources and vegetation. To test whether this is achieved, vegetation is generated and analysed for different virtual worlds varying in resource availability. The resulting vegetation is then analysed to ensure validity and plausibility.

Generating realistic water networks is also a key requirements of our *first goal*. The rivers should follow natural erosion lines and their appearance should be correlated to configured rainfall. To test the validity of the water network generation algorithm, terrains representing given locations on earth are loaded, water networks procedurally generated and compared with the real world equivalent.

Maintaining a continuous feedback loop between user action and corresponding reaction is extremely important for both user-friendliness and to optimize usage. In an attempt to meet our *second goal*, therefore, efficient algorithms must be developed in order to keep time complexity to a minimum. When suited, these algorithms should be developed to run on the GPU. To test this, the processing time of each component is analysed in detail to deduce algorithmic complexity along with the procedural parameters which bare a strong influence.

1.2 Contributions

The primary contributions of this thesis are:

- A carefully designed interface permitting users to model any environment with minimal effort.

- An efficient procedural water network generation algorithm, which relies solely on soil, rainfall and terrain properties.
- A novel vegetation generation component, which uses clustering, statistical analysis and a simulator to ensure both realism and efficiency.

1.3 Structure

To start, a detailed overview of existing work is presented in chapter 2. To better understand the individual system components of this work, an overview of the system is given first in chapter 3. Chapter 4 outlines how the base terrain is specified, associated resources configured and water content procedurally generated. The clustering algorithm used to group vertices based on associated resources is discussed and its performance analysed in chapter 5. Chapter 6 discusses the techniques used to deduce suitable vegetation and efficiently generate highly detailed and large scale plant distributions. Test environments are generated and the systems strengths and weaknesses discussed in chapter 7. Finally, in chapter 8, the thesis is concluded and future directions explored.

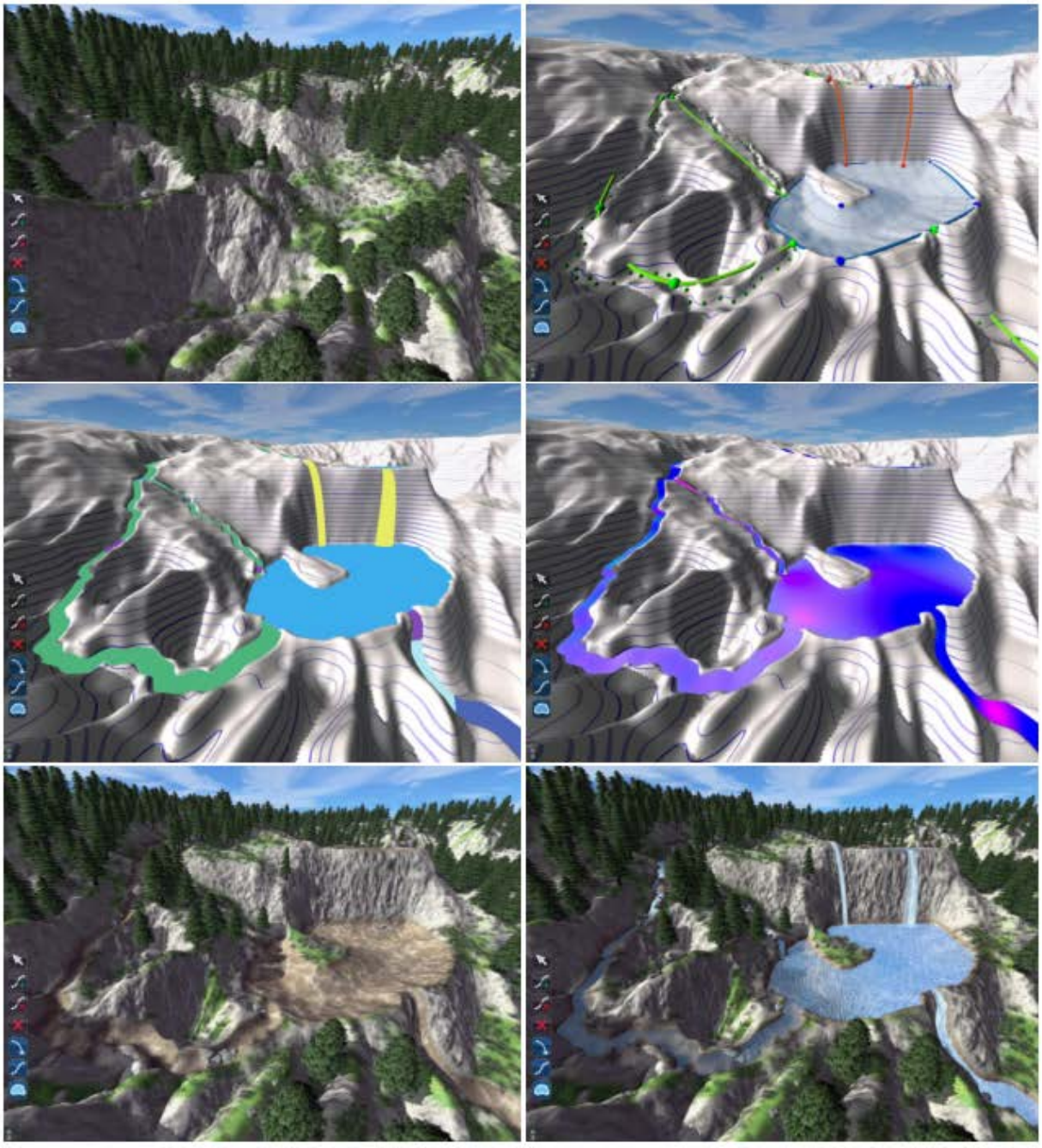

Figure 1.1: *Example of a scene with procedurally generated terrain, vegetation and water networks [EPCV14]*

Chapter 2

Background

This chapter gives an overview of previous work related to our topic. Procedural methods applied to computer graphics is a wide area of research with a large number of publications. As a consequence, we cannot pretend to review all this work. Instead, we will focus on research which is closely linked to the generation of virtual rural worlds.

Our work will not focus on modelling terrain relief but rather terrain content. As a consequence, material focused on procedurally generating terrain will not be reviewed. In this chapter will focus on the two primary constituents of rural terrains: water and vegetation.

2.1 Rivers and Streams

In this section will be reviewed the various techniques that are used to place rivers and streams on a terrain. We will split the review material into the following categories, each with dedicated sections: *Classification-based*, *Simulation-based*, *Heuristic-based*, *Fractal-based* and *Explicit*. *Classification-based* methods use pre-classified data based on real-world analysis to determine the most suited water network given a set of user-defined or terrain-defined constraints. *Simulation-based* techniques attempt to simulate natural phenomena such as gravity to determine the water networks on a given terrain. *Heuristic-based* techniques use algorithms based on real-world observation in an attempt to produce a plausible river network on the terrain. *Fractal-based* techniques use recursive algorithms in their attempt to generate plausible river networks. *Explicit* techniques require the user to specify in great detail the path the river should follow on the terrain.

The various techniques will be critiqued based on their realism, computational cost, the automation they provide and the amount of control the user has on the resulting scene.

2.1.1 Classification-based

Classification-based methods use real-world analysis of river networks to determine, based on terrain parameters (slope, soil type, flow intensity, etc.), the types of rivers best suited (stream, cascade, rapid, etc.) to given landscapes.

Emilien et al [Emi14] use classification-based techniques in their research focused on the lesser explored area of procedurally generated waterfall scenes. They model waterfalls as three separate segments: *running water*, *free-fall* and *pool*. *Running water* segments are parts of the water network in continuous contact with the terrain. *Free-fall* segments are parts which break terrain contact (i.e. waterfall). Lastly, *pool* segments represent the water-basin formed where free-fall segments meet the terrain.
Given a terrain, the user models running water and pool segments by defining control points and free-fall segments by defining a parabola. The control points for the running water and pool segments are not constrained to being in contact with the terrain as the terrain will adapt accordingly. The only constraint is that the path must continuously flow downhill. Based on this input, the system calculates plausible water flow intensities which, if required, can be overridden by the user for finer control.
The slope and water flow intensity requirements are then used as input to the waterfall classification (Figure 2.1) in order to determine realistic waterfall scenes to generate.

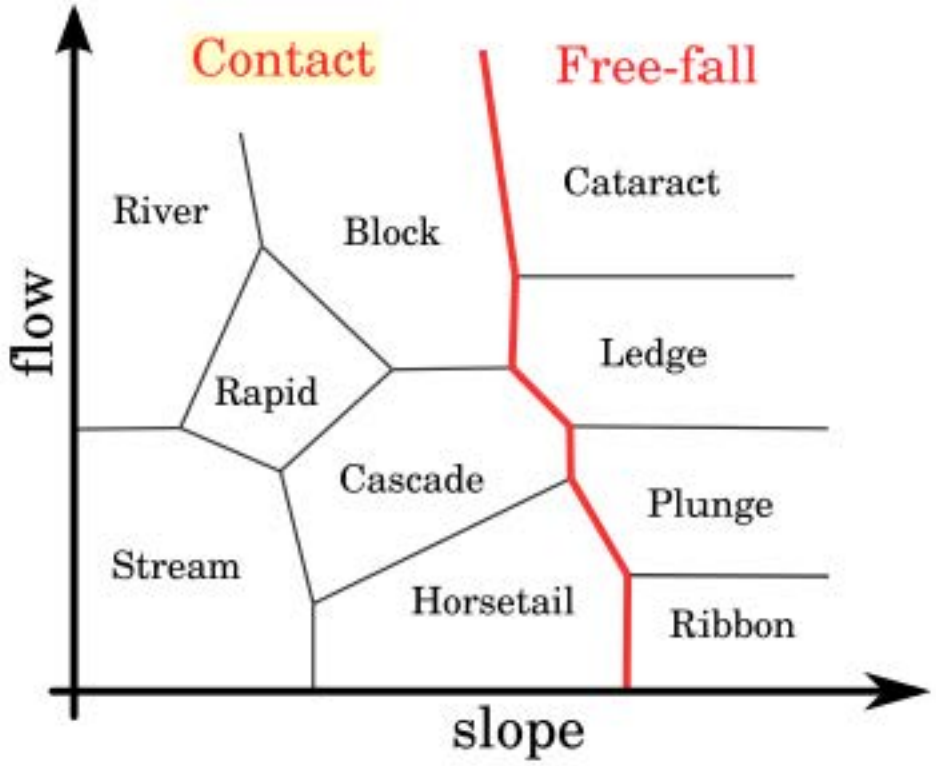

Figure 2.1: *Waterfall classifications [Emi14]*

By automatically generating plausible waterfall scenes based on trajectory input from the user, the technique strikes a good balance between automation and user control. In terms of computational complexity, the work by Emilien et al [Emi14] is able to produce complex waterfall scenes in near real-time.

2.1.2 Simulation-based: Gravity

Gravity simulating techniques attempt to determine the path water will take on a terrain by algorithmically replicating the effects of gravity.

In order to generate plausible rivers, Belhadg et Audibert [FP05] simulate the effect gravity has on water particles placed on the peaks of pre-generated ridges. To create the ridges, particle pairs are first placed at random locations on the terrain. These particle pairs are then randomly assigned a horizontal axis from which they iteratively distance themselves in opposite directions. At each iteration a new vertex is placed and its height decreased from the previous vertex based on a Gaussian distribution. To create the river networks, river particles are placed on the top of these generated ridges and a physical simulation which takes into consideration

particle velocity, particle mass and surface friction is used to model the motion of these particles on the terrain. The path followed by these particles is tracked and, when two paths intersect, their particle velocity and mass are combined. When all particles have stopped moving the simulation is deemed balanced and all particle paths which do not lead to terrain extremities discarded. The remaining particle paths are kept and form the core river network.

Similarly, in the work by Soon Tee [Teo08], water is placed at specific locations on the terrain either by the user or whilst simulating rainfall. To determine the course the placed water takes on the terrain, water is iteratively evacuated into the surrounding cell with lowest elevation. This continues until a local minima or terrain extremities is reached.

In their work on modelling the effects of hydraulic erosion, t'Ava et al. [ŠBBK08] determine the course user-placed water takes on the terrain using a hydrostatic pipe-model simulation. In order to do so, the terrain is split into equal-sized (configurable) columns and the simulation iteratively evacuates water from source to surrounding destination columns based on column elevations, fluid density and gravitational acceleration.

These techniques can produce very plausible results but have the downside of being dependent on the base terrain as their height-field must cater for river networks in the first place. This is not the case, however, for the work by t'Ava et al. [ŠBBK08] for which the gravitation simulation is used as a feedback loop to model terrain erosion. The performance of these methods depend heavily on the level of detail of the underlying water flow simulation. t'Ava et al. [ŠBBK08], for example, succeed in generating the water flow in real-time by optimizing their algorithms to use the heavily parallel architecture of GPUs.

2.1.3 Simulation-based: Erosion

Erosion-based simulations attempt to produce realistic terrains by modelling the effects of erosion. Erosion results from exogenic processes (water flow, wind, temperature) and is characterised by the removal of soil and rock from one location on earth's surface to be redeposited on another. Earth's landscape is a direct consequence of erosion and reproducing this phenomena accurately is core to procedurally generating accurate landscapes. Both Kelly et al. [KMN88] and t'Ava et al. [ŠBBK08] attempt to produce plausible terrains by modelling these effects.

In the work by Kelley et al. [KMN88], the user specifies, on a horizontal plane, the terrain outline along with the main trunk stream. The terrain outline is used to configure the terrain extremities once ported to a three-dimensional space. The main trunk stream specifies the path which the highest order water stream should follow on the resulting terrain. Given this terrain outline and the position of the initial main trunk stream, the system iteratively increments

the number of nodes which form the main trunk in order to add streams to the network. The number of new nodes added depends on the drainage area (surface area that a stream needs to channel) and the soil type as more resistant soil materials (e.g. stone) will be less influenced by water erosion than weaker ones (e.g. clay).

t'Ava et al. [ŠBBK08] are able to simulate the effects of hydraulic erosion on a terrain in real-time by using the massively parallel architectures of GPUs. Virtual pipettes are used by the user to drop water at required locations on the terrain and a gravitation simulation mentioned previously (2.1.2) is used to determine the initial water course on the terrain. Whilst the water is being routed through the terrain, the effects of *force-based* and *dissolution-based* erosion are simulated. *Force-based* erosion is a direct consequence of the the force of the water on the terrain surface (Figure 2.3). Dissolution-based erosion is a consequence of the water mass on the terrain surface under the water and is most often characterised by a smoothing effect (Figure 2.2).

In their work, Vanek et al [VBHS11] also model hydraulic errosion split the terrain into varying sized cells depending on the necessary resolution. This resolution depends on the height variance of the given location on the terrain in respect to the average variance on the entire terrain. By doing so, better accuracy is obtained for areas where hydraulic errosion will have a greater impact (i.e more height variation). Similarly to the work by t'Ava et al. [ŠBBK08], by implementing the errosion algorithm as shaders to be run on the GPU, simulations can be run in real-time.

Whether modelling erosion indirectly like in the work by Kelley et al. [KMN88] which builds the terrain around models of erosion or directly like the work by t'Ava et al. [ŠBBK08] which simulates the effect of erosion in real-time, both succeed in producing plausible terrains with integrated river networks. Fine-control over the resulting terrain, however, is limited in the work by Kelley et al. [KMN88] due to extensive automation. This is overcome in the work by [ŠBBK08] et al. by permitting the user to place water using a virtual pipette and remodel the terrain relief on-the-fly. In terms of computational cost, t'Ava et al. [ŠBBK08] are able to reproduce the effects of erosion in real-time.

2.1.4 Simulation-based: Rainfall

In order to determine where on the terrain rivers will appear, work by Soon Tee [Teo08] performs a rainfall simulation to determine both the location and quantity of water at different points on the terrain followed by a gravitation simulation (mentioned above) to determine the course of the water on the terrain. The rainfall simulation requires the user to specify wind direction and maximum rainfall. Then, starting from the source of the wind, the system sim-

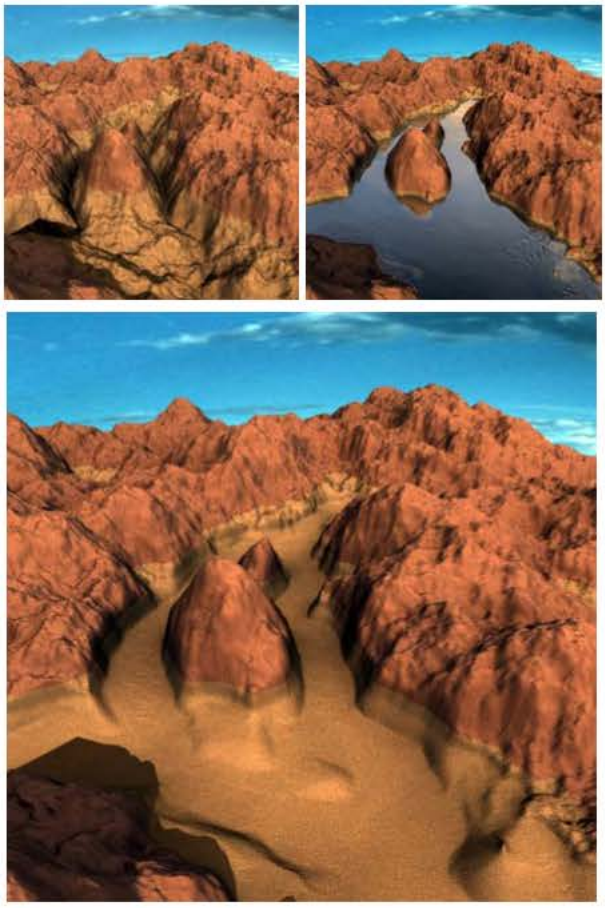

Figure 2.2: *Simulation of dissolution-based erosion erosion caused by water movement[ŠBBK08]*

ulates clouds moving in the direction of the wind with a configured velocity. When contact is made with points on the terrain, water is dropped on the corresponding cell. The amount of water dropped increases with altitude and zeroes out when all available rainfall is depleted.

Simulating rainfall in order to determine where water will fall on the terrain and therefore where river networks will form is an original approach and one that successfully generates visually plausible terrains. Requiring only wind direction, wind velocity and maximum rainfall from the user, the system provides a good level of automation. Determining the influence these inputs have on the resulting scene could be unintuitive however, and require a "trial-and-error" approach. Their algorithm creates the terrain along with the river networks in O(n) time, n representing the number of cells on the terrain.

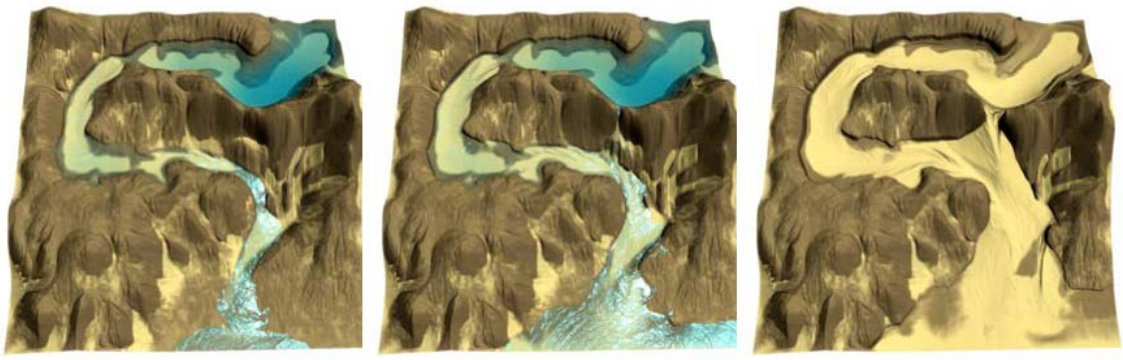

Figure 2.3: *Simulation of the effect of force-based erosion caused by running water [ŠBBK08]*

2.1.5 Heuristic-based

Heuristic approaches attempt to build river and stream networks on terrains by algorithmically reproducing key characteristics based on real-world observations.

Derzapf et al. [DGGK11] use such methods in their work based on procedurally generating virtual planets in real-time. To do so, only a very basic mesh-representation of the terrain is generated at first and detailed content is generated on demand as the user navigates through the virtual world. This method of adaptive rendering permits memory usage to me manageable whilst not compromising on realism. To ensure updates are performed in real-time, their algorithms are designed to make use of the massively parallel architecture of GPUs.
To initialise the base representation of the planets, the system first creates the base mesh with all vertices representing the sea. The system then randomly assigns a certain number of these vertices to act as seed continent vertices to spread until a user-configured land-to-water ratio is reached. To place rivers, similarly to the work by [GGG+13] et al., they first locate continental points which are on coastal edges to act as river mouths. When such a vertices are found, adjacent continental vertices are iteratively selected pseudo-randomly and connected in order to form the river network.
To assign ground altitudes to connected river vertices the system employs the following formula, starting from the river mouth:

$$a_v = a_u + e_a l_e \xi, e_a = \frac{a_{maxriver}}{l_r}$$

Where:

- a_v is the ground altitude of the current vertex.
- a_u is the ground altitude of the previously processed vertex (or zero if v is the first vertex).
- e_a is the average ground elevation.

- l_e is the length of the current vertex.
- $\xi \in [0, 1[$ is a uniformly distributed pseudo-random number.
- $a_{maxriver}$ is the user-configured maximum river altitude.
- l_r is the current river length.

When the ground altitudes have been assigned, the following formula is used iteratively on each river vertex to assign water altitudes:

$$w_v = a_v + e_w l_e, e_w = \frac{\epsilon_{river}}{l_{cr}}$$

Where:

- w_v is the water altitude of the current vertex.
- a_v is the ground altitude of the current vertex.
- e_w is the average water elevation.
- l_e is the length of the current vertex.
- ϵ_{river} is the user-configured maximum river depth.
- l_{cr} is the distance from the current vertex to the river spring.

All randomness in these algorithms depend on a configured seed value enabling the virtual world to be easily reproducible.

This heuristic approach offers an extensive level of automation and, as a result, fine control over the resulting scene is lost. Rather than generating virtual worlds fitting specific user requirements, it is more suited to generating plausible virtual worlds which fit loose constraints (e.g. maximum river altitude, maximum water depth, river stream must flow downhill, etc.).

2.1.6 Fractal-based

Another technique employed to produce river streams is by employing fractal-based algorithms. Such methods use recursive splitting and string rewriting to determine plausible river networks.

In their work, Pmsinkiewicz et al. use a fractal-based technique based on midpoint-displacement to procedurally generate plausible rivers on a terrain. Midpoint-displacement is most commonly used for procedurally generating realistic terrain height-maps and works follows follows: Given a starting triangle representing a terrain A, midpoint-displacement iteratively subdivides A it into four smaller triangles. Each time new triangle vertices are created they are displaced vertically by a random offset. This process is repeated until a given recursion limit is reached. See Figure 2.4 for an example of a single iteration of the process.

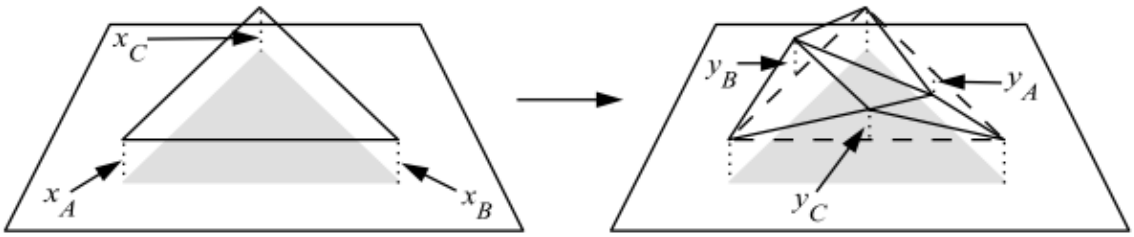

Figure 2.4: *A single iteration of midpoint displacement for the creation of mountains [PHM93]. New vertices y_A, y_B and y_C are created and shifted vertically by a random offset*

To adapt this method to the generation of rivers on the terrain, rather than vertically displace newly formed triangle vertices, there edges are labelled as *entry*, *exit* or *neutral* (Figure 2.5). An *entry edge* defines the point of entry for the river into the triangle, an *exit edge* the point of exit and a *neutral edge* prevents the river from passing through.

When a production step is applied and a triangle split, the following constraints must be applied:

- An entry edge must split into an entry and a neutral edge.
- An exit edge must split into an exit edge and a neutral edge.
- A neutral edge must split into two neutral edges.
- The newly formed edge-pairs within the triangle must either be "entry/exit" or "neutral/neutral".

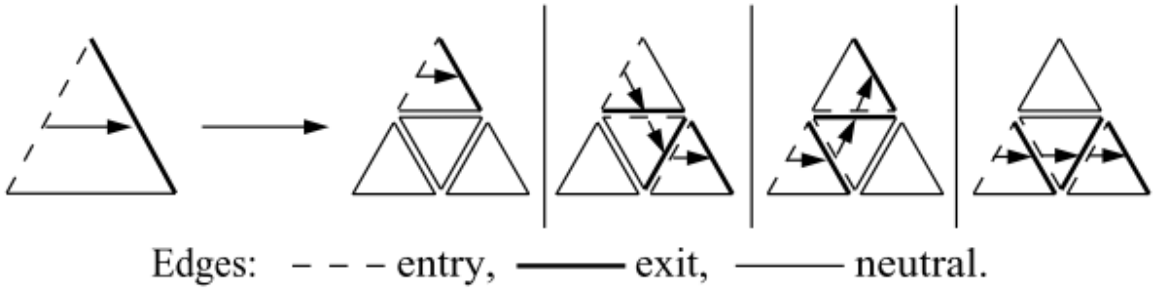

Figure 2.5: *Single production of midpoint displacement adapted to river generation [PHM93]. Given the initial triangle, four valid split scenarios.*

One difficulty with this technique is to ensure two adjacent triangles are coherent once split. That is, that the exit edge of one coincides with the entry edge of the other. To solve this, the location of edge vertices are used as the key to a random number generating hash table which,

based on its output number, determines the segment that will be crossed by the river, if any.

In their work, Gnevaux et al. [GGG+13] use fractal-based string rewriting to produce the river networks on the terrain. Once initial nodes have been selected to act as the river mouth, rewriting grammar is used to perform river node expansion. Configured values of $\rho_a, \rho_s and \rho_c$ influence the probability of selecting productions favouring asymmetric branching, symmetric branching and continuation without branching, respectively. The position for the new node is then selected based on the following constraints:

- It should be at a minimum distance from existing nodes and edges.
- The new node should be at a greater distance from the terrain contour.
- The new node should be compatible with the elevation constraints of existing nodes.

If a position satisfying these constraints is found, a new node is added at the given position and the process is repeated.

Both these techniques are successful in generating realistic river networks on terrains. The user, however, is limited in the amount of control he has over the resulting rivers. In the work by Gnevaux et al. [GGG+13], for example, this is limited to specifying the preferred river branching behaviours. In terms of performance, Gnevaux et al. [GGG+13] are able to produce terrains of several hundred square kilometres in a matter of seconds.

2.1.7 Explicit

Explicit techniques use explicit input from the user to determine locations and properties of the river networks to generate.

Flood-filling is such a technique and is used in the work by Soon Tee [Teo08] to permit users to place water reserves (e.g. sea, lakes, etc.) by clicking a single point on the terrain. This point which will act as the seed point for the water surface and will propagate iteratively to surrounding points at lower heights until all such points have been depleted.

Smelik et al. also use explicit techniques to create an interactive system permitting users to model a complete virtual world with content ranging from rural features (mountains, rivers, etc.) to man-made ones (buildings, road networks). When modelling the virtual world, interactions are split into two modes: **Landscape** and **Feature**. *Landscape mode* permits the designer to paint ecotopes onto the terrain using traditional image editing tools. These ecotopes are predefined by the user and encompass both elevation and soil material information. In *feature mode*, the user is able to place terrain content, including rivers. To do so, similarly

to the interface provided by Emilien et al. [Emi14], the user sketches vector lines outlining the core path of the river and, based on this, a suitable course is plotted through the landscape. Other terrain features to which the river takes precedence adapt accordingly. For example, if the river is plotted to pass through a forest, trees on the rived bed and bank will be removed automatically.

Rather than placing vector-lines, the work by Soon Tee [Teo08] and t'Ava et al. [ŠBBK08] permits users to click single points on the terrain which will act as the water source. The system then automatically generates a plausible path for the water down slope of the terrain.

As these methods provide very little automation in terms of guaranteeing consistency in the scene, the resulting realism is very much user-dependent. Real-time action-reaction feedback is essential with explicit modelling and so the majority of the methods run in real-time.

2.1.8 Summary

Each technique has it's associated pros and cons and so choosing which one is best suited depends heavily on the requirements of the system. For example, if the terrain is fixed, using techniques which simulate real-time erosion of the terrain would be ill-suited. Similarly, if fine control over the resulting scene is necessary, heavily automated procedural methods which generate realistic scenes using very little user input would certainly not meet the requirements of the system. In this section we will summarize the pros and cons of the individual techniques in table form. These techniques will be rated based on:

- *Automation*: The level of automation the technique provides.
- *Realism*: The realism of generated scenes.
- *Computational efficiency*: The techniques efficiency in terms of computational resources.
- *User-control*: How much control the user has over the final scene.

In our system, the base terrain will take the form of a preloaded height map. Modifications to this terrain and modelling new terrains will be out-of-scope and, as such, all techniques which require such behaviour can be discarded.
Rainfall is a vital requirement to plant life and, as such, gathering rainfall data will be essential to model realistic vegetation on the terrain. Using this rainfall data along with soil properties, it is possible to calculate the amount of standing water which has not been absorbed by the soil. Using this, along with a gravitation simulation, it should be possible to determine main river networks on the terrain based on water builds up. The water drainage simulation employed here will be gravitational based and inspired from the hydrostatic pipe-model of t'Ava et al.

	Automation	**Realism**	**Computational Efficiency**	**User-control**
Classification-based	Good	Very Good	Good	Very Good
Fractal-based	Excellent	Good	Good	Poor
Explicit	Poor	Fair	Very Good	Excellent
Simulation-based				
Gravity	Very Good	Good	Fair	Fair
Erosion	Very Good	Very Good	Good	Fair
Rainfall	Very Good	Good	Good	Poor

Table 2.1: *Summary of river placement techniques*

[ŠBBK08]. It should be optimized to work in real-time and its duration controllable by the user in order to have fine-control over the size and depth of the resulting river networks (i.e. a longer simulation will drain more water and, as such, the river networks will be less intense).

2.2 Vegetation

Vegetation is core to rural landscapes. The species present along with their associated densities create a relationship between ecosystems and areas on earth on which resources are adequate. To ensure realism in virtual environments, much emphasis must be put on efficiently modelling these underlying ecosystems.

This section will review different methods to generate suitable vegetation for virtual worlds. These methods can be split into three main categories: *Explicit Instancing, Probabilistic Instancing* and *Plant Growth Modelling*.
Explicit Placement require explicit user-input to directly or indirectly pinpoint exact locations for individual plant instances.
Probabilistic Placement methods use statistical models to generate suitable vegetation.
Simulators attempt to algorithmically reproduce plants battling for available resources.

We will measure the success of these techniques based on the level of automation they provide, the realism they achieve, their computational cost and their adaptability. Adaptability, here, represents the ease at which a given technique is able to model a number of different vegetation scenarios.

2.2.1 Explicit Placement

Explicit placement methods require input from the user to determine the location and properties of individual plants.

Arnaud et al. [EVC+15] permit users to insert individual plants manually by simply clicking a given location on the terrain. To overcome the tedious task of manually placing individual plants on large terrains, the system is able to analyse existing distributions for reproduction. For example, to generate a large forest, the user is only required to generate a small subsection which can then be used to reproduce it on any scale (Figure 2.6)

Similarly, Deussen et al. [DHL+98] allow users to use grayscale raster images as input to specify terrain vegetation. The location of individual plants is determined by pixel location whereas plant properties are correlated to pixel intensity.

In their work focused on improving the realism of roadside landscapes, Andujar et al. [ACV+14] use orthophotos as input to determine the location and properties of individual plants. Unlike ordinary aerial photographs, aerial orthophotos use normalisation techniques

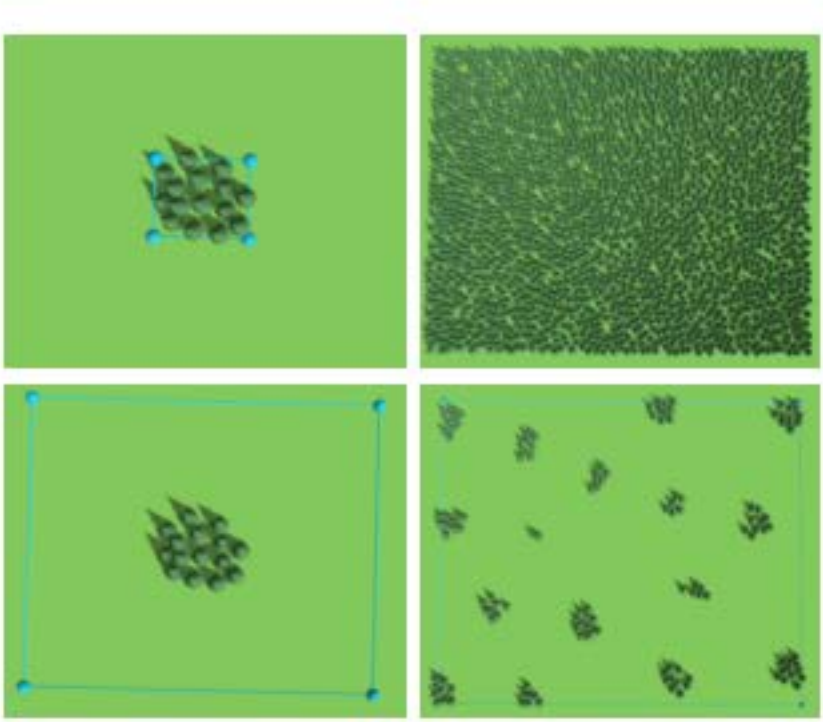

Figure 2.6: *Using explicit placement as input examplars for reproduction [EVC+15]*

to take into account terrain relief and camera tilt. The result is an image with uniform scale throughout which, similarly to a map, can be used to accurately measure distances between points. These orthophotos are used to measure the distances between plants. To later reproduce the roadside landscape, they use a dart throwing algorithm to place individual plants whilst respecting the measured distances.

Explicit placement methods provide significant user control over the resulting virtual world. However, as there is *little to no automation* of this process, it can be very tedious and time consuming for the user. This is especially true when the virtual world being created are very large (e.g. open world video games). An advantage of this limited automation, however, is that modifications are most often very small and are therefore performed in real-time.
The *adaptability* of these methods are very poor. Running a different scenario would most often involve starting the entire plant placement process again.
Creating vegetation for large virtual worlds using these methods is extremely strenuous and, as a consequence, realism is often compromised.

2.2.2 Probabilistic Placement: Radial Distribution Analysis

Work by Emilien et al. [EVC+15], Boudon et al. [BM07] and Lane et al. [LP02] use radial distribution analysis to convert to metric form the underlying plant distributions of input examplars. The data generated by the analysis stage can later be used to synthesise, at any scale,

Figure 2.7: *Reconstructed roadside vegetation using orthophotos [ACV+14]*

new point distributions which respect the characteristics of the input exemplar.
For example, by analysing the positions of individual plants in a small subset of a forest and using it as the input exemplar, it is possible to reproduce it at a much larger scale in order to model its full size counterpart.

Analysis Generating the analytical data involves measuring the distances between individual points of different categories from the input examplar. For plant distribution analysis, the points represent individual plants and the categories represent the different species.

Before performing the analysis, the following parameters are configured:

- $\mathbf{R_{min}}$: The minimum distance from which point distances need to be analysed.

- $\mathbf{R_{max}}$: The maximum distance after which point distances don't need to be analysed.

- **Bin size**: When analysing the distances of given points, it is necessary to aggregate the points which reside at similar distances into bins. The bin size is the range represented by a single bin.

- **Category ranking**: If there are points of multiple categories (different plant species, for example), it is necessary to generate a category ranking. This ranking will affect the order

in which the points are placed during the reproduction and the analysis data to generate.

A core part of radial distribution analysis is generating pair correlation histograms for the necessary category pair combination. A pair correlation histogram H_{AB} represents the variation in the distance between points of of category C_A and C_B ranging from R_{min} to R_{max} in *bin size* increments (Figure 2.8). Pair correlation histograms do not need to be generated for each category pair combination. Given a category x, it is only necessary to produce the pair correlation histograms H_{xy} for $R_y \leq R_x$ where R_y is the rank of category $_y$. The reason for this becomes apparent when discussing the reproduction 2.2.2

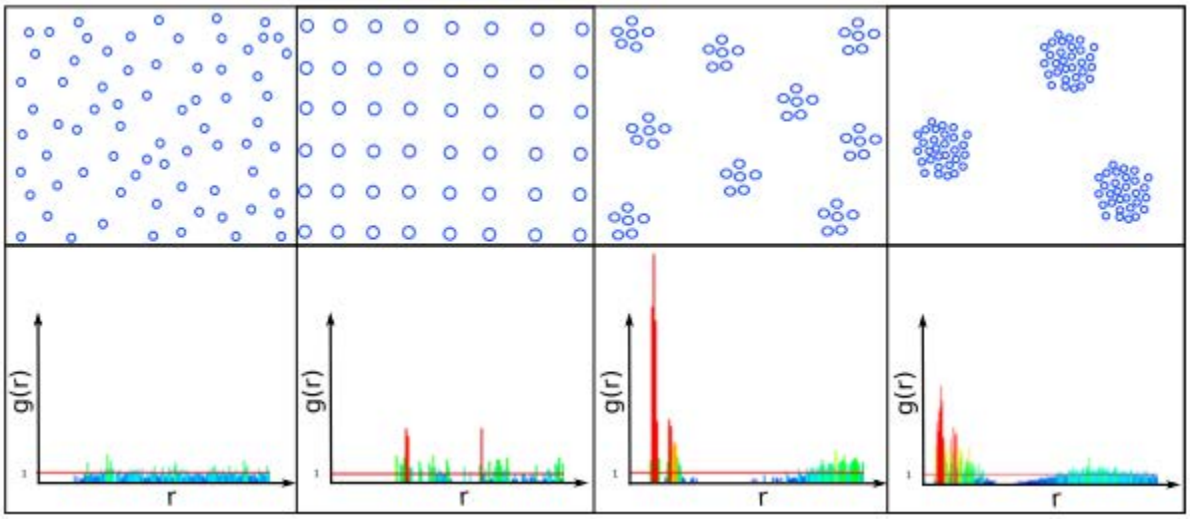

Figure 2.8: *Point distributions with associated pair correlation histogram [Emi14]*

To generate the pair correlation histogram H_{AB}, the algorithm iterates through each reference point of category C_A and, for each destination point of category C_B at a distance between R_{min} and R_{max}, increments the relevant bin in the histogram. In Figure 2.9, for example, are being measured the points that lie within the annular shell of radius r with bin size d_r (area d_A).

Because of their larger circumference, the coverage area of annular shells get larger as the distance bin being measured increases. In other words, $A_r < A_{r+1}$ where A_r is the area covered by the annular shell starting at distance r. A direct consequence of this is that annular shells at further distances will naturally be prone to containing more points. To counter for this, normalisation is performed based on annular shell area.

The radial distribution analysis function h_{rdf} is as follows:

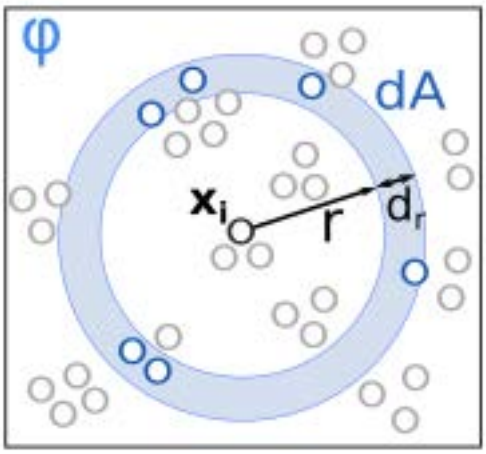

Figure 2.9: *Radial distribution analysis*

$$h_{rdf}(k) = \sum_{x_i \in X} \sum_{y_j \in Y \& k d_r \leq d(x_i, y_j) < (k+1) d_r} \frac{A}{d_A n_x n_y}$$

Where:

- *hrdf(k)* is the k-th value of the pair wise histogram.
- X are the points of category X (reference points).
- Y are the points of category Y (target points) .
- d_r is the annular shell width.
- $d(x_i,\ y_j)$ is the distance from point x_i to y_j.
- A is the total analysed area.
- n_x and n_y are the number of points of categories x and y respectively. Note that pairwise histograms also need to be calculated for points of the same category. In this situation, category x and category y would be the same.
- d_A is the area of the annular shell being analysed.

Conceptually, this formula iterates through the points of each source category (X) and determines the number of points of the different target categories (Y) at incremental distances of the annular shell size. Normalization is performed based on the area of the annular shell as there would naturally be more points in annular shells at further distances from the source point as the area covered will be larger.

Reproduction In order to reproduce the distribution of the input exemplar, points are added iteratively whilst matching as closely as possible the corresponding pair correlation histogram data calculated during the analysis stage. Metropolis-Hastings sampling [HLT+09] is the most common way to do this. It involves performing a fixed number of point birth-and-death perturbations. A change from the initial arrangement X to the new arrangement X' is accepted with probability R, where:

$$R = \frac{f(X')}{f(X}$$

f(X) is the probability density function (PDF) of a given arrangement and is expressed as:

$$f(X) = \prod_{C_{Y_K} \leq C_X} \prod_{x_i \in X} \prod_{y_i \in Y_k} h_{X,Y_k}(d(x_i, y_j))$$

Where:

- C_y and C_x represent categories Y and X, respectively.
- X are all points of category X.
- Y are all points of category Y.
- $\mathrm{h_{X,Y_k}(d(x_i, y_j))}$ is the value retrieved from the pairwise histogram of categories X and Y given the distance between points x_i and y_i.

Intuitively, the PDF defines, given a set of points, the aggregate strength of the current distribution. It does so by iterating through each point and determining the points at a distance within $\mathrm{R_{min}}$ and $\mathrm{R_{max}}$ and of a higher or equal category ranking. Given this information, the generated pairwise histograms are used to calculate an aggregate strength.

Because the PDF formula is a product, calculating it for a new layout X' with appended/removed point P only involves calculating the PDF for the single reference point P. As a consequence, reproduction can be performed very efficiently. In their work, Emilien and Cani [EVC+15] are able to perform analysis and reproduction in near real-time.

When using this technique to reproduce a plausible plant distribution, Boudon et al. [BM07] take it one step further by enabling plant crowns to deform based on predefined elasticity parameters. Because the crowns are not constrained to being circular, they can deform to facilitate the survival of plants at a lower height.

2.2.3 Probabilistic Placement: Predefined Ecosystems

In their work, Hammes el al. [Ham01] predefine ecosystems along with their preferred environment. These environments are defined in terms of:

- Elevation: All plant species have an upper limit after which temperature or oxygen levels are ill-suited.
- Relative elevation: The local changes in height. Local minimums tend to be valleys and therefore wetter with less illumination. Local maximums, on the other hand, tend to me ridges which are dryer and much more exposed.
- Slope: Gradient has a direct impact on the quality of the soil and therefore the plants which can grow. When slopes get steeper, plants tend to get much smaller as they struggle to get required nutrients from the soil.
- Slope direction: This has a direct effect on sunlight exposure. Southern facing slopes in the northern hemisphere will have a greater exposure to the sun and vice-versa for the southern hemisphere.

All these ecosystems are stored in a database and, when vegetation is to be placed on the terrain, the most suitable ecosystems are chosen based on the terrain properties mentioned above. See Figure 2.10 for an example landscape generated using this technique.

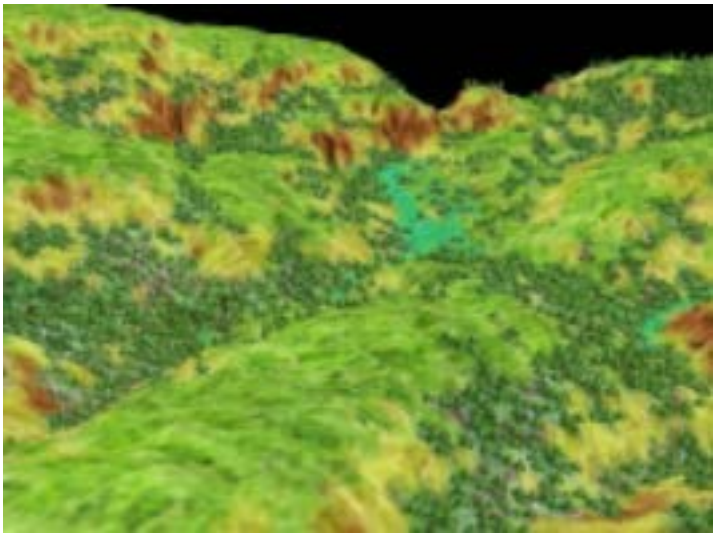

Figure 2.10: *Vegetation generated using predefined ecosystems [Ham01]*

2.2.4 Probabilistic Placement: Conclusions

Probabilistic Placement permit users to specify only small portions of input data to populate large areas. For the *Radial Distribution Analysis* approach, this input data would be in the

form of an input distribution. For the *Predefined Ecosystems* approach, it would be a predefined ecosystem along with its preferred environment. Although this automation does ease the task for artists, specifying accurate input data is still crucial to produce realistic vegetation. Consequently, although the realism achieved by these methods is generally good, their adaptability is still limited.
Thanks to the use of efficient algorithms, the computational complexity of these methods are often low and real-time updating is achievable.

2.2.5 Simulators: Plant Growth Modelling

Plant growth modeller attempt to algorithmically reproduce the laws of nature with such precision that they can be used in agronomical sciences and forestry to estimate and maximize crop yield. To achieve this, such simulators go into great detail to model the available resources. For example, work by Soler et al. [SSBR01, SED03] splits single plants into geometrical organs with unique light transmittance and reflectance properties. By doing so, light propagation within the plant can be simulated in order to determine the aggregated photosynthetic potential. This work, along with that of Yan et al. [Yan04], base their simulators on two vital and widely accepted laws of nature:

- *Law of the sum of temperatures*: Plants grow in cycles which vary from days to years depending on the specie. The law of the sum of temperatures states that the frequency of these cycles is proportional to the sum of the daily average of the temperatures.

- *Law of the water use efficiency*: The amount of fresh matter fabricated by a plant is proportional to the water evaporation of the plant. This factor is called the water use efficiency.

Water evaporates during photosynthesis as the plant exchanges water for carbon dioxide. Based on this and the law of water use efficiency outlined above, the amount of fresh matter produced (i.e growth) for a given plant is directly correlated to the amount of photosynthesis performed. Using this, Soler et al. [SSBR01] apply the following formula to calculate the amount of fresh matter, $Q_m(t)$, created by a given plant at time t:

$$Q_m(t) = \sum_{x=1}^{N(t)} \frac{E(x,t)}{R}$$

Where:

- $E(x,t)$ is the potential for matter production of the x-th leaf at the t-th cycle. It is proportional to the incoming radiant energy up to a certain threshold, after which it remains constant.

- R is the hydraulic resistance of the given leaf. This resistance is what limits water evaporation (photosynthesis) and therefore growth. It varies depending in the species and surface area.

Intuitively, this formula calculates the total available fresh matter, Q_m, that can be produced for an individual plant P at a given time t, by calculating the photosynthesis potential of each individual leaf of P given the current lighting.

Using this, the algorithm iterates through growth cycles with a frequency that is calculated based on the *law of the sum of temperatures* mentioned above. Each growth cycle performs the following two steps:

1. The lighting and therefore photosynthesis potential of each individual leaf of the plant is calculated. This is then used to calculate, as above, the quantity of fresh matter produced.
2. The fresh matter is then distributed to different organs of the plant according to an associated organ strength.

Both Palubicki et al. [PHL+09] model plant growth based on two vital requirements: space and light availability. The space is calculated based on the presence of existing buds and trees in the surrounding. In terms of lighting, a shadow propagation algorithm is used to efficitently determine an estimation of the exposure of each bud to direct sunlight. In their work on TreeSketch, a tablet application aimed at plant modelling, Longay et al. [LRBP12] also constrain plant development to light and space availablility.

In their work, Pirk et al. [PNH+14] bring wind into the equation. Particle-based wind dynamics are implemented to emit varying strengthed wind at pre-configured directions. Collision detection is then performed on the particles in order to determine the impacted plant instances. Winds can then influence plant growth in many ways by, for example: changing the direction of growth and breaking off branches and buds. More so, if a forrest is modeled, this detailed wind simulation accurately depicts the plants which are impacted and those that are protected from the wind by other plant instances.

By going into such detail, these simulators produce very realistic simulations of the evolution of plants. For example, to maximize growth, plants are able to grow in direction of the light source (Figure 2.11).

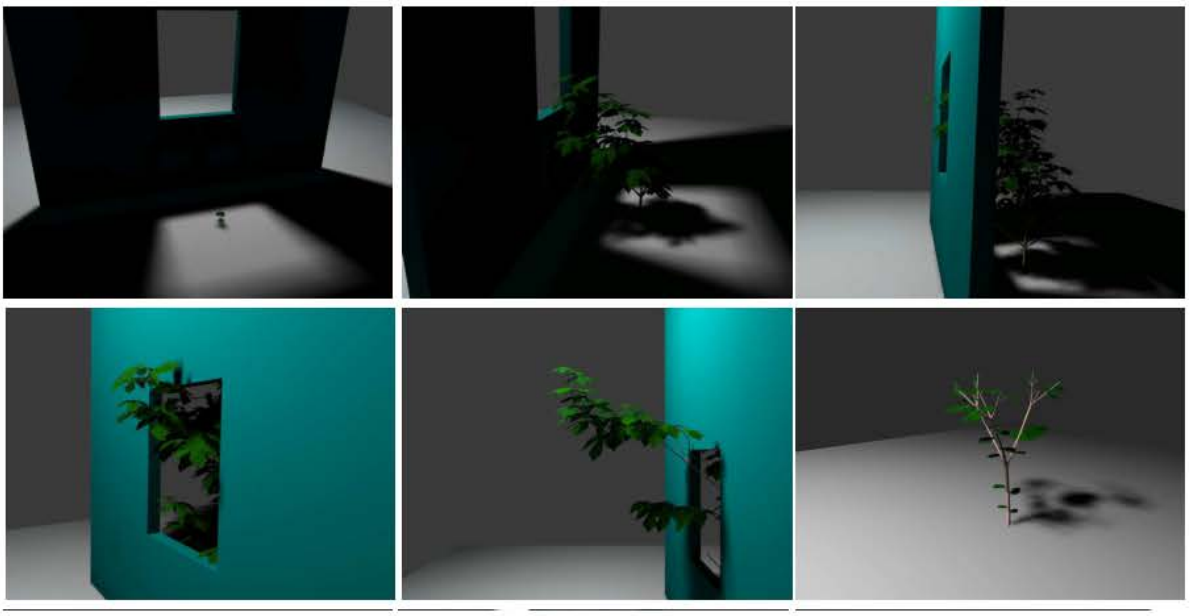

Figure 2.11: *Plant growing towards light source [SSBR01]*

2.2.6 Simulators: Ecosystem Simulators

Ecosystem simulators use procedural methods to algorithmically reproduce the competition for resources that occurs in nature during plant growth. In nature, this competition is an extremely complex process and so reproducing it exactly would be infeasible. Instead, a simplified model of this ecological process is implemented. During these simulations, available resources fluctuate and each plants strength is continuously recalculated based on its associated properties. This strength directly affects the plants growth and chance of survival.
Such plant properties include: age; vigor; shade tolerance; humidity requirement and temperature requirements. Amongst others, the resources modelled include: available illumination; available humidity; temperature and slope.

The aim of ecosystem simulators is to determine, given an initial state $\boldsymbol{S_t}$ of the system at time $\boldsymbol{t}$ and a simulation time $\boldsymbol{n}$, the state $\boldsymbol{S_{t+n}}$.
The state of the system represents individual plant instances with associated location and properties.

Lindenmayer systems, commonly referred to as L-systems, use a formal grammar along with a set of production rules to iteratively create larger strings from a starting string called the axiom. Such systems are commonly used to model plants and plant growth [PL90, DCSD02, BPC+12, PHM93].
An extension to basic L-systems, referred to as open L-systems, adds a communication grammar which permits the set of production rules to behave differently depending on predefined conditions [Pru96]. In their work modelling the growth of spruce trees, Berezovskava et al.

[BKK+97] use different production rules depending on local bud density. This is a simplified representation of buds competing for available light.

By introduction multiset L-Systems, Lane and Przemyslaw [LP02] extend L-systems yet further to model an ecosystem simulator. The production rules for multiset L-systems work in two stages. The first, identical to basic L-Systems, produces a new string given an input string and production rule. The second, splits the resulting string into new sets using a predefined separation symbol. In their work, the different sets represent different plant instances, thus enabling new plants to spawn during the production steps. When building their L-System, Lane and Przemyslaw [LP02] focus on reproducing three important properties of nature, each distinctly testable to determine the plausibility of the results:

- *Self-thinning*: When plants grow, their resource requirements increase and, as a direct consequence, inter-plant competition for resources increases. Eventually, the competition becomes too intense and resources too scarce leading to more vigorous plants starving smaller plants. At this point, self thinning begins and plant densities decrease.
- *Succession*: Given plant species A with a fast growth rate and species B with a slower growth rate but higher shade tolerance. At first, the faster growing species A will dominate and flourish but, with time, the slower growing but more shade tolerant species B will flourish and dominate.
- *Propagation*: Plants often propagate in clusters surrounding the seeding plant.

The L-System they implemented contains different production rules to represent the different properties of nature mentioned above. A single simulation and the corresponding output can be seen in Figure 2.12.

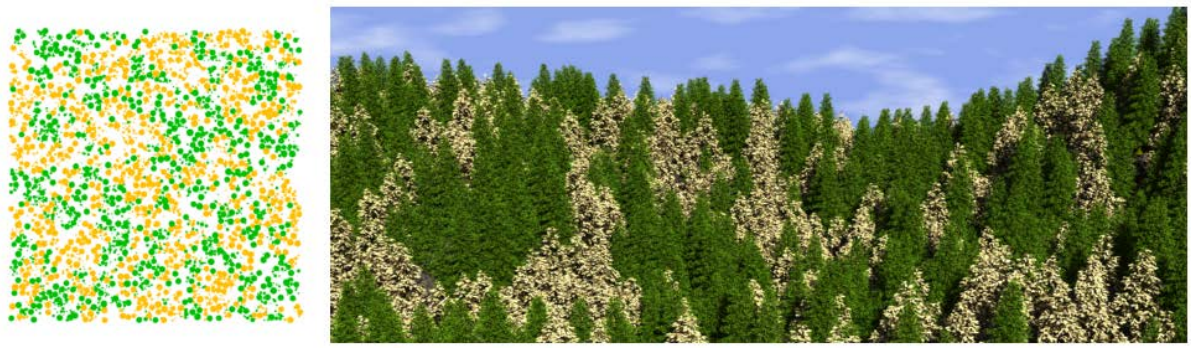

Figure 2.12: *Plant placement using an ecosystem simulator modelled by L-Systems [LP02]. Left: Result of the simulation where orange circles indicate the positions of poplar trees and green circles the positions of spruce trees. Right: Reproduced virtual world where the location of individual plants is deduced from the output of the simulator.*

Work by Deussen et al. [DHL+98] also uses L-Systems as the basis for an ecosystem simulator. As an extension to the work by Lane and Przemyslaw [LP02], they introduce the notion of soil humidity and an associated soil per species humidity preference.

Benes et al. [BMJ+11] also use ecosystem simulators to generate plausible vegetative distributions in city environments. They do so by extending the work by Deussen et al [DHL+98] and introducing a plant management factor. This factor is based on the location of the given plant within the city. At the border of the city, this value will be extremely low whereas in the center, where plant management is high, so will the factor. This then directly influences plant seeding and growth during the ecosystem simulation.

A direct consequence of the automation provided by these ecosystems is that fine control over the final vegetation content is lost. Deussen et al. [DHL+98] overcome this, however, by offering a hybrid approach where the ecosystem simulator is first used to populate the entire terrain and explicit instancing is used thereafter for the detailing .

Another weakness of procedural ecosystems based on L-Systems worth mentioning is that the communication parameter is binary; in the work by Lane et al. [LP02] a plant will be dominated as soon as its radius intersects another larger plant, at which point it will die with a set probability. This probability of death will stay constant and will not increase as this domination increases. Similarly, in the humidity model of Deussen et al. [DHL+98], a plant has a preference for wet or dry areas and there is no notion of a measurable humidity preference range. This could prove problematic to model species which are able to adapt to a multitude of environments with varying resource availability (e.g. grass).

2.2.7 Simulators: Conclusions

Probably the main advantage of simulators over other approaches is the level of *automation*. Running simulations is done with easy and requires very little input from the user.
Although the adaptability of these methods is also impressive, it is limited by the necessity to configure the properties for individual species. This is especially true for *Plant Growth Modelling* approaches where topological data must be configured. Obtaining topological data often involves real-world analysis of the plants growth cycles.
Computational cost is often high when using simulators. The extent of which is dependant on the level of detail and the number of plants being simulated simultaneously. For example, in the highly detailed simulations of Soler et al. [SSBR01], simulating 45 cycles for a single plant takes approximately 15 minutes.

2.2.8 Summary

Which technique (Explicit, Probabilistic or Simulators) to use entirely depends on the requirements of the system. For example, if realism is the key priority then ecosystem simulators able to provide botanical realism would be the most suitable approach. Choosing the technique is therefore all about minimizing the associated compromises. In Table 2.2 we summarize the pros and cons of the individual techniques based on the following criteria:

- *Automation*: The level of automation the technique provides. That is, how little user input is needed.
- *Realism*: The level of realism with which the technique models real-world ecosystems.
- *Computational efficiency*: The techniques efficiency in terms of computational resource requirements.
- *Adaptability*: How well the technique can adapt to model different scenarios.

	Automation	**Realism**	**Computational Efficiency**	**Adaptability**
Explicit Placement	Poor	Poor	Excellent	Poor
Probabilistic Placement				
Radial Distribution Analysis	Good	Very Good	Very Good	Fair
Predefined Ecosystems	Good	Fair	Very Good	Poor
Simulators				
Plant Growth Modelling	Excellent	Excellent	Poor	Fair
Ecosystem Simulators	Excellent	Very Good	Fair	Good

Table 2.2: *Summary of vegetation placement techniques*

Given a set of plant species, available resources and terrain, our system must be able to specify the locations of individual plants. The output must be: visually realistic; easily scalable in order to be able to re-run simulations with different input species; computationally efficient to ensure the effect of user actions appear in close to real-time.
Given these requirements, a hybrid approach is best suited which combines the adaptability and realism of ecosystem simulators with the computational efficiency of probabilistic placement. More specifically, the ecosystem simulator will inspire heavily from previous work by Deussen et al. [DHL+98] and Lane and Przemyslaw [LP02]. Radial distribution analysis, inspired by

the work by Emilien et al. [EVC+15], will be used for probabilistic placement.
Computationally expensive ecosystem simulator runs will be performed beforehand in order to acquire the necessary distribution data. This data will then be stored in order for it to be queried at a later stage without having to redo expensive simulations. When placing vegetation in the virtual world, pre-calculated distribution data will be queried and probabilistic instancing used to fill user-defined areas with suited plant species and realistic distributions.

Chapter 3

System Overview

In this chapter are discussed the design requirements, motivations and constraints. Given this, a chosen architecture is discussed along with the individual components and how they fit together (as illustrated in Figure 3.1).

To conclude, the *limitations* of the system will be discussed and *comparisons and differences* drawn with previous work in the field.

3.1 Design Motivations

This work attempts to bridge the gap between ecosystem simulators and probabilistic placement by using the simulator to determine realistic vegetation distributions on a a predefined area and radial distribution analysis to efficiently reproduce it at any scale.
By limiting the coverage area of the ecosystem simulator, it is possible to generate more accurate vegetation distributions by performing a more accurate simulation whilst keeping simulation times manageable.
A combination of user-focused input tools, procedural methods and an efficient clustering algorithm is used to split the terrain into clusters based on the resources associated with each terrain vertex. By doing so, the system automatically determines the areas of the terrain which differ sufficiently in resources to necessitate a new vegetation distribution to be calculated.

Given this, to meet the requirements of this research, the final system must present an interface which permits users to load a virtual terrain and easily specify position, clustering and resource properties. With all this information, the system should automatically determine terrain clusters and associated suitable plant species to suggest to the user. Finally, given the selected species, a suitable vegetation distribution must be generated.

Given these constraints, the core system can be split into the following building blocks:

- The *terrain and resource gatherer* whose purpose is to gather the terrain and associated resources.
- The *resource clusterer* to cluster points of the terrain into clusters based on resource similarities.
- The *plant data manager* which is responsible for gathering and storing plant species with their associated properties. It is also responsible for determining suitable plants given the calculated resources.
- The *ecosystem simulator* which uses plant species data obtained from the *plant data manager* to simulate plants battling for resources in a given environment.
- The *distribution analyser and reproducer* which is able to store a plant distribution output from the ecosystem simulator and efficiently reproduce it at much larger scales.

Although these building blocks will work together to fulfil the requirements of our system, they are very much independent. As such, they are all implemented as seperate components which can function as standalone components or as libraries. Running individual components separately is also valuable as, for example, it makes it possible to use the ecosystem simulator to pre-calculate plant distribution data and, in turn, the distribution analyzer to calculate the underlying distribution characteristics.

3.2 Architecture

3.2.1 Resource Gatherer

Vegetation requires resources to grow and the distribution of these resources identifies a given species and associated it with a given climate and, subsequently, location on earth. Determining resource data is essential, therefore, to generating realistic virtual worlds as it is vital to determining vegetation distribution patterns. The purpose of the resource gatherer is to determine, for each terrain vertex: *sun exposure*, *soil humidity*, *temperature* and *slope*. Figure 3.2 illustrates the output of the resource gatherer along with the user inputs required.
The latitude and orientation of the terrain must be specified by the user in order to determine the sun position throughout the year. The *sunlight exposure* calculation then determines, given this information and the terrain relief, the average daily illumination (in hours) received by each terrain vertex for each month of the year. To calculate the average illumination for a given month, the trajectory of the sun is calculated for the fifteenth day (discussed later in the body of the thesis).
In order to calculate the *soil humidity* for each month, the soil infiltration rate, monthly rainfall

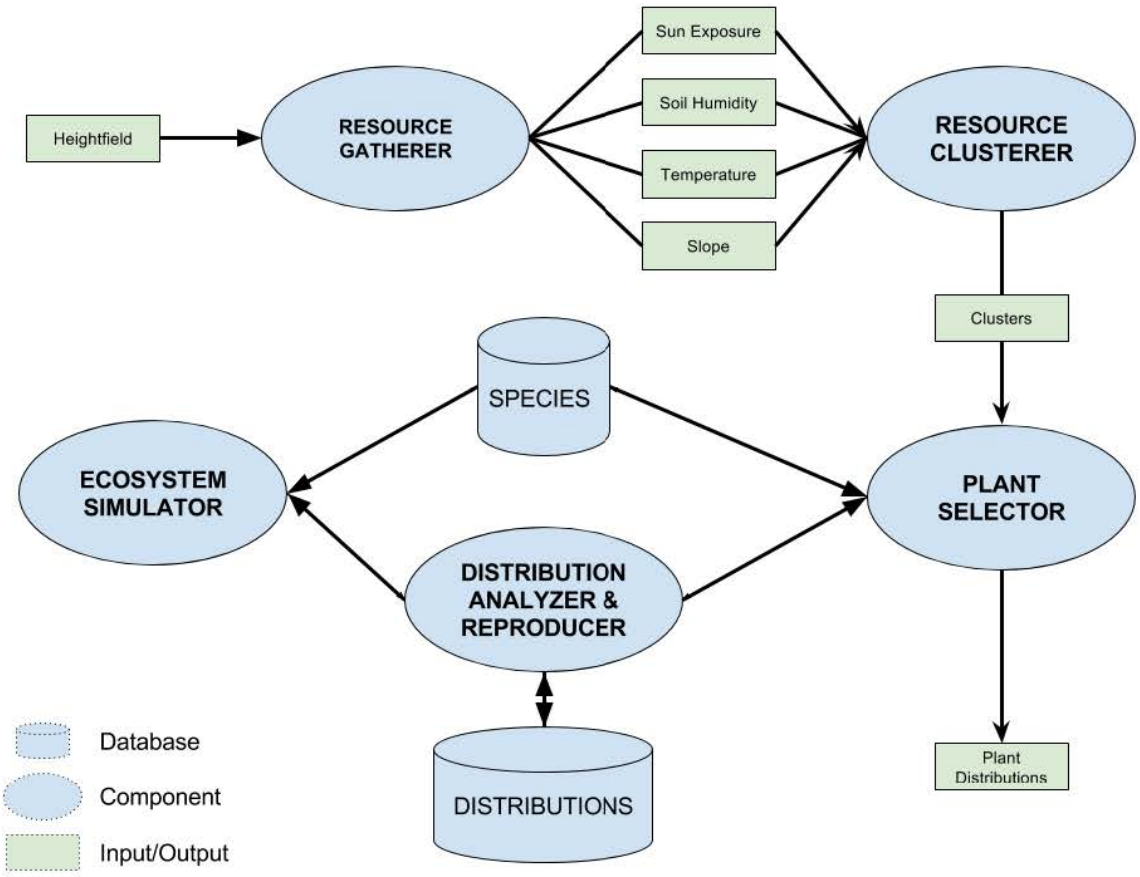

Figure 3.1: *System overview*

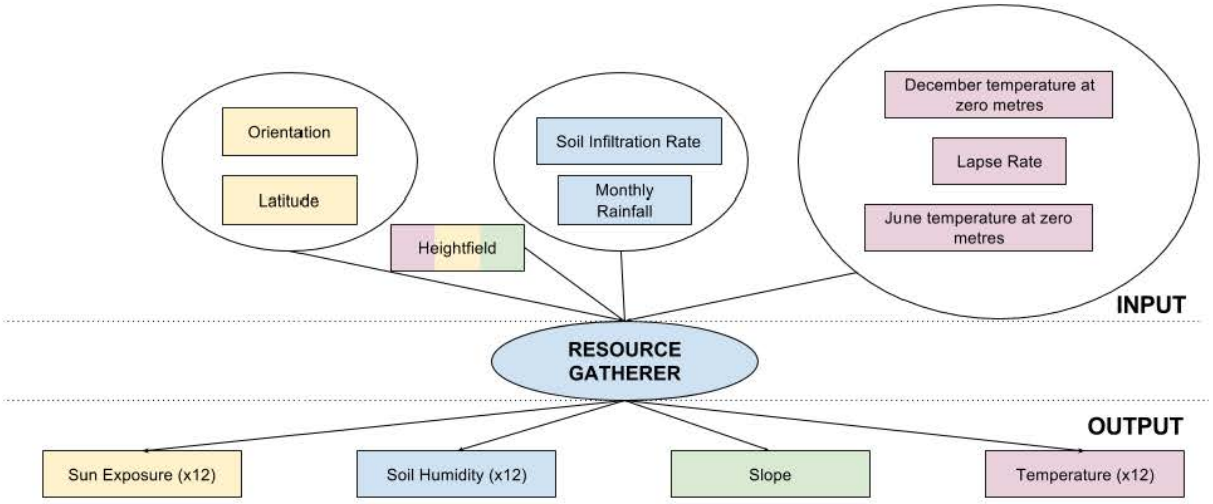

Figure 3.2: *Resource gatherer overview with colour coding to correlate input with corresponding output.*

quantity and monthly rainfall intensity must be configured.
To determine the *temperature* of each terrain vertex, the user must specify the temperature at zero metres in June and December along with the associated lapse rate. These temperatures are then considered the annual minimum and maximum and are used to deduce the temperature for any month through linear interpolation. The lapse rate represents the decrease in temperature with altitude and is used to determine the temperature for any terrain vertex given its altitude.
The *slope* is determined automatically from the input terrain.

3.2.2 Resource Clusterer

Determining a suitable plant distribution for each individual terrain vertex is infeasible. Indeed, this would require running an instance of the the computationally expensive ecosystem simulator (see Section 6.3.8) for each individual terrain vertex.
To reduce the amount of plant distributions to calculate, K-means clustering is performed on the terrain to group together points with similar resource properties. The mean value of each cluster is then used to determine suitable vegetation and its distribution. Figure 3.3 illustrates the input requirements and output of this component.
The *cluster count* which must be specified as input dictates how many clusters the algorithm produces. In essence, it controls the sensitivity of the clustering algorithm.
The per-vertex resource data represents all the resource information discussed in Section 3.2.1. Given all this information, the clustering algorithm gathers points which are most similar in terms of resources into a set of k clusters.

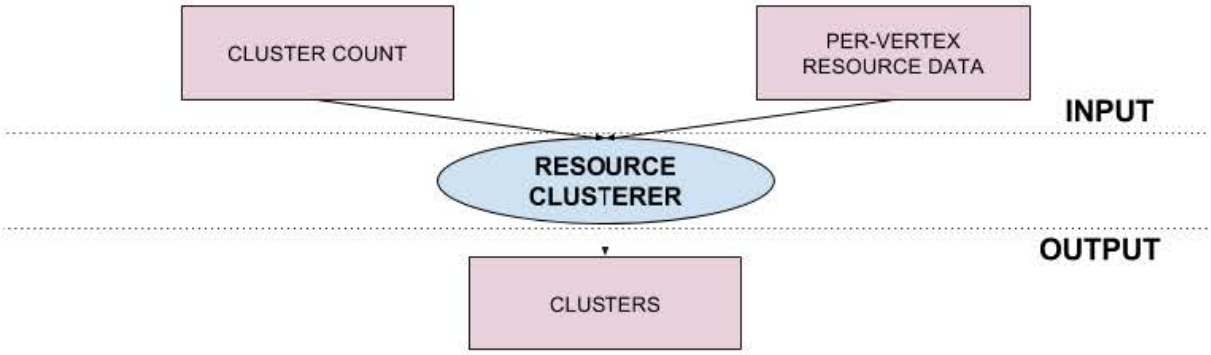

Figure 3.3: *Resource clusterer overview with required input and generated output.*

3.2.3 Plant Selector

Given the mean value of the individual clusters, the plant selector determines the plants which are able to survive in each terrain cluster and calculates for each of them a suitability score. This score represents how suited a species is to each individual cluster and is displayed to the user for informational purposes in order to facilitate the species selection procedure. Figure 3.4 shows the input requirements and outputs of this component.

3.2.4 Ecosystem Simulator

The ecosystem simulator is used to determine a valid plant distribution given a set of plant species and resources (soil humidity, illumination, slope and temperature). It simulates plants spawning, growing, battling and dying through time at monthly intervals on a hundred by hundred metre simulation area. Figure 3.5 shows the input and output requirements of this component.

3.2.5 Distribution Analyser and Reproducer

Because the ecosystem simulator is computationally expensive, the simulation area is restricted to ten thousand square metres (hundred by hundred metres). In order to place vegetation in clusters with larger surface areas, radial distribution analysis and reproduction is performed [EVC+15, BM07, LP02]. This technique analyses the variation in plant density over distance of an input exemplar in order to generate pair correlation histograms which are used to reproduce distributions matching the characteristics of the input exemplar. Because the reproduction is much less computationally costly than the ecosystem simulator, it is possible to efficiently produce distributions covering much larger areas. Figure 3.6 shows the input requirements and outputs of this component.

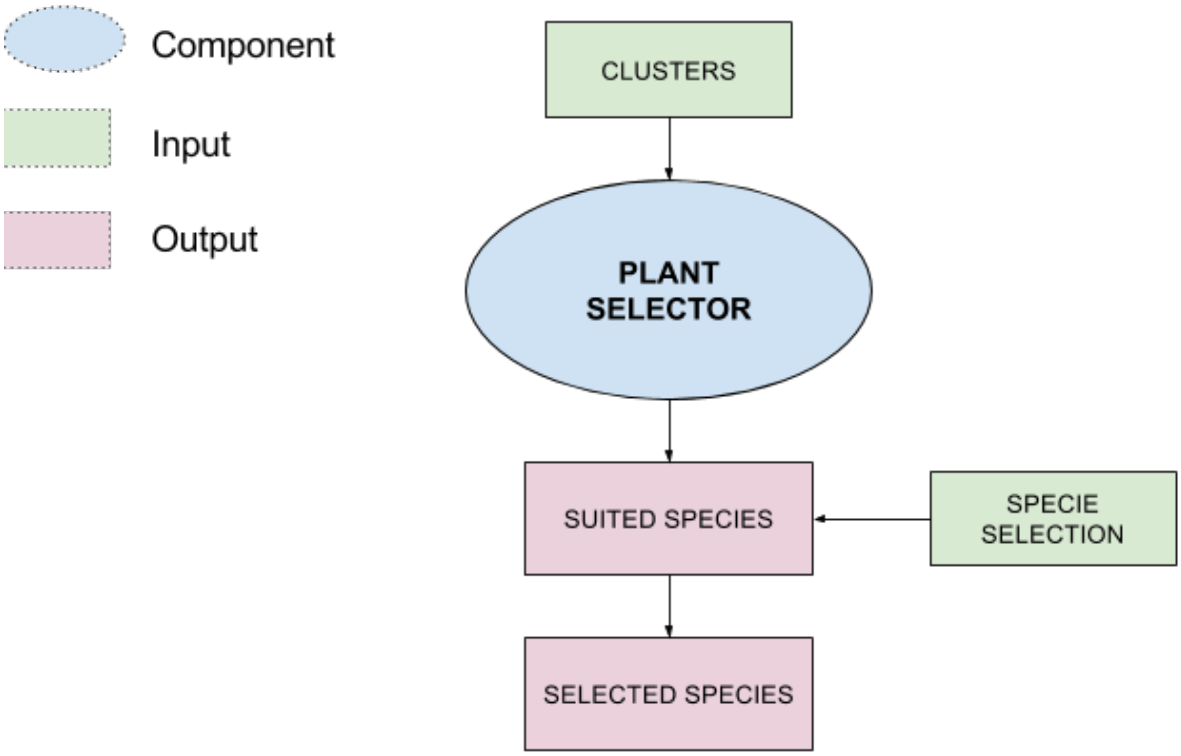

Figure 3.4: *Plant selector overview.*

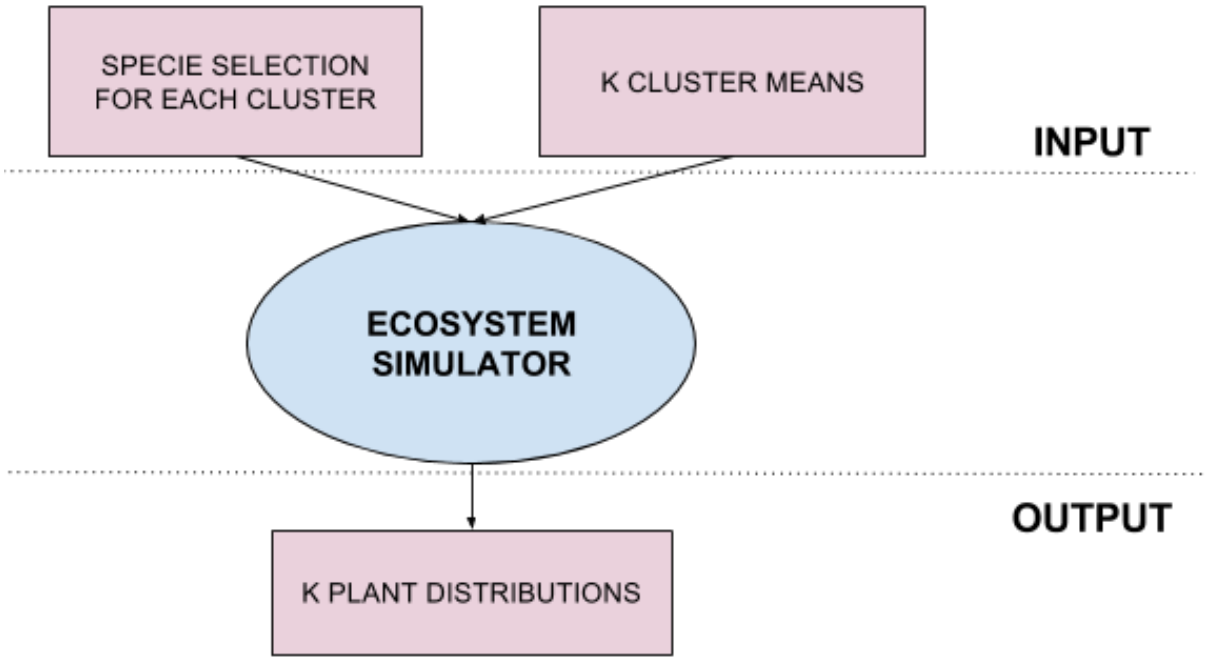

Figure 3.5: *Ecosystem simulator overview.*

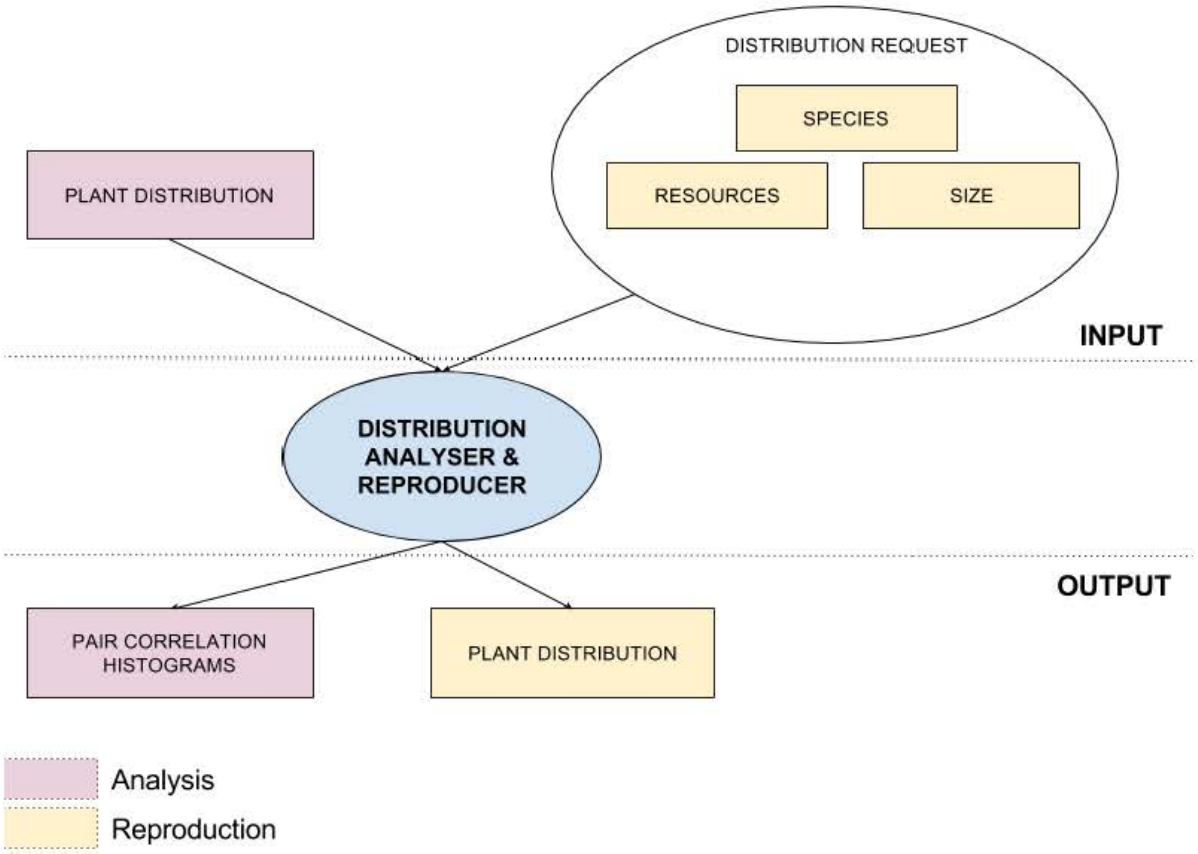

Figure 3.6: *Distribution analyser and reproducer overview.*

3.3 Limitations

The system places vegetation procedurally and does not permit users to place them manually. Unfortunately, this means that fine control over the final vegetation distribution is lost.

This work focuses on the generation of untouched rural terrains and does not permit the placement of man-made objects such as roads, cities, buildings and crops.

Because of limited scope, the system does not render realistic three dimensional models of the different plants to place on the terrain. The output of the system rather states the position of individual plant instances along with associated properties such as height, root size, canopy width and age.

Because of these limitations, it is more accurate to see this system as a tool to be used alongside a game engine such as the *Unreal Engine* [1]. This system could then be used to determine vegetation and water networks on the terrain and the game engine used to generate realistic renders of the scene.

3.4 Similarities to Existing Work

Explicit instancing techniques [EVC+15, DHL+98, ACV+14] permit users to explicitly state the exact location of individual plant instances. Although this gives users fine-control over terrain content, the task can be long and tedious for very large terrains. Realism can also be poor in these systems as they rely entirely on the user for accurate vegetation placement.
Ecosystem simulators techniques [LP02, DHL+98] attempt to overcome this by generating plausible plant distributions by algorithmically reproducing the competition for resources that occurs in nature during plant growth. In order to generate plant instances on areas large enough to fill entire terrains in a manageable time frame, however, these algorithms are often over-simplified.
Probabilistic placement techniques such as that employed in the work by Emilien et al. [EVC+15] attempt to overcome the downside of explicit placement by analysing inter and intra plant distributions in order to replicate similar distributions on much wider areas. This way the user is only required to perform explicit plant placement on a small area, the distribution of which can be analysed and reproduced on any scale with no repetition. Vegetation realism is still not guaranteed, however, as the user is still responsible of producing the input exemplars as well as delimiting areas on the terrain on which to map the exemplar.

[1] `www.unrealengine.com`

Chapter 4

Terrain and Resources

The first step in creating virtual worlds is specifying the base terrain on which features will be placed. Subsequently, terrain resources with direct influence on content are determined. In order to strike a good balance between resulting realism and user experience, procedural methods must be employed when suited.

This chapter discusses how a system fitting these requirements was built. The discussion is split into the following core sections: *Terrain & Navigation*, *Resources*, *Rivers & Streams*, *Water Reserves* and *Results*.

Terrain & Navigation discusses how the base terrain is selected and navigated through.

In order to determine suitable vegetation and river sources, resource data needs to be specified. How this is done is discussed in the *Resources* section.

Essential to the realism of virtual terrains is water placement. This water can take the form of rivers and streams or water reserves. Techniques used to place such content are discussed in the *Rivers and Streams* and *Water bodies* sections, respectively.

4.1 Terrain and Navigation

In order to give the user the freedom to model any type of virtual world, providing the ability to specify any type of base terrain is essential. Efficiently rendering and navigating this terrain is also key for both the user experience and visual realism. How our system manages these requirements are discussed sections *Loading Terrain*, *Rendering Terrain* and *Navigating Terrain* below.

4.1.1 Loading Terrain

As stated previously, our work focuses on terrain content and not terrain relief modelling. As such, the user is only able to load a static, pre-generated terrain in the form of a Terragen height-map. A height-map is a 2-dimensional grid of height values which, once loaded and converted, represents the height of the terrain on a regular grid. The Terragen file format is a freely available and widely used file-specification created by PlanetSide [1] for their realistic virtual world generation software, Terragen. The format wraps raw height data with other important information essential to accurate rendering such as base height, scales and dimensions.
Note that modelling the base terrain as static is a simplification as in reality it is affected by erosion. The extent of which depends on many factors including wind, vegetation and water.

4.1.2 Rendering Terrain

Once parsed, the height-map data is transferred to the GPU as a two dimensional texture for rendering. In order to better visualize the terrain relief, a BlinnPhong shading model is used when rendering the terrain. This shading model takes into consideration camera viewpoint and lighting incidence angles to determine the influence of diffuse and specular lighting on individual terrain vertices. This information is subsequently used to calculate a weighted contribution of ambient, specular and diffuse colors to determine the aggregate color of individual terrain vertices. By accurately modelling specular and diffuse highlights, renders are more realistic and shapes more distinguishable [Bli77]. By employing this model in this work, terrain relief is made clear.
Essential to the Blinn-Phong shading model are the normal vectors for each terrain vertex. This is done using the algorithm outlined in Equation 4.1 and illustrated in Figure 4.1. Each normal is calculated in parallel on the GPU, thus ensuring real-time results.

$$N_P = V_{ac} \times V_{db} \tag{4.1}$$

Where: N_P is the normal vector at point P and P_A, P_B, P_C and P_D are the direct points surrounding P in the X and Y direction (see Figure 4.1).

[1] `http://www.planetside.co.uk`

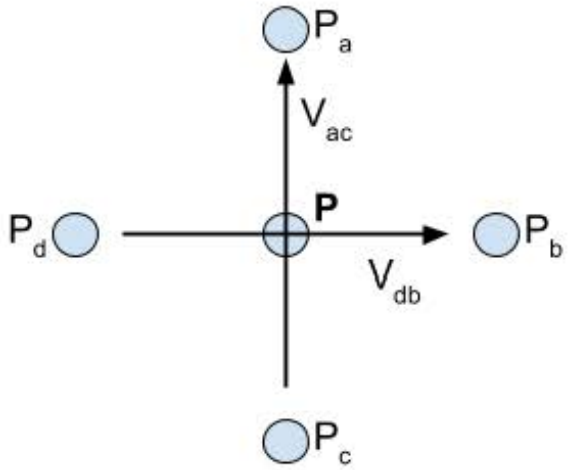

Figure 4.1: *Illustration of the vertices and vectors used to calculate terrain normal at position P.*

4.1.3 Navigation

In order for users to successfully and intuitively navigate through virtual worlds, it is important to prevent disorientation by ensuring continuous user awareness of location and orientation [DSS93]. In their work, Darken et al. [DSS93] explore various navigation techniques to do so, including the flying scenario where users explore virtual worlds as if they were flying through it. This navigation technique provides a birds eye view of the virtual worlds and enables users to gain an overview of the terrain and efficiently locate landmarks to serve as point of references. Locating such landmarks proves extremely useful in keeping the user aware of his location and therefore preventing disorientation [DSS93]. Birds eye has become the most widespread navigation technique employed in video games, simulators and virtual world generation software. In order to support a variety of users (novice to computer graphic experts), this is the navigation style used in our system. To further prevent disorientation, a compass is continuously displayed stating the current heading.

Intuitive controls and suitable sensitivity thereof are also essential. The correlation between key-press and mouse movement must be predictable so that the user can navigate in three dimensional space without losing his bearings. In an attempt to cater for the control requirements of a wider user-base, two different control types are available in this system: *keyboard-driven* and *click-and-drag*. Details of which can be found in Table 4.1. The active control type is easily configurable, along with sensitivity parameters, through the application's configuration interface.

Control-type	**Translate Left/Right**	**Translate Up/Down**	**Translate Front/Back**	**Rotate Left/Right**	**Rotate Up/Down**
Keyboard driven	A/D key-press	-	W/S key-press	Horizontal mouse move-ment	Vertical mouse move-ment
Click-and-drag	Horizontal click & drag	Vertical click & drag	Scroll wheel	Ctrl + hori-zontal click & drag	Ctrl + ver-tical click & drag

Table 4.1: *Control types instruction sheet*

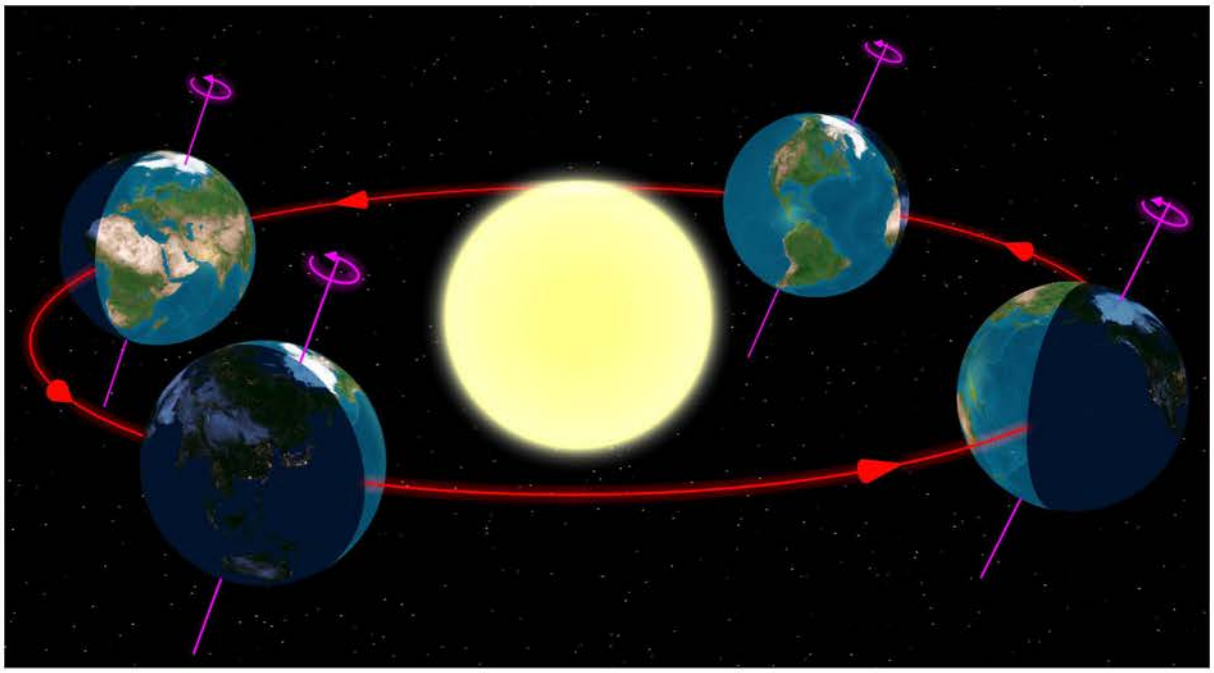

Figure 4.2: *Annual orbit of the earth around the sun. Source:* `http://en.wikipedia.org/wiki/Summer_solstice`

4.2 Resources

A core feature of rural landscapes is vegetation and accurately replicating it is therefore critical to the resulting realism of a modelled virtual world. Accurate plant growth modelling is an active area of research as it proves to be an important tool for crop yield optimization [FZS+08]. In their work, Fourcaud et al. [FZS+08] note the importance of modelling the interaction of plants with environmental light, temperature, soil nutrients and water to accurately simulate growth. In order to determine a plausible vegetation layer in our system, *Illumination*, *temperature*, *precipitation*, *soil humidity* and *slope* are modelled. Note that although these resources are deemed essential for accurate plant growth modelling [FZS+08], it is a simplification as other influential factors such as air quality and air pressure are discarded.

4.2.1 Illumination

The earth rotates around the sun with an axial tilt, also known as obliquity, of approximately 23.5 degrees (see Figure 4.2). Because of this obliquity, given a position X at latitude L, the amount of illumination received at X in a a 24-hour period will vary during the course of the year (see Figure 4.3).

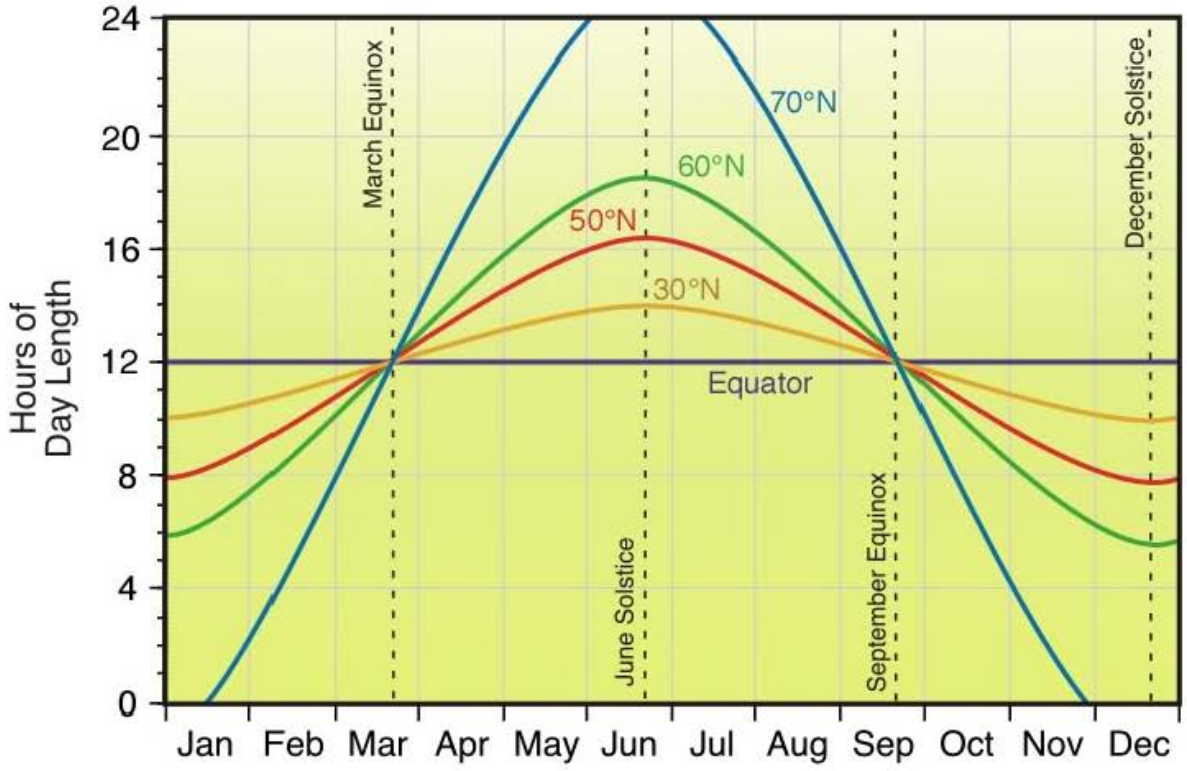

Figure 4.3: *Variation in day length for different latitudes. Source:* `http://www.physicalgeography.net/fundamentals/6i.html`

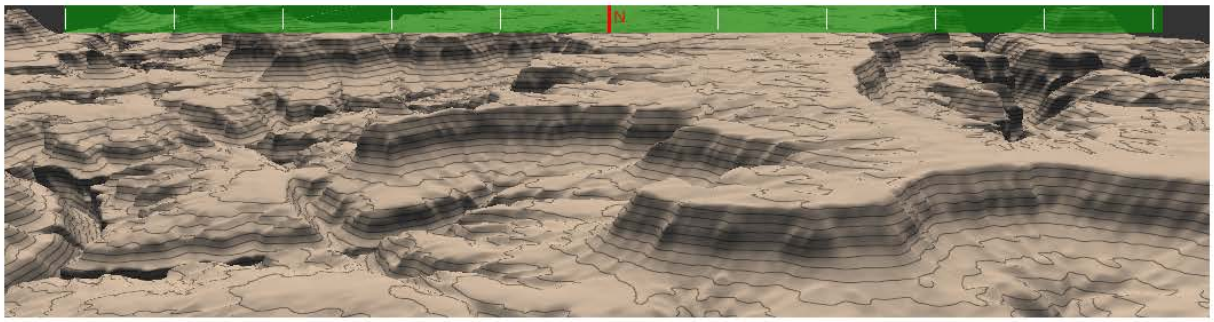

Figure 4.4: *Orientation controller compass (top of render window). The compass is displayed green to provide feedback to the user that orientation edit mode is active.*

In order for the terrain to remain static, when calculating the sun's trajectory the frame of reference is changed to be the earth, around which the sun orbits. To calculate the sun's position at any given time, there are four vital pieces of information that need to be specified by the user: *Latitude*, *Orientation*, *Time of day* and *Month of year*.

Specifying the *latitude*, *time of day* and *month of year* is done using sliders which overlay the rendering window. By keeping the render window active during this edit, modifications are clear to the user. When any of these values are changed, the position of the sun is automatically recalculated in real-time.

Orientation is displayed to the user at all times with the use of an overlay compass (Figure 4.4) inspired by first-person video games. When in orientation edit mode the compass changes to green to provide feedback that the user edit mode is active, at which point the orientation can be modified by using the right/left keyboard keys. Again, all modifications update the sun position in real-time.

Given all this information, the first step is to calculate the rotation axis V_{RE} of the sun at the equinox. This is done using Equation 4.2. Taking V_{RE} as the rotation axis for the sun is a simplification. However, the distance between earth's center axis and V_{RE} is negligible in comparison to the distance between the earth and the sun and is therefore deemed an acceptable simplification.

$$V_{RE} = R(V_N, \text{-}L, V_E) \tag{4.2}$$

where: V_{RE} is the rotation axis of the sun at the equinox; V_N is the north-facing vector passing through the terrain center; V_E is the east-facing vector passing through the terrain center; L is the latitude of the terrain; $R(V_a, a, V_b)$ is the resulting vector after rotating V_a by a degrees

around V_b

V_{RE} is the rotation axis for the sun at the March and December equinox. During the equinox, axis tilt has no effect on daytime duration as the tilt is not directed away or towards the sun. At this point, latitude alone is the determinant of daytime duration. In order to calculate the rotation axis V_R(m) of the sun at month m, axis tilt must be taken into consideration by further rotating V_{RE} using Equation 4.3.

$$V_R(m) = R(V_{RE}, a_m, V_E) \tag{4.3}$$

where: $V_R(m)$ is the rotation axis of the sun at month m; V_{RE} is the rotation axis of the sun at the equinoxes (Equation 4.2); V_E is the east-facing vector passing through the terrain center; a_m is the rotation angle calculated using Equation 4.4; $R(V_a, a, V_b)$ is the resulting vector after rotating V_a by a degrees around V_b

$$a_m = -tilt_{max} + |6 - m| \times tilt_{monthly}$$

$$tilt_{monthly} = tilt_{max}/3 \tag{4.4}$$

where: a_m is the rotation angle at month m; $tilt_{max}$ is the maximum axis tilt of the earth (23.5 degrees)

The time of day, t is then used to determine the amount the sun is rotated around the rotation axis $V_R(m)$. With a full rotation being performed every 24 hours.

4.2.1.1 Calculating Illumination

A point on the terrain is illuminated if there is a direct path from it to the sun with no intersections with other points on the terrain. To test for this on the terrain ray casting is performed from each vertex position towards the sun to check whether or not it intersects with other points on the terrain.

This process can be lengthy, however, as a ray casting operations needs to be performed for each individual terrain vertex. In order to accelerate this process, a spherical hierarchical acceleration structure is used. This hierarchical acceleration structure employs a tree structure to iteratively search for smaller intersection areas. Using this acceleration structure, illumination can be calculated for 4 million vertices in just over 2 seconds (Figure 4.5). As shown in Figure 4.5, there is a linear relationship between vertex count and calculation time.

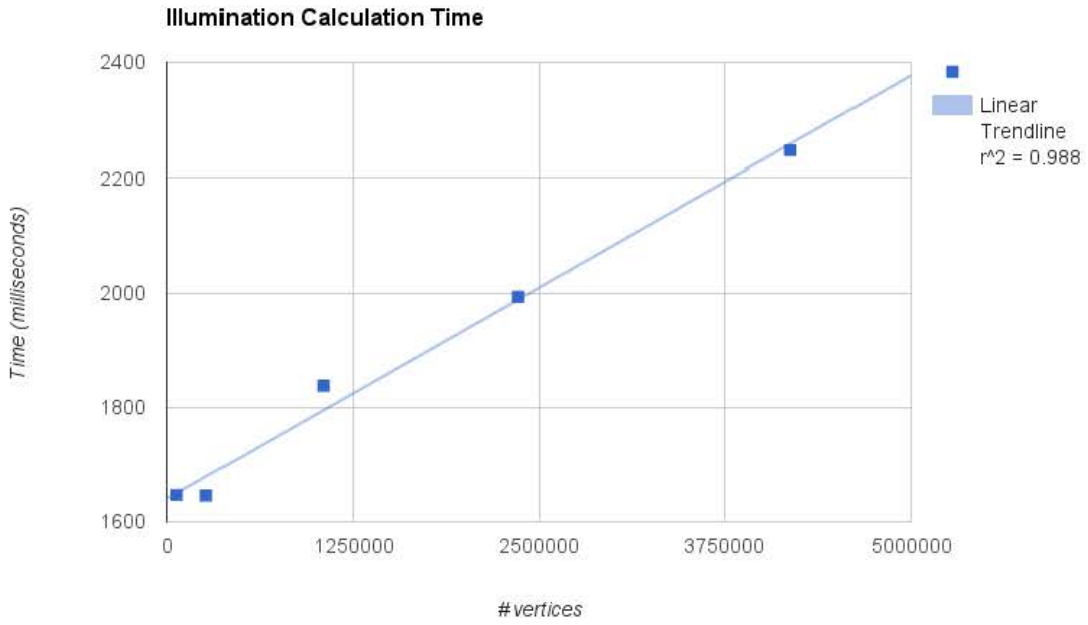

Figure 4.5: *Illumination calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.*

An important element which will influence vegetation on the terrain is the variation in the hours of illumination received daily during the course of the year. To determine the illumination received on a given day, the illumination calculation is used by iterating through each hour of the day consecutively and determining whether or not a vertex V is illuminated. In order to reduce the number of illumination calculations to perform when determining the variation of the illumination throughout the year, the illumination is calculated for the fifteenth day of every month rather than for every day of the year.

To illustrate the daily illumination on the terrain to the user for a given month m, an *illumination overlay* can be enabled which darkens and lights up terrain vertices proportionally to the amount of light received (see Figure 4.6).

4.2.2 Temperature

Whereas tropical climates often have relatively constant temperatures throughout the year, others, such as the continental climate, are characterized by a strong variation between minimum and maximum annual temperatures. Only plants which are able to survive at both extremes can grow, which is why temperature and its variation has a significant impact on vegetation. For modelling purposes, it is acceptable to assume that the minimum temperature, T_{min}, occurs in the middle of winter and the maximum temperature, T_{max}, occurs in the middle of summer. In-

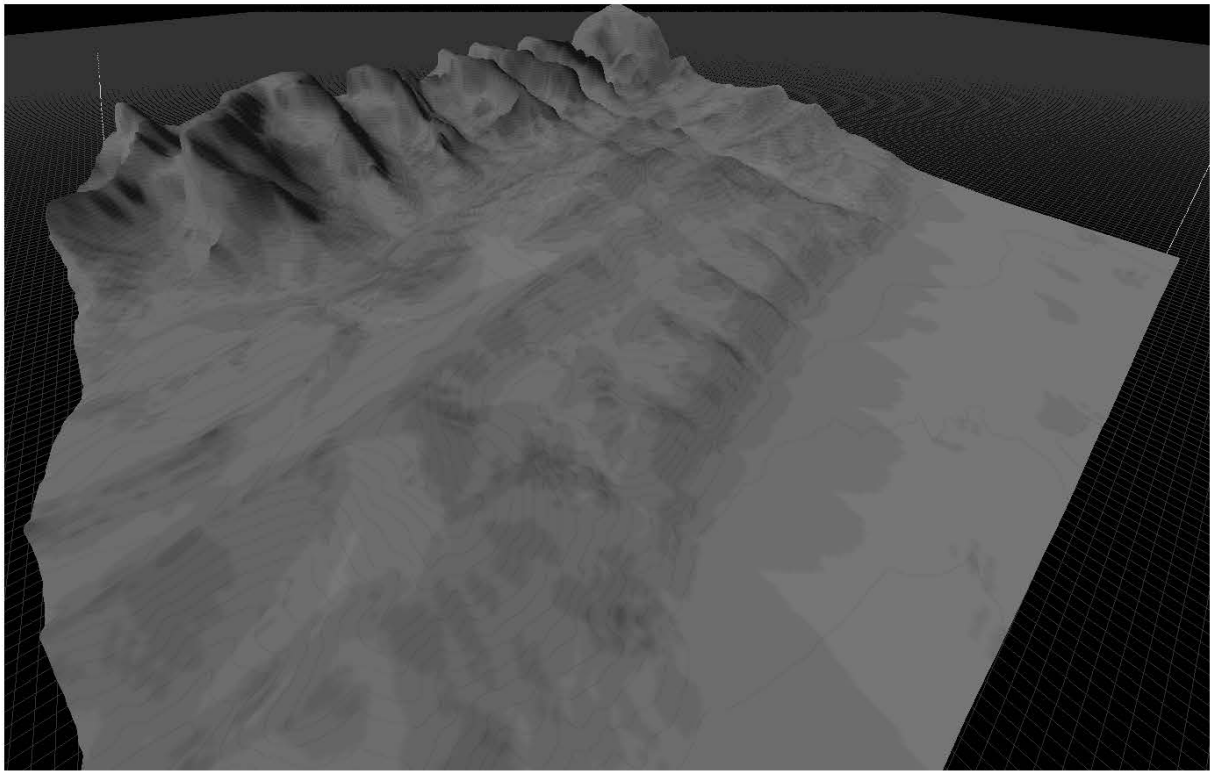

Figure 4.6: *Daily illumination overlay. The brighter the area, the more illumination it receives throughout the day.*

terpolation can be performed to determine the temperature at any time between these two dates.

Properties which are necessary to model temperature in the system include: *Altitude*, $Temp_{december}$, $Temp_{june}$ and *Lapse rate*. The *altitude* of each point on the terrain is calculated automatically based on the properties of height-map loaded. $\text{Temp}_{december}$ and Temp_{june} represent both extremes of the temperature spectrum at zero meters of altitude and need to be configured by the user. The *lapse rate* defines the decrease in temperature with altitude. Although this changes depending on atmospheric conditions, the default is configured to a value of 6.4 degrees Celsius for each km increase in altitude. This is accepted as the average atmospheric lapse rate under normal atmospheric conditions [2].

Given this information, the temperature is calculated for any point on the terrain given the month and altitude using Equation 4.5.

$$T(a,m) = T_{december} + (\frac{6 - |6 - m|}{6} \times (T_{june} - T_{december})) \tag{4.5}$$

where: $T(a,m)$ is the temperature at altitude a and month m; $T_{december}$ is the temperature at zero meters in December; T_{june} is the temperature at zero meters in June.

Calculating the temperature for 4 million vertices (2048 by 2048 terrain) takes approximately 2 seconds. Figure 4.7 points towards a linear relationship between vertex count and temperature calculation time.

An overlay can be enabled to provide a graphical overview of the temperature at different locations on the terrain for the selected month (see Figure 4.8).

4.2.3 Precipitation

Precipitation is a core part of climate classification and, consequentially, plant life. Arid climates have very limited annual precipitation and this is a bottleneck for organic life. Tropical climates, on the other hand, where precipitation is plentiful, have an abundance of vegetation. There are two important properties of precipitation that are modelled: *Quantity* and *Intensity*. The *quantity*, often measured in mm, defines the amount rain that falls. The intensity, often measured in mm/h defines the rate at which it falls.

The user must configure *quantity* and *intensity* values for each month of the year. A custom input dialogue was implemented in an attempt to make this as user-friendly as possible (4.9).

[2] `http://en.wikipedia.org/wiki/Lapse_rate`

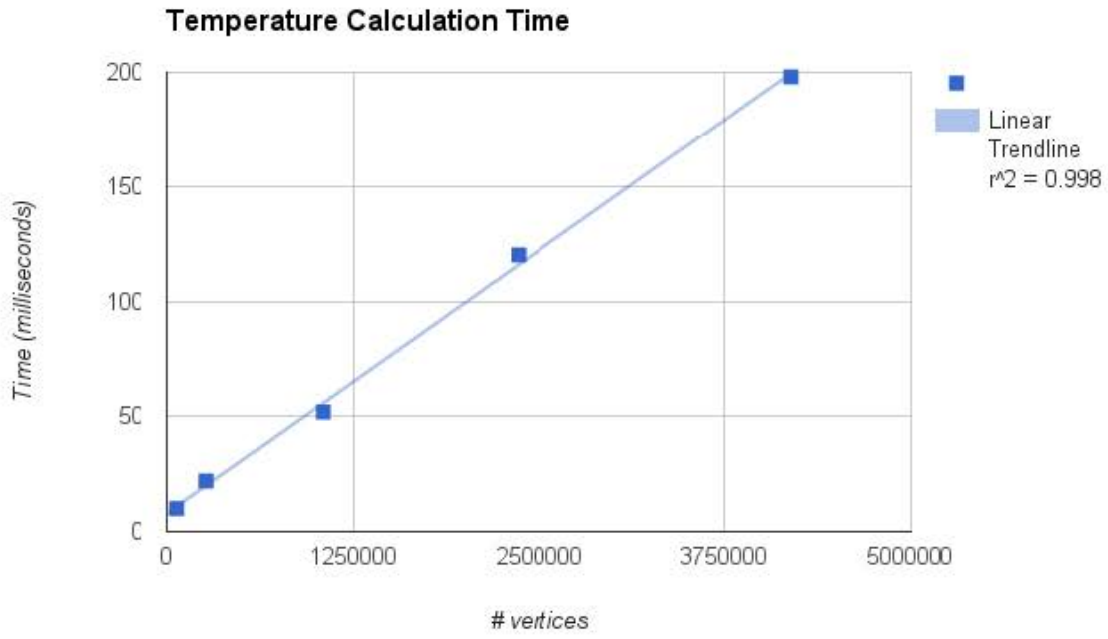

Figure 4.7: *Temperature calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.*

4.2.4 Soil Humidity

When rain falls onto the terrain, a certain portion of it is absorbed into the soil to provide the plant's roots with the necessary nutrients. The portion which is absorbed depends on the type of soil. Rocky soils, for example, have limited water retention and will result in larger water build-up and potentially run-off. In this work, the soil humidity is a measure, in mm, of the rainfall which is absorbed by the soil for each given month. This is determined using the precipitation information outlined above (4.2.3) along with the *Soil Infiltration Rate*.

Soil humidity, also referred to as soil moisture, is most commonly measured as the volumetric water content in the soil, as a percentage [SJM80]. Calculating the volumetric ratio of water to soil would require soil depth data for each terrain vertex which, due to scope, is not represented in our system. Millimetres of rainfall was deemed an adequate measure, however, as the water requirements of different plant species are often stated in millimetres of rainfall, therefore providing a good correlation between available and required resource.

4.2.4.1 Soil Infiltration Rate

The *soil infiltration rate* is a measure of the quantity of water which can be absorbed in a given period. If the rainfall intensity exceeds the soil infiltration rate, it will result in water stagnation

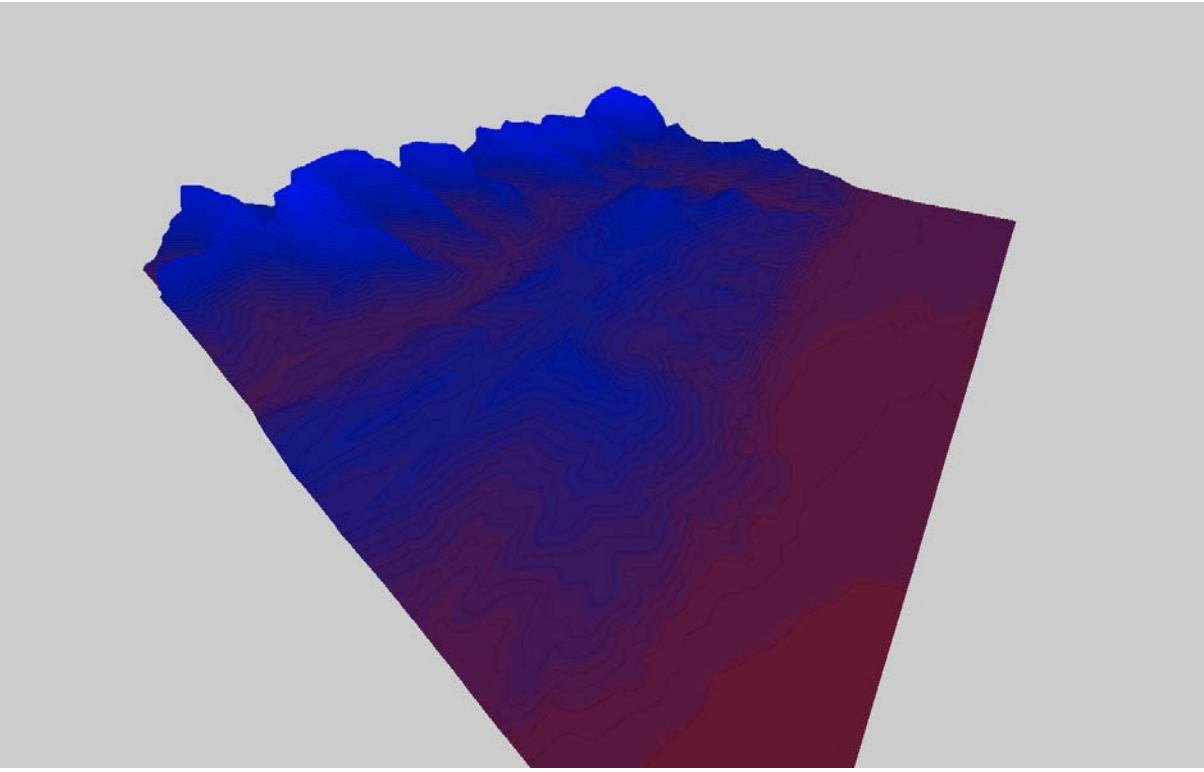

Figure 4.8: *Temperature overlay. Colour spectrum ranging from blue (cold) to red (hot). As can be seen, the temperature drops with altitude.*

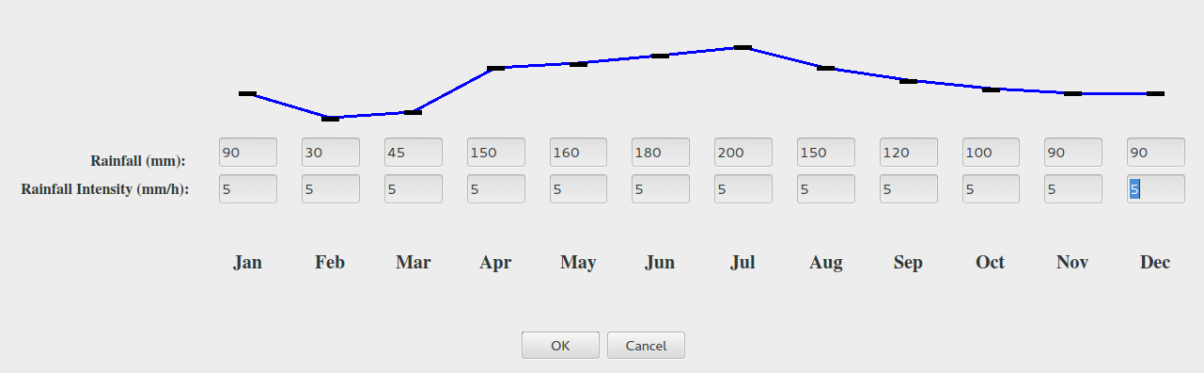

Figure 4.9: *Specifying monthly precipitation and precipitation intensity. The user can enter values manually (using the input fields) or interact directly with the graph.*

(on a flat surface) or water run-off. This rate is correlated with the type of soil and approximate values for some soil types can be found in Table 4.2.

Soil-type	**Infiltration rate (mm/hour)**
sand	< 30
sandy loam	20-30
loam	10-20
clay loam	5-10
clay	1-5

Table 4.2: *Soil infiltration rates for different soil types* [3]

Specifying the soil infiltration rate can be done using three distinct tools: *Filling*, *Slope-based* and through a *Painting interface*.
Using the *filling* tool, the user can fill the entire terrain with a configured infiltration rate. The *slope-based* tool permits the user to configure a slope above which the soil infiltration rate will be zero. This is an efficient tool for modelling cliff faces or simply areas where water run-off is too severe. The *painting interface* enables user to paint a soil infiltration on the terrain directly using a common brush tool found in basic paint applications. The size of the brush can be configured using the scroll-wheel and the painted soil-infiltration using a dedicated slider controller (4.10).

These tools can be used interchangeably and users are encouraged to do so. Configuring the soil infiltration of the terrain in Figure 4.10, for example, was performed in approximately three minutes using all three tools as follows:

1. The filling tool was used to fill the entire terrain with an infiltration rate of 25.

2. The slope-based tool was used to set to zero the infiltration rate of all areas with a slope above 30 degrees.

3. The painting-tool was used to paint an infiltration rate of zero on the flat area to the right to cater for a water build-up (reservoir, lake, ocean, etc.).

Given the *soil infiltration rate*, *monthly precipitation* and *monthly average precipitation intensity* for each vertex on the terrain, it is possible to calculate the quantity of rainfall that is actually absorbed by the soil for each month. This represents the soil humidity in our system and is calculating using Equation 4.6.

[3] `http://www.fao.org/docrep/s8684e/s8684e0a.htm`

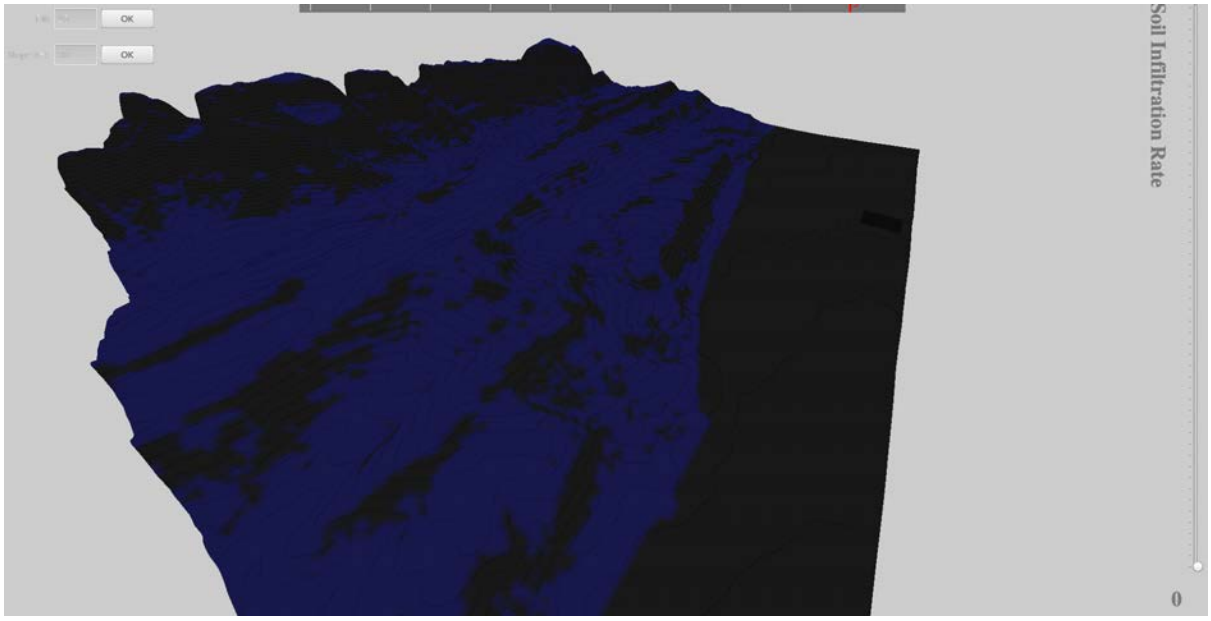

Figure 4.10: *Editing the soil infiltration rate on the terrain. Top left are the controllers for the filling and slope-based tools. On the right is the slider to configure the soil infiltration rate of the paint brush.*

$$S_h(R_q, R_i, S_{ir}) = \frac{S_{ir}}{R_i} \times R_q \tag{4.6}$$

where:S_h is the soil humidity for a given month, in mm; R_q is the monthly rainfall quantity, in mm; R_i is the average monthly rainfall intensity, in millimetres per hour;S_{ir} is the soil infiltration rate, in millimetres per hour.

Intuitively, Equation 4.6 calculates the proportion of the total rainfall which is able to be absorbed by the soil given the rainfall intensity and absorption rate of the soil.

To accelerate the process, the soil humidity is calculated in parallel for each vertex on the GPU. By doing so, 4 million vertices (2048 by 2048 terrain) can be processed in 104 milliseconds (Figure 4.11). There is a linear relationship between vertex count and calculation time (Figure 4.11).

4.2.4.2 Weighted Monthly Soil Humidity Calculation

Monthly soil humidity does not take into account the humidity of previous months and therefore fails to model water retention in the soil. By retaining water in the soil, the drought caused by an arid month can be counteracted to some extent by precipitation in previous months. To model this, a moving weighted average soil humidity is calculated for each month using Equation 4.7.

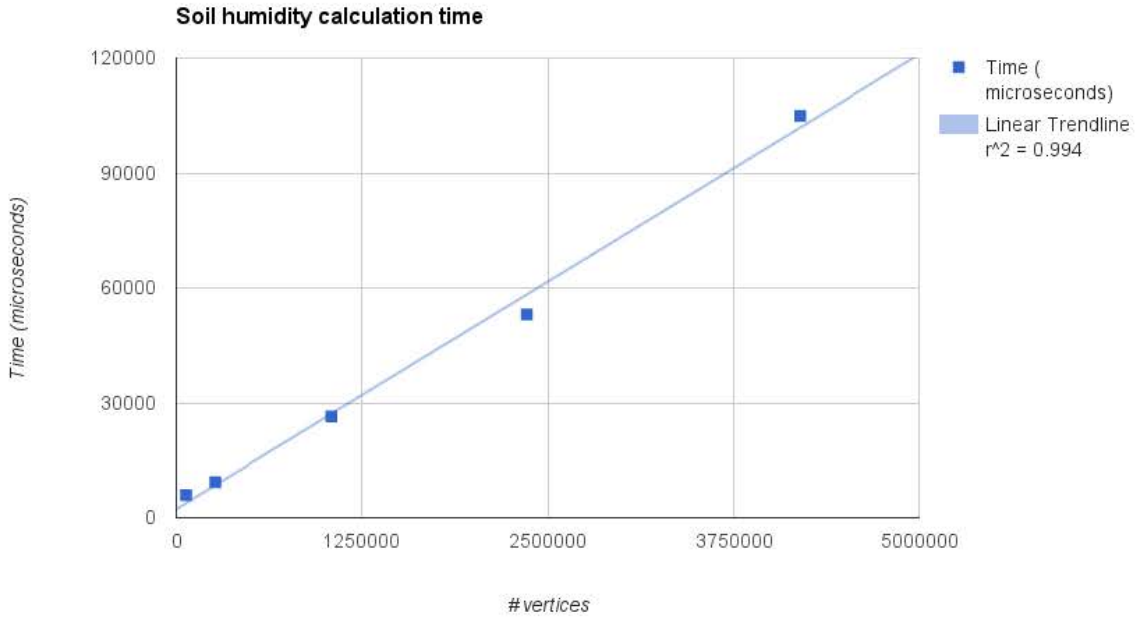

Figure 4.11: *Soil humidity calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.*

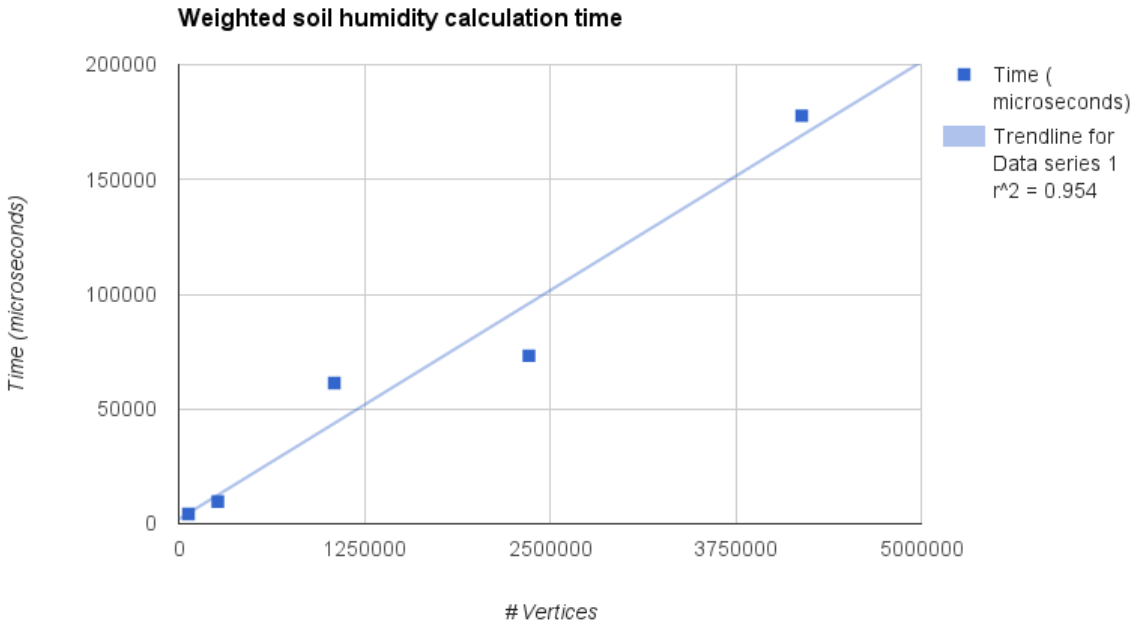

Figure 4.12: *Weighted soil humidity calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.*

$$S_{wh}(m) = (\frac{1}{2} \times S_h(m)) + (\frac{1}{3} \times S_h(m-1)) + (\frac{1}{6} \times S_h(m-2)) \tag{4.7}$$

where: $S_{wh}(m)$ is the weighted soil humidity at month m, in mm; $S_h(m)$ is the soil humidity at month m, in mm.

Similarly to the monthly humidity, the GPU is used to accelerate the calculation process. By doing so, 4 million vertices (2048 by 2048 terrain) can be processed in 178 milliseconds (Figure 4.12). There is a linear relationship between vertex count and calculation time (Figure 4.12).

A visual overlay can be displayed to give an overview of both the monthly soil humidity and the weighted average monthly soil humidity on the terrain (Figure 4.13).

4.2.5 Slope

Slopes cause soil to be lost due to the effects of gravity. This loss in soil and therefore soil nutrients cause a bottleneck for plant growth [KD01] . This is clearly visible in nature, where only smaller plants (shrub, grass, etc.) grow on steeper landscapes. To determine suitable

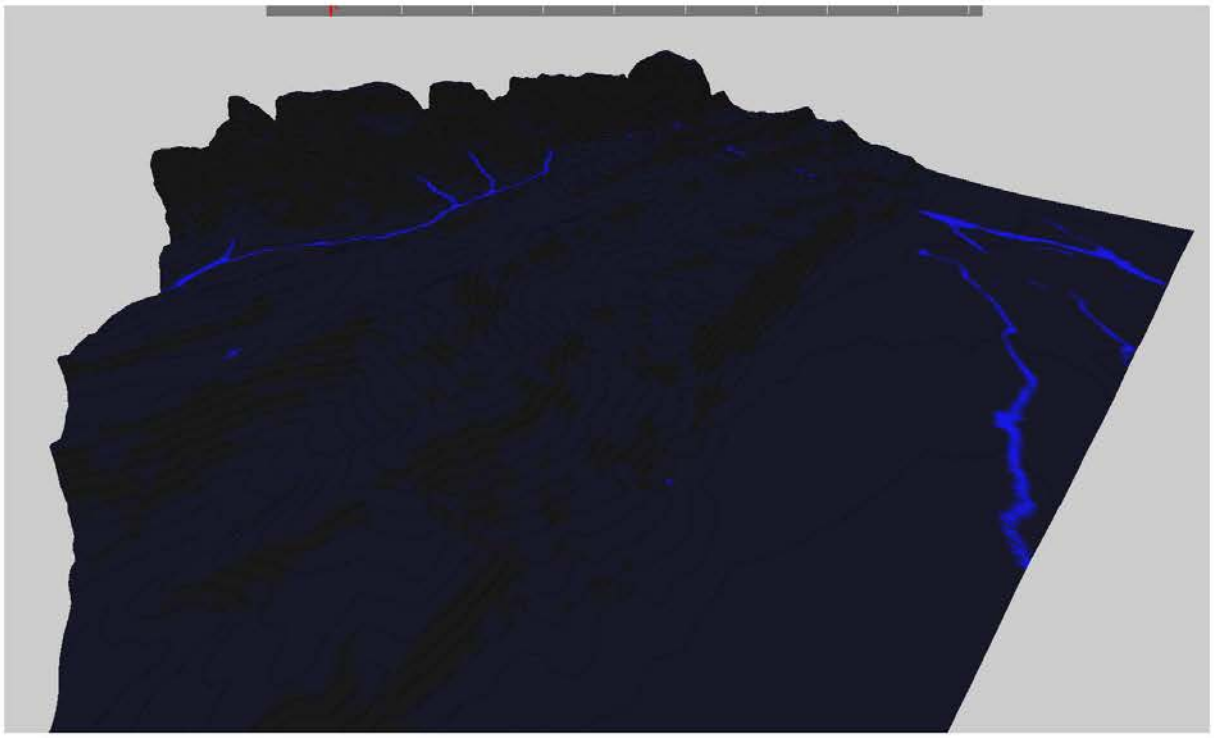

Figure 4.13: *Soil humidity terrain overlay. Colour spectrum ranging from black (zero humidity) to blue (standing water).*

vegetation given the terrain relief, it is therefore important to model slope.

This is calculated in real-time using the GPU (104 milliseconds for 4 million vertices) and the relation between vertex count and calculation time is linear (4.14)

To better visualize the slope throughout the terrain, a *slope overlay* can be enabled (see Figure 4.15).

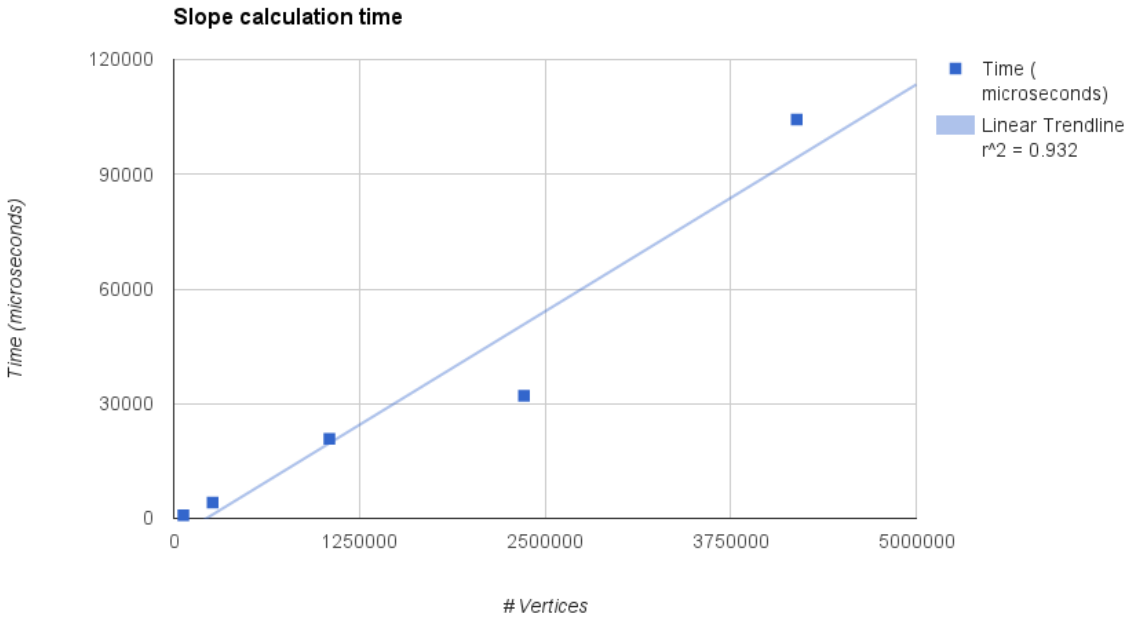

Figure 4.14: *Slope calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.*

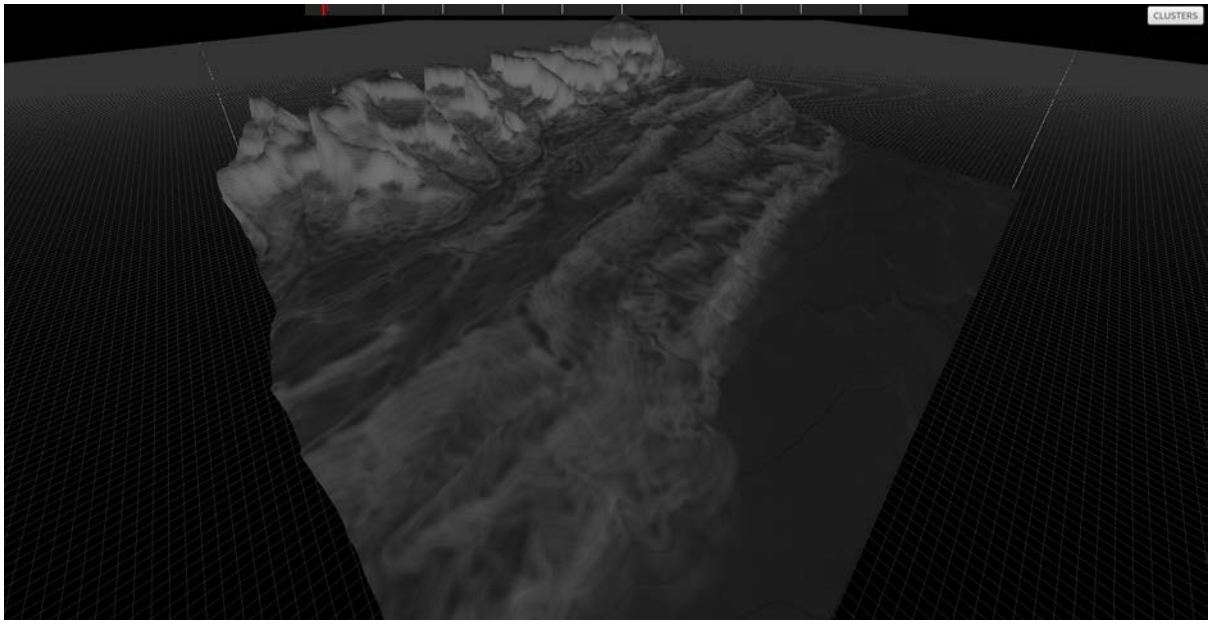

Figure 4.15: *Slope visual overlay. Colour spectrum ranging from black (zero degree slope) to white (90 degree slope).*

4.3 Rivers and Streams

Water networks are essential to the realism of virtual rural terrains. These water networks are constituted of rivers and streams and are the consequence of water being transported by gravity from higher to lower elevation. The algorithm used to model water movement on the terrain is outlined below, along with details concerning the GPU implementation.
Unlike work by [KMN88] and t'Ava et al. [ŠBBK08], this work does not simulate hydraulic erosion. That is, there is no feedback loop between water movement and the terrain to model changes in terrain relief.

4.3.1 Algorithm Overview

Precipitation, precipitation intensity and soil infiltration are used to calculate the soil humidity, S_h (see Equation 4.6). This equates to the quantity of water, in millimetres, absorbed by the soil. The standing water, $W_{standing}$, is the remaining water that is not absorbed by the soil and can be calculated using Equation 4.8.

$$W_{standing} = R_q - S_h \tag{4.8}$$

where: $W_{standing}$ is the standing water, in millimetres; R_q is the monthly rainfall quantity, in millimetres; S_h is the quantity of water absorbed by the soil, in millimetres.

Given the quantity of standing water, $W_{standing}$, for each vertex, a hydrostatic pipe-model similar to that of Stava et al. [ŠBBK08] is used to determine water movement and build-up on the terrain. This works by iteratively evacuating water from each terrain source vertex V to it's eight directly surrounding vertices when possible. The amount of water evacuated to individual surrounding vertices depends on slope and existing water content. During the water evacuation process, Stava et al. [ŠBBK08] also model the effects of force-based and dissolution-based erosion by modifying the terrain relief depending on calculated forces. This is not simulated in our work however, and the terrain relief remains fixed throughout the simulation. For this reason, our system works best with terrains with pre-existing erosion lines (e.g obtained from real world cartographic data).

Although the algorithm is implemented to work in three dimensions where each vertex can evacuate water content to eight surrounding vertices, it also works in a two-dimensional space. With this reduced dimensionality, each vertex V_n has only two neighbouring vertices (V_{n-1} and V_{n+1}) in which water can be placed. To better grasp the algorithm, it is described for a two-dimensional space. The concepts are identical when generalised to three dimensions.

Each vertex V_n, is characterised by its position (n), terrain height (TH_n), water height (WH_n) and absolute height ($AH_n = TH_n + WH_n$). Using this data it is possible to calculate the water evacuation capacity, WEC_n, of each terrain vertex (see Equation 4.9). The value of which effects how much water is evacuated and how it is split amongst surrounding vertices (Section 4.3.2).

$$WEC_n = 2 \times TH_n - AH_{n-1} - AH_{n+1} \tag{4.9}$$

Where: WEC_n is the water evacuation capacity of vertex V_n; TH_n is the terrain height at vertex V_n, WH_n is the water height at vertex V_n; AH_n is the absolute height at vertex V_n.

Intuitively, this equation calculates the maximum water quantity which can be evacuated to neighbouring vertices of source vertex V_n whilst ensuring that once the water is added, the aggregate height of these neighbouring vertices does not surpass the terrain height of vertex V_n.

4.3.2 Water Evacuation Approaches

Using the water evacuation capacity WEC along with Table 4.3, one of three evacuation approaches are used: *all water is evacuated*, *a portion of the water is evacuated* or *no water is evacuated*. Examples scenarios for each approach are illustrated in Figure 4.16.

	WEC >= WaterHeight_n	0 < WEC < WaterHeight_n	WEC < 0
All water can be evacuated	X	-	-
A portion of water can be evacuated	-	X	-
No water can be evacuated	-	-	X

Table 4.3: *Evacuation approach based on water evacuation capacity (WEC)*

When all water can be evacuated, it is split to surrounding vertices proportionally to their height. This is to model water flowing more intensely on steeper slopes. The quantity of water W_{n-1} and W_{n+1} to be evacuated to vertices V_{n-1} and V_{n+1} is calculated using equations 4.10 and 4.11 .

$$W_{n-1} = \frac{TerrainHeight_n - AbsoluteHeight_{n-1}}{WEC} \times WaterHeight_n \tag{4.10}$$

$$W_{n+1} = \frac{TerrainHeight_n - AbsoluteHeight_{n+1}}{WEC} \times WaterHeight_n \tag{4.11}$$

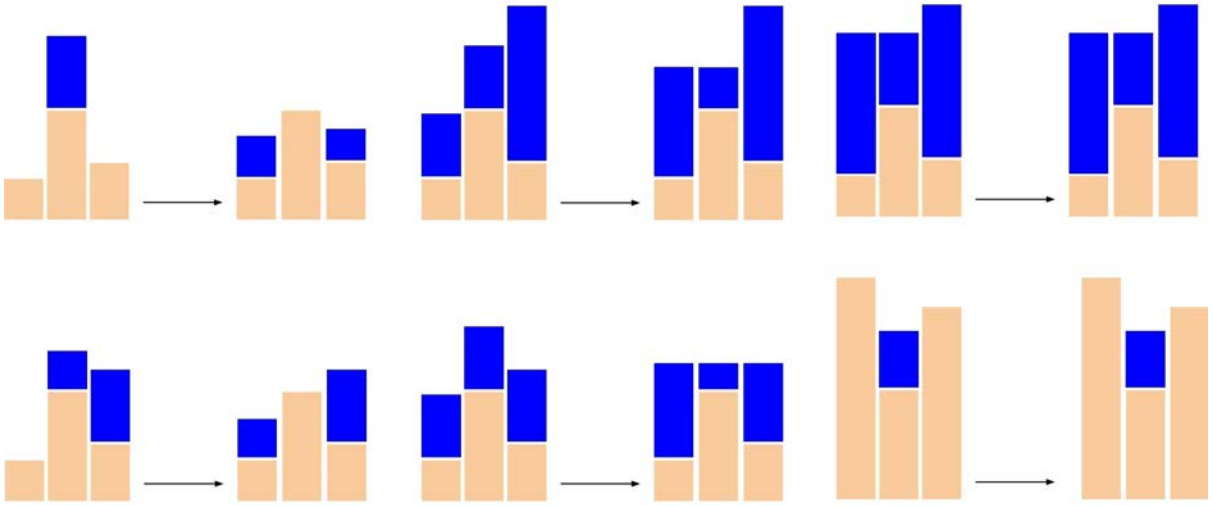

Figure 4.16: *Example water-evacuation scenarios. Left: All water can be evacuated from source vertex (middle). Middle: A portion of water can be evacuated from source vertex (middle). Right: No water can be evacuated from source vertex (middle).*

When the water evacuation capacity, *WEC*, does not permit all water to be purged but is sufficient to purge a subset, the amount of water to be evacuated, $W_{evacuate}$ need to be calculated using Equation 4.12.

$$W_{evacuate} = AbsoluteHeight_n - W_{level} \tag{4.12}$$

Where: W_{level} is the water level to which water can be evacuated (see Equation 4.13).

$$W_{level} = \frac{AbsoluteHeight_{n-1} + AbsoluteHeight_n + AbsoluteHeight_{n+1}}{3} \tag{4.13}$$

When calculated value of *WEC* is less than or equal to zero, no water can be evacuated and therefore no water is purged to neighbouring vertices.

4.3.3 Terrain Extremities

When attempting to evacuate water from vertices on the terrain extremities, one or more vertices will be absent. One way to deal with this would be to discard those vertices and permit water to flow only to existing surrounding vertices. By employing this approach, however, water would never leave the terrain and could build-up unrealistically. To overcome this, a one vertex thick border is generated around the terrain as illustrated in Figure 4.17. The height of each border vertex is calculated such that the slope matches that of it's neighbouring vertices. During the water-flow simulation, water is permitted to evacuate to border vertices but water cannot build-up on them.

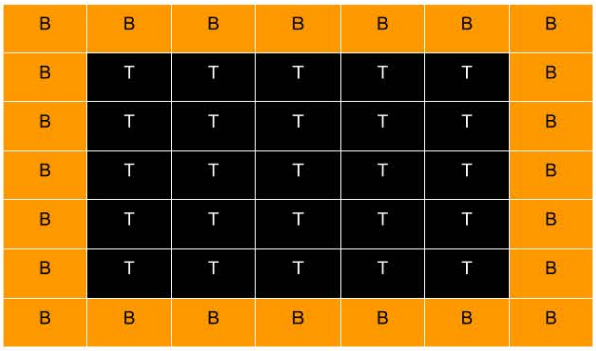

Figure 4.17: *Border of vertices generated around the terrain to cater for water evacuation at the extremities. Orange vertices form the border.*

4.3.4 Stopping the simulation

The flow of water on the terrain can be measured by keeping track of the number of vertices that are completely purged at a given iteration of the simulation. This flow will tend to be much larger at the start of the simulation when water is being evacuated from higher ground and will gradually decrease as water starts to build up and less vertices are able to purge their water content entirely. By also keeping track of the aggregated water flow of all previous iterations of the simulation, it is possible to analyse the evolution of water-flow. The system automatically stops the simulation when the water-flow calculated at iteration n is less than one thousandth of the total aggregated water which has flowed during the course of the simulation. This value was selected after a large number of trial runs and is deemed conservative as it very often leaves a significant amount of standing water on the terrain. If the user is not satisfied with the formed water network upon completion of the water-flow simulation, he is then able to manually continue the simulation for as long as necessary.

4.3.5 GPU Implementation

Processing water flow on the terrain is computationally expensive as the amount of water to evacuate needs to be calculated iteratively for each terrain vertex. To accelerate the process it is implemented to make use of the heavily parallel architecture of GPU's (see appendix I for the shader code).

GPU's have a single-core multiple thread (SIMT) architecture meaning each operation is performed simultaneously by a large number of threads on different input data. A *work-group* is a grouping of threads within the GPU architecture which are guaranteed to run in parallel and which share local memory with very fast access speeds. One or more work-groups can execute at the same time.

Storage Type	**Data Type**	**Element Count**	**Usage**
2-D Texture	Float	W × H	Water height-map
2-D Texture	Float	W × H	Cached water height-map from previous iteration in order to calculate the water-flow
2-D Texture	Float	(W+1) × (H+1)	Terrain height-map with border (see 4.3.3)
2-D Texture	Float	$W \times GroupCount_y \times 2$	Horizontal overlaps
2-D Texture	Float	$GroupCount_x \times 2 \times H$	Vertical overlaps
2-D Texture	Float	$4 \times GroupCount_x \times GroupCount_y$	Corner overlaps
2-D Texture	Unsigned int	$GroupCount_x \times GroupCount_y$	Water flow tracker

Table 4.4: *Global memory allocations necessary for the GPU implementation of the water-flow simulation algorithm where W and H are the width and height of the terrain height-map respectively.*

Below are discussed the challenges faced when implementing the water-flow algorithm to run on this heavily parallel architecture.

Copying data from CPU to GPU and vice versa is a costly process and, if substantial, can often be the bottleneck to the GPU optimisation. To prevent this, all data is copied to the GPU at the start of the simulation and only when the simulation is complete is the resulting data copied back to the CPU. The only piece of data which is copied back to the CPU between each iteration of the simulation is the aggregated water-flow in order to automatically stop the simulation when suited (see 4.3.4).

To better fit into the *openGL* pipeline used in this system, GPU implementations are done using *compute shaders*. Compute shaders are a feature of *openGL* since version 4.3 and permit the use the GPU's acceleration potential for tasks not directly linked to rendering. One of the main incentives to use compute shaders is the ability to use existing openGL data storages, notably pre-existing height-map and water-height textures used for rendering. Table 4.4 summarizes the global memory allocations necessary for the GPU implementation of the water-flow simulation.

Each thread within a work-group relates to a unique vertex on the terrain from which water must be evacuated. Water evacuated from source vertex must be added to one or many of its

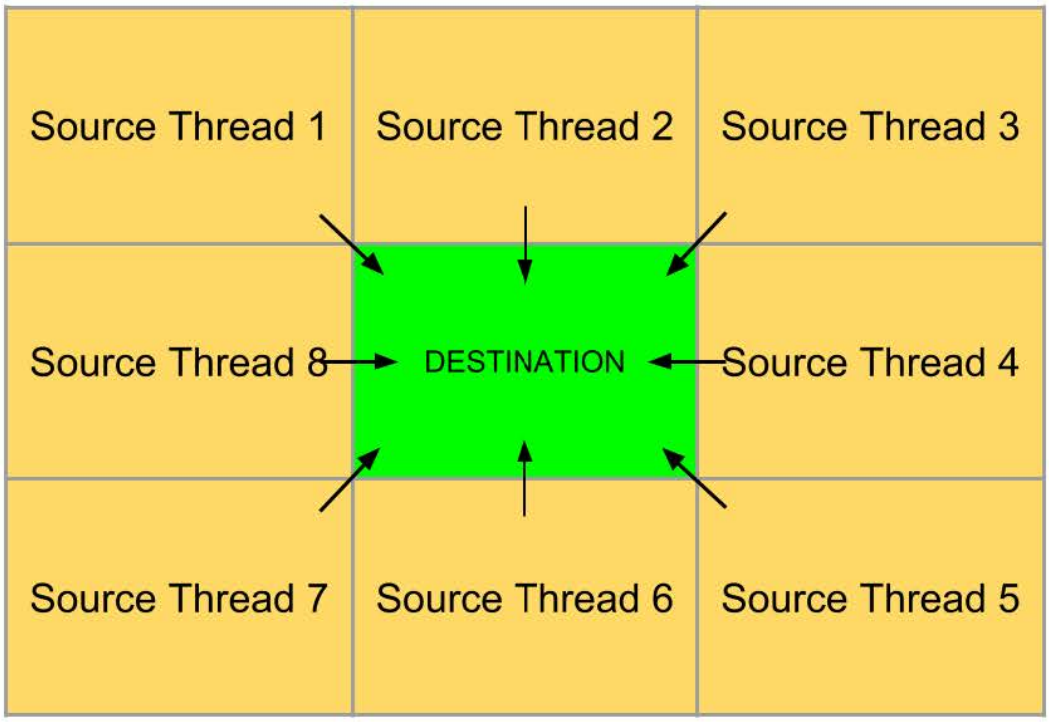

Figure 4.18: *Memory conflict cause by 8 surrounding threads attempting to write to a single destination memory location.*

eight surrounding vertex/vertices by incrementing it's value within the water height-map data structure. Memory conflicts can arise, however, when multiple threads attempt to evacuate water to the same location within this water height-map. Given a destination location D, up to eight threads can attempt to write to it simultaneously (see Figure 4.18).

One way to prevent this is to use atomic additions which ensure there are no memory conflicts between threads when writing to the same location in memory. Unfortunately, atomic additions are only possible with integer values. Here, the water levels are represented as floating points. This is important as if the water height is casted to an integer during the simulation, some water is lost. Although this isn't substantial for a single iteration, because the simulation performs thousands of iterations, it adds up. To prevent such memory conflicts with floating point values, a temporary three-dimensional array is allocated in local storage (fast-access) for each work group in which water movement is written. The x and y dimensions of this temporary array are the width and height of the associated work-group respectively and each [x,y] pair represent a unique destination on the terrain in which water can be added. The third-dimension

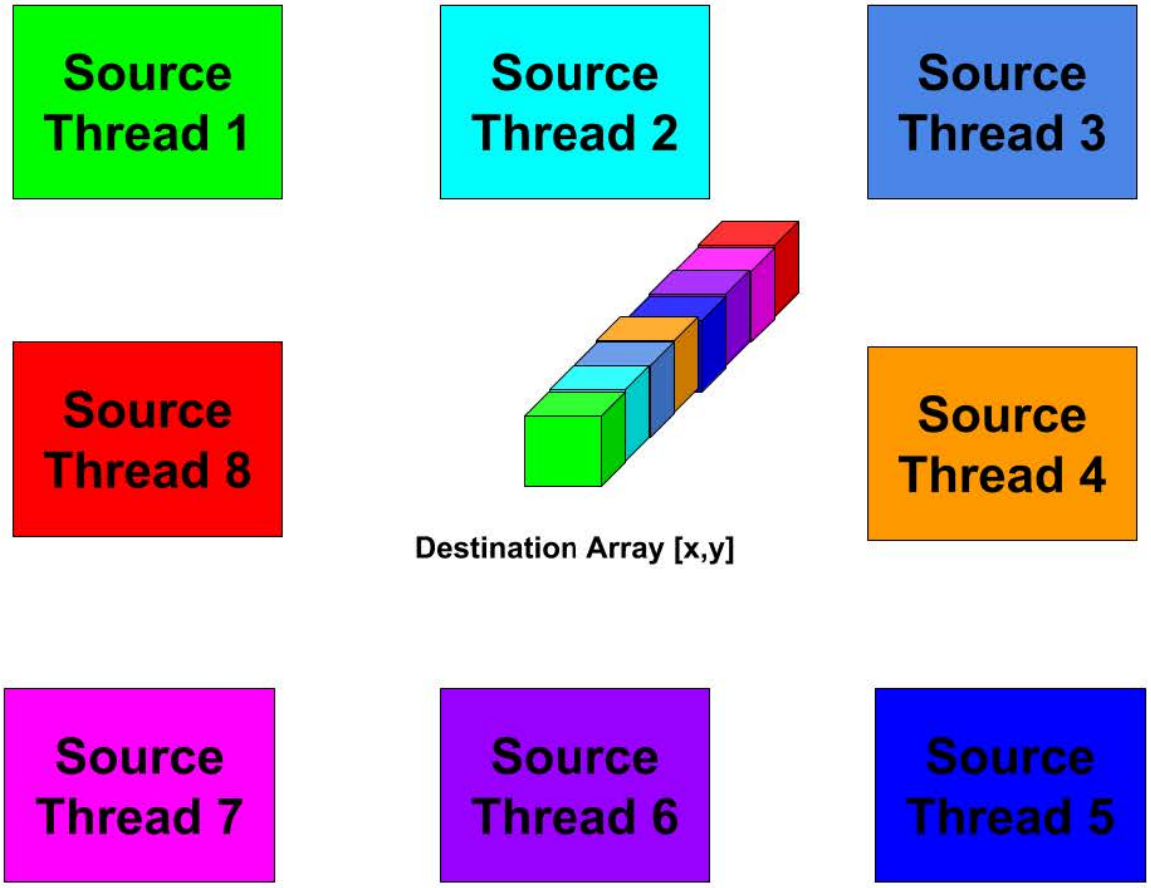

Figure 4.19: *Memory conflict prevention by using a three dimensional array to aggregate the water to add.*

serves to prevent memory conflicts by giving each source vertex a unique index in which to write water to add (see Figure4.19).

When water movement is complete for the given iteration, each thread T_{xy} within a work group reduces the third dimension of it's associated aggregate array index [x,y] by adding all values to the respective location within the water height-map.

Because local memory cannot be shared amongst work-groups, a problem arises when threads on the extremities need to place water on vertices managed by threads from a separate work-group. A work group WG_{xy} at index [x,y] may need access to neighbouring work groups in the horizontal direction (Figure 4.20), vertical direction (Figure 4.21) and diagonal direction (Figure 4.22). Global memory is allocated (2-D textures) to aggregate the data to be written to these locations. When water-flow for the given iteration is complete, this data is then reduced by selected threads and written back to the water height-map.

Figure 4.20: *Global memory allocation requirement (green) to cater for inter work group memory access in the horizontal direction. WG = work group*

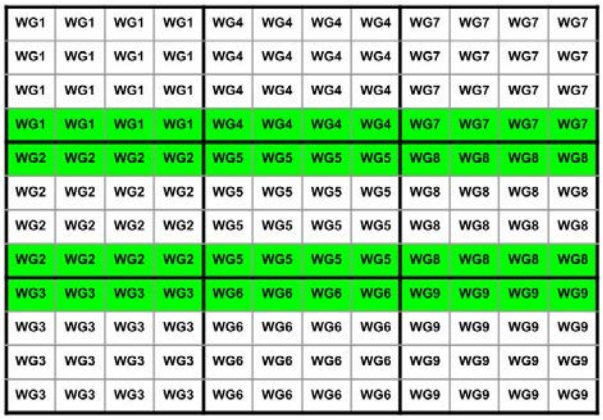

Figure 4.21: *Global memory allocation requirement (green) to cater for inter work group memory access in the vertical direction. WG = work group*

Figure 4.22: *Global memory allocation requirement (green) to cater for inter work group memory access in the diagonal direction. WG = work group*

4.3.6 Performance

Due to limited scope, a CPU version of this algorithm was not implemented and therefore, calculating GPU speed-up is not possible. The section focuses on the processing time to complete the water-flow simulation and the average time per single-iteration of the simulation for terrains varying in size.
In order to ensure consistency between the tests:

- The different terrains used are all scaled-down versions of the same seed terrain. This ensures the terrain relief, and therefore water-flow capacity, is the same.

- The amount of water to evacuate at time t_0 on each terrain vertex is the same for all simulation runs (100 millimetres).

As can be seen in the results summarised in Figure 4.23, the water-flow simulation takes approximately 22 seconds to complete for a terrain with over one million vertices (1024 $\times$ 1024 terrain). The simulation time decreases linearly with the vertex count of the terrain. A similar pattern emerges when analysing the average processing time for a single iteration of the simulation (see Figure 4.24).

4.3.7 Results

The U.S. Geological Survey [4] freely provides detailed elevation data for the north American continent. Also, Google-Earth [5] provides detailed satellite images of the same locations. To test the water-flow simulation, we use terrains downloaded from the U.S. Geological Survey. The resulting rivers and streams which are then compared with corresponding satellite images to see if the main water bodies match. For the tests to be as accurate as possible, only the water-flow simulation is performed on the terrains with not fine-tuning of soil infiltration rates. The results illustrated in figures 4.25, 4.26 and 4.27 show that the simulation successfully reproduces core water networks. Note that the replicated terrains also contain water bodies and rivers which are not present in the corresponding satellite image. The purpose of the test is not to reproduce an exact match but rather ensure that, by running only the water-flow simulation, the core water networks are reproduced. Better results could be achieved by fine-tuning the soil infiltration rates but this would bias the test as it would influence the flow of water on the terrain.

An important simplification worth noting of the water-flow simulation is that water absorption is not performed whilst it is running. In other words, when water is evacuated from

[4] `http://www.usgs.gov`
[5] `http://www.google.com/earth/`

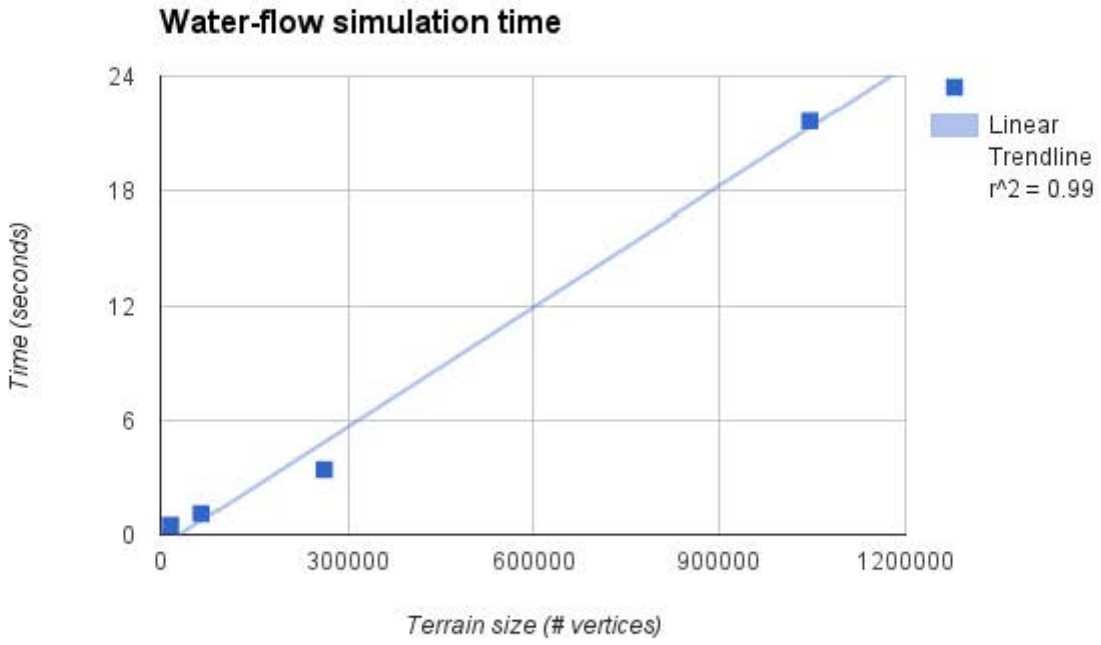

Figure 4.23: *Total water-flow simulation time for 100 millimetres per terrain vertex. Analysis performed for terrains with dimensions: 128 by 128, 256 by 256, 512 by 512 and 1024 by 1024.*

source to destination vertex, no water is then absorbed in the destination vertex. Integrating secondary absorption into the simulation would improve realism as not only would it improve the accuracy of standing water but also of the vegetation as it would lead to more precise soil humidities. This would expensive, however, as similarly to the work by Lau [Lau10] it would require the calculation of water flow velocities based on relief and soil properties. Because of scope, it was not possible to incorporate this into this water-flow simulation but would form a valuable addition in future work. As shown in the tests below, however, this simulation is still extremely valuable in determining where river networks form on the terrain.

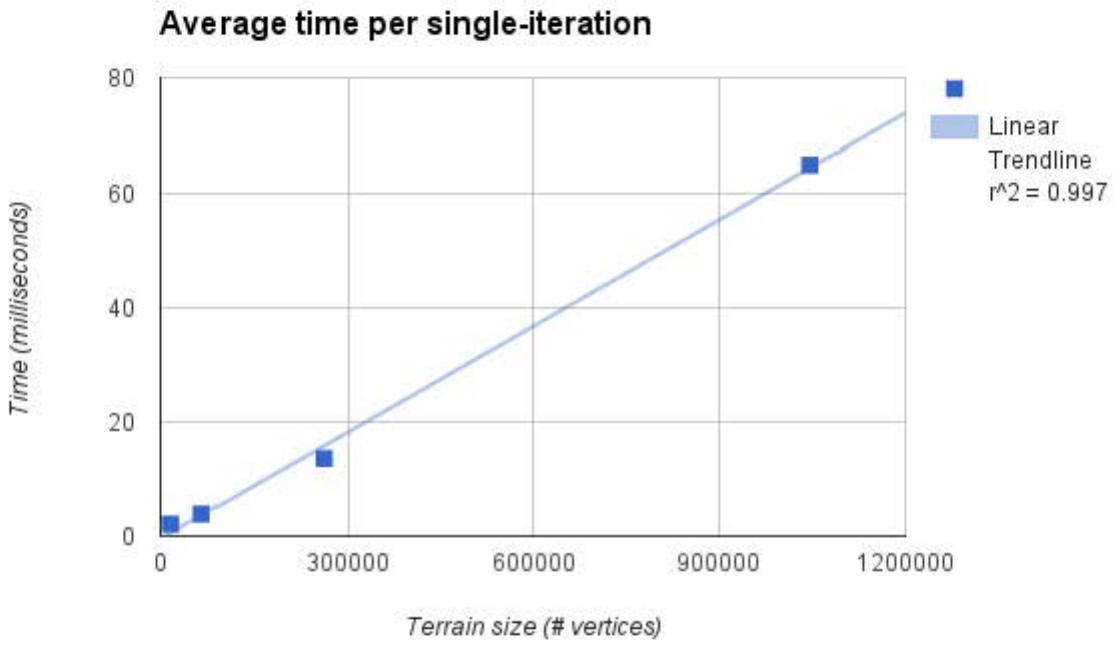

Figure 4.24: *Average time for a single iteration of the water-flow for 100 millimetres per terrain vertex. Analysis performed for terrains with dimensions: 128 by 128, 256 by 256, 512 by 512 and 1024 by 1024.*

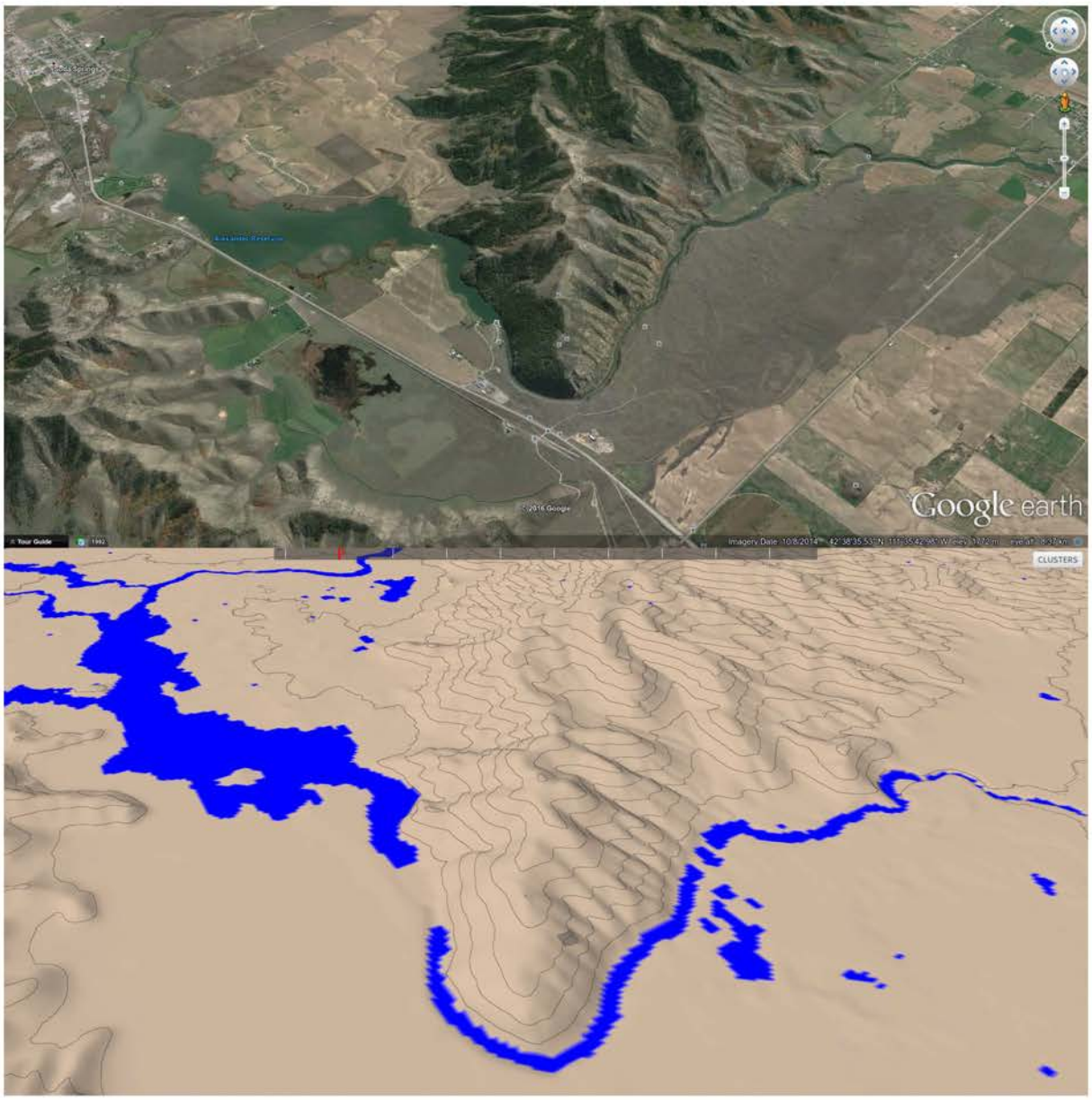

Figure 4.25: *Top: Real world water-network at geographic coordinate location: 42°38'N 111°35'W. Bottom: Replica using the water-flow simulation.*

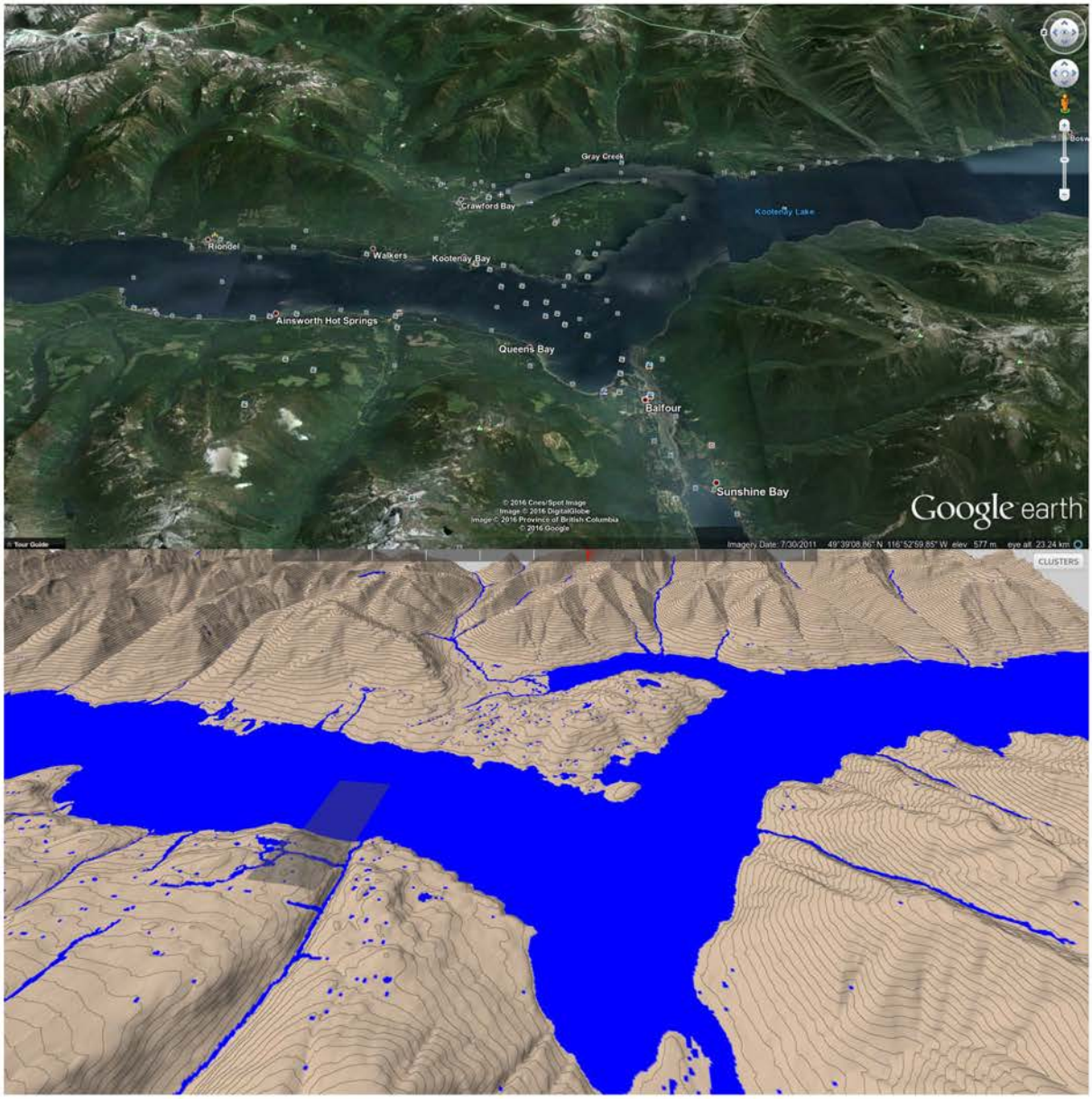

Figure 4.26: *Top: Real world water-network at geographic coordinate location: 49°39'N 116°52'W. Bottom: Replica using the water-flow simulation.*

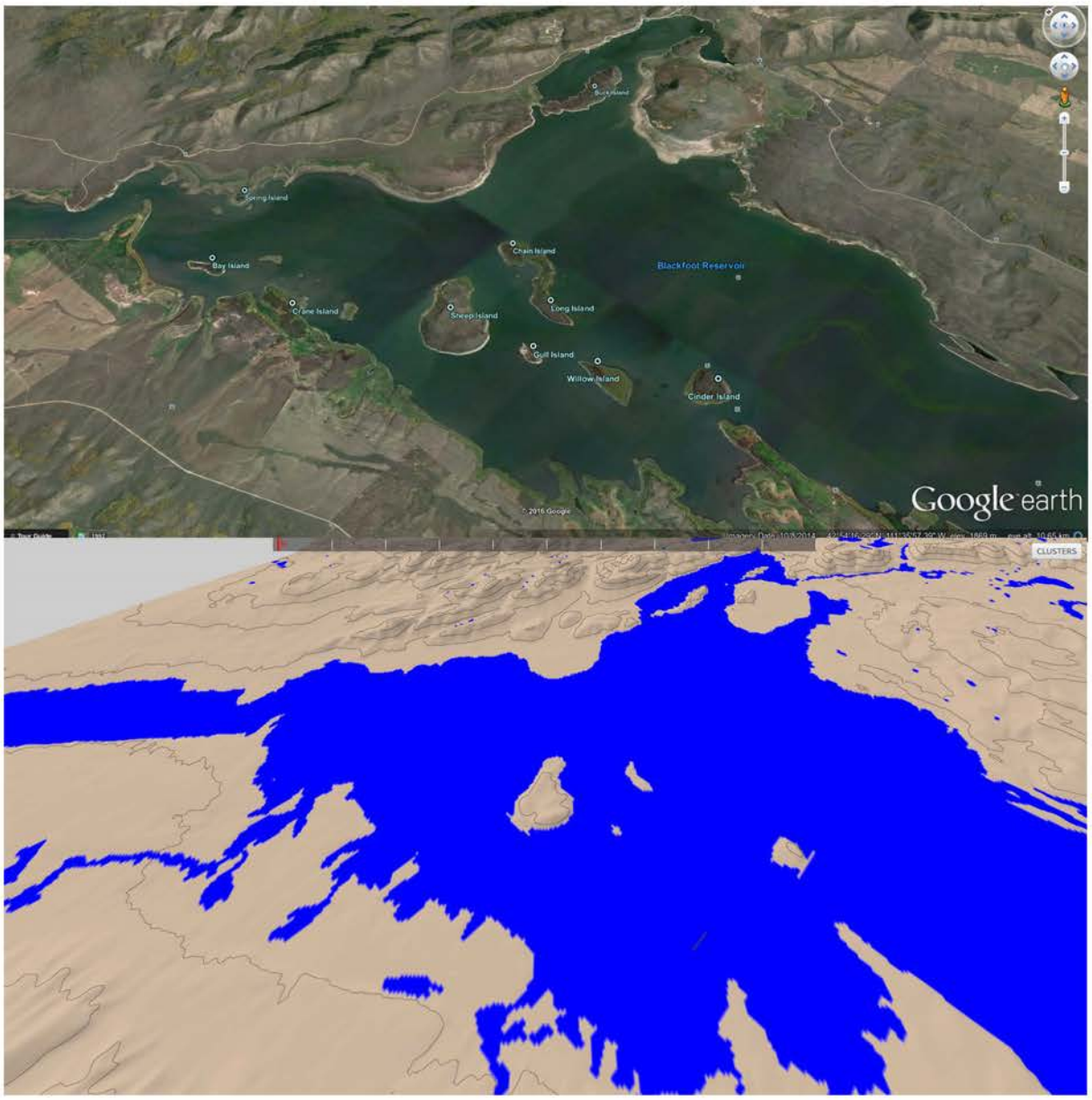

Figure 4.27: *Top: Real world water-network at geographic coordinate location: 42°38'N 111°35'W. Bottom: Replica using the water-flow simulation.*

4.4 Water Bodies

Water bodies refers here to static water build-ups such as seas, oceans and lakes. The water-flow simulation (see Section 4.3) often fails to reproduce these accurately as they are the result of years of water accumulation, groundwater and different soil infiltration rates. Users can place such water bodies by using a *flood-fill*.

Flood-filling uses a single seed point, P_{seed}, to determine the height, H_{level}, at which to set the water-level. This seed point then iteratively propagates to all surrounding points which have height H equal or lower than H_{level} using a flood-fill approach. The process continues until there are no more valid destination points or the terrain extremity is reached. When flood-filling is activated, the user specifies the seed point by simply clicking on it with the mouse pointer. The flood-filling algorithm produces water bodies in real-time even for large terrains and large water-bodies. It is possible to undo an unlimited stack of water body placements added with the flood-filling tool (Ctrl+Z). This is deemed important in case the user mistakenly specifies an incorrect seed point.

Figures 4.28 and 4.29 shows that the tool can be used to successfully place water bodies on the terrain.

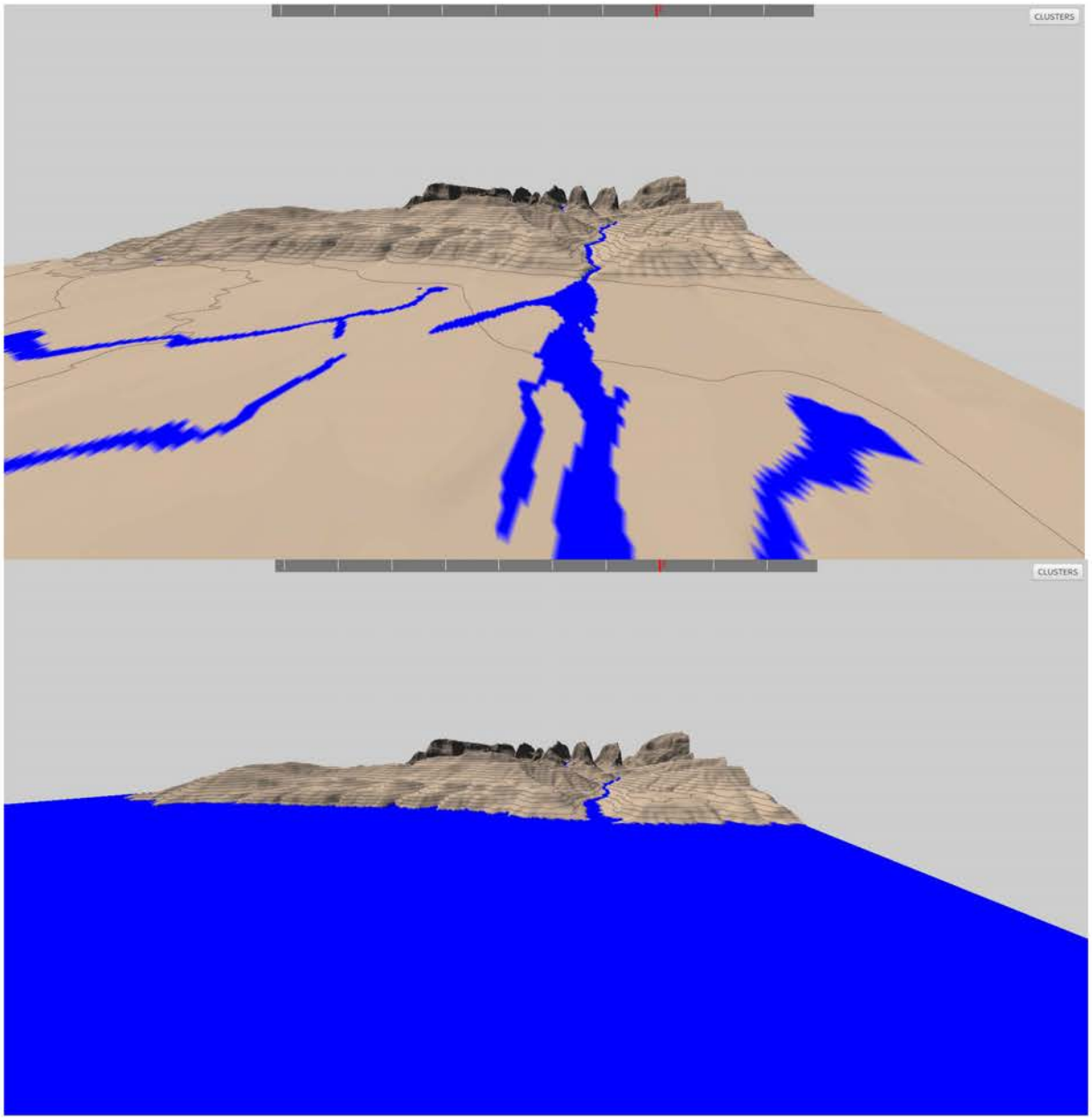

Figure 4.28: *Terrain before (top) and after (top) using the flood-fill tool to place a large water body (e.g sea or ocean).*

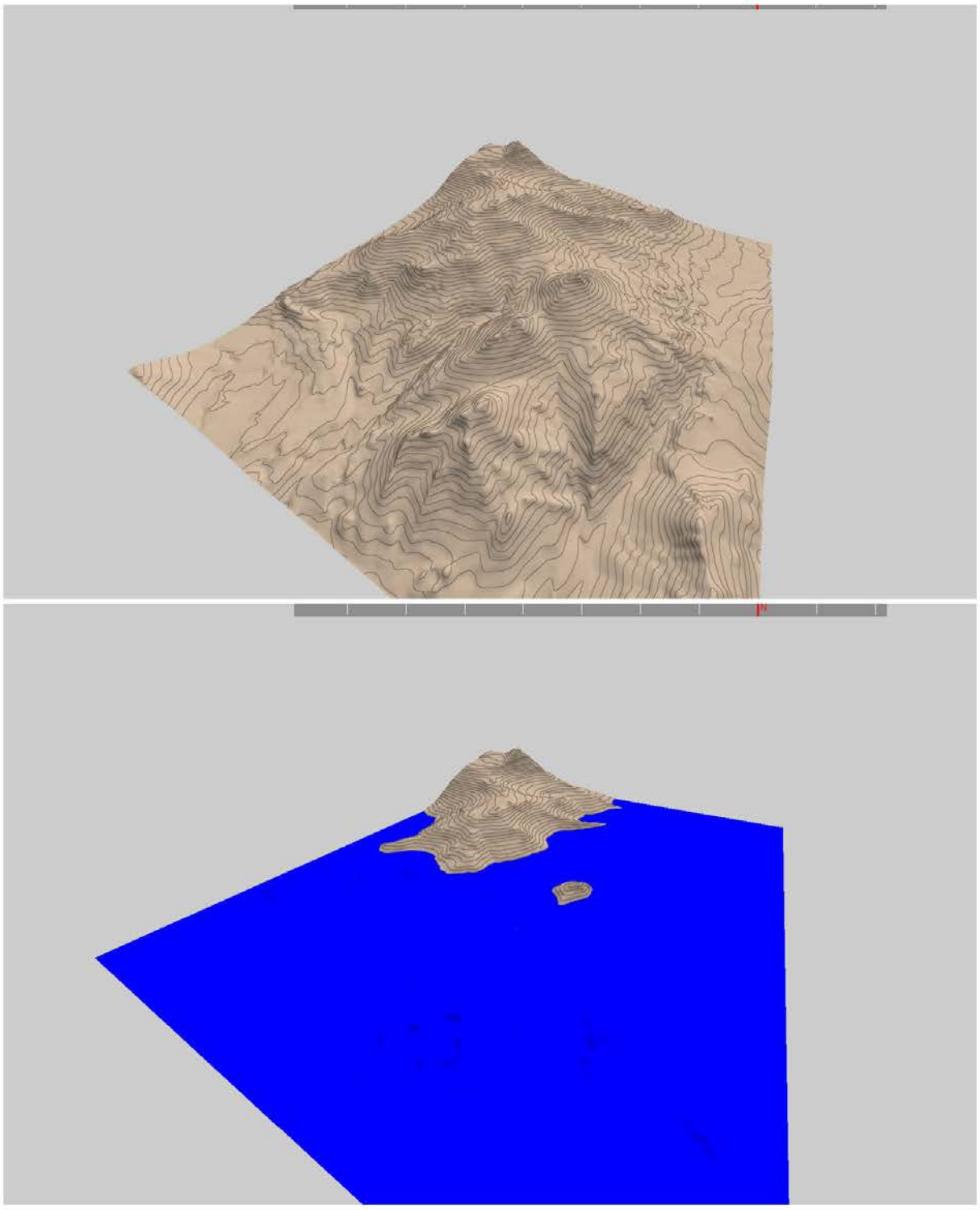

Figure 4.29: *Terrain before (top) and after (top) using the flood-fill tool to place a large water body to make an island.*

Chapter 5

Clustering

Each point on the terrain has a number of associated resource properties (summarised in Table 5.1), which are used to determine suitable vegetation and, using an ecosystem simulator, a distribution of this vegetation. An instance of the ecosystem simulator could be started to determine suitable vegetation for each individual terrain vertex and then simple interpolation performed to map the results onto the actual terrain. The ecosystem simulator is computationally expensive (see Section 6.3.8), however, and employing this technique is infeasible. As such, to make the number of ecosystem simulations manageable, clustering is performed on the terrain based on the resources associated with individual points. The clustering algorithm used is *K-Means Clustering*, discussed in Section 5.1. To accelerate clustering towards interactive feedback, it is implemented to run on the GPU, details of which can be found in Section 5.2. To conclude, the performance and results of the clustering algorithm are discussed.

Resource	**Count**	**Comments**
Slope	1	-
Temperature	12	Temperature for each month
Illumination	12	Illumination for each month
Soil Humidity	12	Soil humidity for each month

Table 5.1: *Resource properties associated each terrain vertex. Temperature, illumination and soil humidity are monthly values, hence why there are twelve.*

5.1 K-Means Clustering

K-means is a well known and broadly used vector-quantization clustering algorithm [Jai10] which groups points into clusters based on the euclidean distance from a set of K cluster means. It has been used to to tackle a wide range of problems including character recognition [PI15], image segmentation [BF98, KMN+02] and compression [KMN+02].
The algorithm is of complexity O(N) and can be optimized to make use of parallel architectures [Xu05]. Efficiency is a key requirement for the terrain clustering process in order to provide interactive user feedback, irrespective of terrain size.
K-means often fails to produce adequate clusters when dealing with large dimensionality data sets [SWF12]. Because the data here has only four dimensions (slope, temperature, illumination and soil humidity), it is well-fitted.
Common downsides of K-means clustering, however, is the necessity to configure the number of clusters to produce K and the selection of the initial K points to act as the cluster seeds. How these are handled are discussed in sections 5.1.3 and 5.1.4 respectively.

Given a set of data points P and a configured value k, k-means clustering behaves as follows:

1. Choose K points at random to act as the seed cluster means, C_{mean}.
2. Iterate through every data point P_n from P and calculate its Euclidean distance from each cluster mean using Equation 5.1. Point P becomes a member of its closest cluster.
3. Use the members of each cluster to calculate the new cluster means.
4. Repeat from step 2.

$$D(A, B) = \sum_{n=1}^{m} (A_n - B_n)^2 \tag{5.1}$$

Where: **A** and **B** are either data points or a cluster mean; $\mathbf{n} \in \mathbb{R}_m$ is the dimensionality of the data set; $\mathbf{A}_n$ is the value of data point A in dimension n.

5.1.1 Algorithm Complexity

For each clustering iteration, the algorithm needs to run through each data point twice. The first time, the distance between the data point and each mean cluster is calculated in order to determine the cluster membership. The second time is to calculate the new cluster means. Given this, the algorithm complexity is 2 × K × n × i where: K is the number of clusters, n is the number of data points and i is the total number of clustering iterations performed.

5.1.2 Normalisation

The Euclidean distance calculation outlined in Equation 5.1 works well when all values have similar scale. Unfortunately, this is not the case for terrain resource data as illumination, slope, temperature and soil humidity are all measured in different units. This means that a one millimetre change in soil humidity will have as much of an influence on clustering as a one degree change in temperature. To equalise the effect on clustering across all resources, normalisation is performed based on the range of the individual resources. Equation 5.2 is used to calculate the Euclidean distance between two terrain positions (or cluster means) A and B for clustering.

$$D(A,B) = (\frac{S(A) - S(B)}{R_S})^2 + \sum_{n=1}^{12}((\frac{T_n(A) - T_n(B)}{R_T})^2 + (\frac{H_n(A) - H_n(B)}{R_H})^2 + (\frac{I_n(A) - I_n(B)}{R_I})^2) \tag{5.2}$$

Where:$S(x)$ is the slope of a point or cluster mean x; $R_T(x)$ is the slope range; ;$T_n(x)$ is the temperature of point or cluster mean x at month n; $R_T(x)$ is the temperature range; $H_n(x)$ is the soil humidity of point or cluster mean x at month n; $R_H(x)$ is the humidity range; $I_n(x)$ is the illumination of point or cluster mean x at month n; $R_I(x)$ is the illumination range;

5.1.3 Configuring the Number of Clusters, K

The value of K will affect the number of ecosystem simulators which need to be run when placing vegetation on the terrain. Although larger values will potentially result in more realistic vegetation distributions, the added simulations will require additional processing time. Choosing a good value for K is therefore about finding a balance between realism and processing time and, as such, depends on user requirements. Too small of a K value will negatively impact realism as insufficient clusters will be generated and terrain vertices will be gathered into a same cluster even if they differ heavily in their underlying resources. Too big a value of K will negatively impact performance as ecosystem simulator runs will be performed to fill distinct areas of the terrain which are heavily similar in terms of the resources they offer. Because of this, the value of K is required as input from the user before the clustering is performed.
Admitably, K can be an abstract to users who don't have an understanding of K-means clustering. Future work could build on this and automatically calculate a suitable value of K given the variance of the resources on the terrain.

5.1.4 Choosing Seed Cluster Means

A downside of classic K-means clustering techniques is that clusters are non-reproducible. This is because the final clusters depend heavily on the initial seed points which were chosen to act as the cluster means. As these are selected at random, different runs will result in different

clusters. Reproducibility is important here, however, as if the clusters cannot be reproduced neither will the final terrain. To ensure reproducibility, the terrain positions to act as the initial clusters are chosen using a pseudo-random number generator with a predefined and fixed seed.

Storage Type	**Data Type**	**Element Count**	**Usage**
3-D Texture	Float	W × H × 12	Monthly Soil Humidity
3-D Texture	Float	W × H × 12	Monthly Illumination
2-D Texture	Float	W × H	Temperature
2-D Texture	Float	W × H	Slope
3-D Texture	Float	$K \times WorkGroups_x \times 13 \times WorkGroups_y$	Slope and Humidity Reducer
3-D Texture	Float	$K \times WorkGroups_x \times 2 \times WorkGroups_y$	Temperature Reducer
3-D Texture	Float	$K \times WorkGroups_x \times 12 \times WorkGroups_y$	Daily Illumination Reducer

Table 5.2: *Global memory allocations necessary for the GPU implementation of K-Means clustering. W and H are the width and height of the terrain, respectively.* $WorkGroups_x$ *and* $WorkGroups_y$ *are the horizontal and vertical workgroup counts, respectively.*

5.2 GPU Implementation

Performing K-Means clustering is an *O(DKN)* problem where K is the number of clusters, N is the number of data points and D is the number of clustering iterations [Xu05]. As a consequence, given a cluster count, the processing time will increase linearly with terrain area. For large terrains with millions of vertices this could prove time consuming and, consequentially, have a negative effect on user experience. To accelerate the clustering process and improve interaction clustering is implemented to make use of the heavily parallel architecture of the GPU. Below are discussed relevant details and optimizations.

5.2.1 Core Algorithm

Given the cluster means for iteration i, the algorithm must determine to which cluster each terrain vertex belongs. To do so efficiently, each GPU thread is associated with a unique vertex and is responsible for determining its cluster membership.

Once all terrain vertices have been assigned to a cluster, they must be iterated over in order to calculate the new means of each cluster.

Within a work-group (group of cores which are guaranteed to run in parallel and have access to shared memory), when each core calculates its distance from individual cluster means, first

it loads its associated resource data into a unique index of local fast-access shared memory. In this memory, it also stores the calculated cluster membership ID. Using reduction techniques, K work-group threads calculate the K new work-group cluster means. These K threads then write the calculated means to a unique index of global memory along with the number of points belonging to each given cluster.

Once all work-groups have finished calculating their associated cluster means, K threads are spawned to calculate the aggregate cluster means as described in Equation 5.3.

$$ClusterMean_k = \sum_{wg=0}^{n} WGM_{wg}(k) \times \frac{MC_{wg}(k)}{TMC(k)} \tag{5.3}$$

Where: $AggregateMean_k$ is the new mean of cluster k; wg is the work-group ID; $WGM_{wg}(k)$ is the work-group mean of cluster k in work-group wg; n is the number of work groups; $MC_{wg}(k)$ is the member count of cluster k in work-group wg; $TMC(k)$ is the total member count of cluster k

5.2.2 Optimizations

The temperature on the terrain increases linearly with altitude. As such, even though the temperature changes monthly on the terrain, the temperature difference between two points P_a and P_b will remain constant throughout the year. As a consequence, it is only necessary to use a single months temperature data to establish terrain clusters, saving vital GPU storage space.

As mentioned previously, copying data to and from CPU and GPU is a costly process. To prevent such costly transfer operations, all data required for the clustering algorithm (table 5.2) is copied to the GPU at the start of the clustering process and no further transfers are performed until completion.

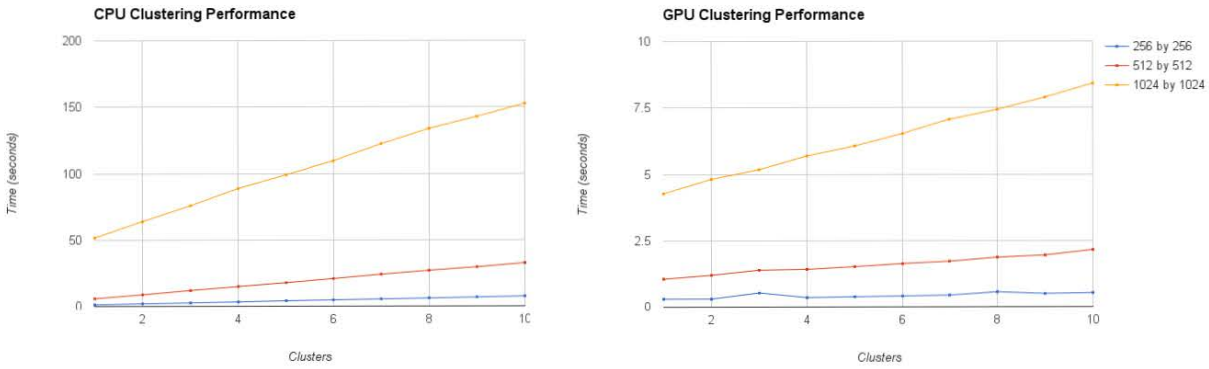

Figure 5.1: *Time it takes for the clustering process to complete on the CPU (left) and GPU (right) in relation to the number of clusters. The analysis was performed for terrains of size: 256 by 256 (blue), 512 by 512 (red) and 1024 by 1024 (orange).*

5.3 Performance

As mentioned previously, the user must specify the requested cluster count k. To to find a suitable value for k, the user will need to trial a number of values. To minimize any negative effect on user-experience it is important, therefore, that the clustering performs in near real-time, irrespective of terrain size and cluster count.

The performance of the CPU and GPU clustering implementations are analysed below along with an evaluation of the GPU speed-up. In order to evaluate the performance of the different implementations, the clustering time is analysed in relation to terrain size and cluster count. In order to accurately compare their performance, the same terrains are used with identical resources specified. In these tests, the maximum value of K is set to ten. Although the system permits users to generate up to fifty clusters, ten is deemed sufficient for most use cases. All tests were performed on a machine with specifications outlined in appendix D.

5.3.1 CPU Performance

Figure 5.1 shows the clustering time achieved on the CPU for different terrain sizes and number of clusters. From this data, it is possible to conclude that the clustering time increases linearly with the number of clusters and the clustering time is proportional to terrain area.

Although the clustering time is reasonable for smaller terrains, this processing time increases sharply with terrain size. This is especially true when combined with an increase in the number of clusters to generate. Generating ten clusters on a large terrain (1024 by 1024) takes just over 2.5 minutes.

Due to limited scope, the CPU implementation is not optimized and is single-threaded.

5.3.2 GPU Performance

Figure 5.1 illustrates the performance of the same tests run on the GPU. From this data, it is possible to conclude that the processing time increases linearly with cluster count but at a significantly slower rate than on the CPU. This is because individual clusters are managed in parallel on the GPU. Also, similarly to the CPU implementation, the processing time increases linearly with terrain area. Unlike the cluster count, however, the rate of increase is comparable to that of the CPU implementation. The reason for this is because, although individual terrain vertices are managed in parallel on the GPU implementation, the number of vertices far outweighs the number of GPU cores.

5.3.3 GPU Speed-up

Figure 5.2 shows the GPU clustering speed-up over CPU for square terrains of size 256, 512 and 1024, respectively. These graphs show that the GPU significantly outperforms the CPU, irrespective of terrain size and cluster count.
Also visible is the increased sensitivity to cluster count of the CPU implementation over the GPU (speed-up increases with respect to cluster count).
This graph also shows that the GPU speed-up increases with terrain size. Because each terrain vertex is processed by a separate thread in the GPU clustering algorithm, increasing the number of vertices will accentuate the GPU acceleration.

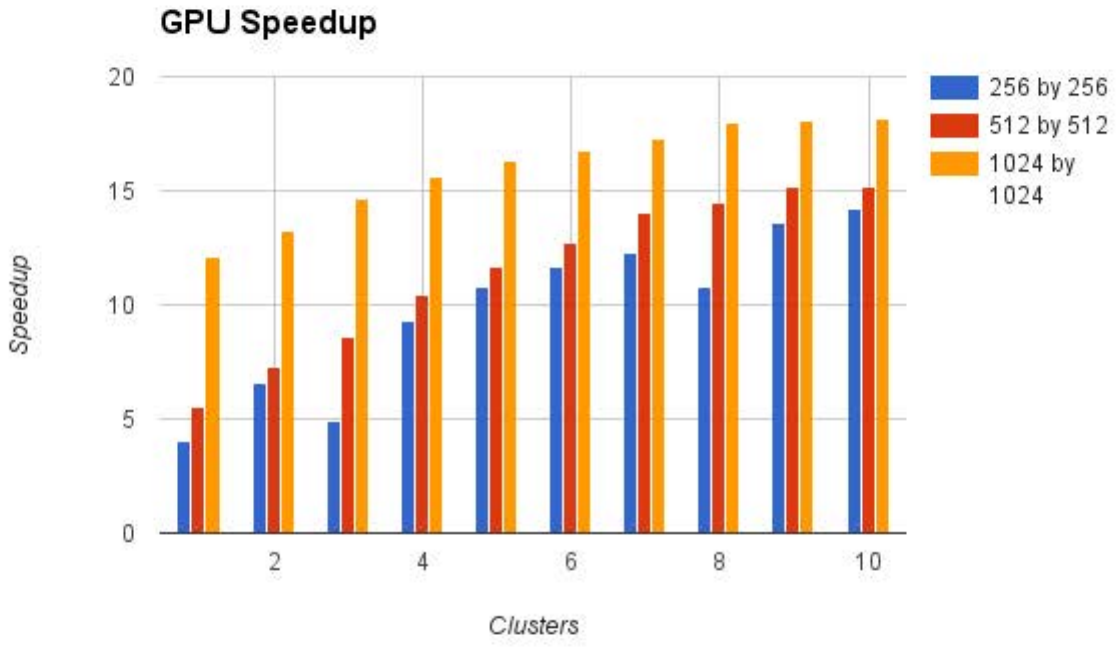

Figure 5.2: *Calculated clustering speed-up of the GPU over CPU implementation for square terrains of size 256 by 256 (blue), 512 by 512 (red) and 1024 by 1024(yellow).*

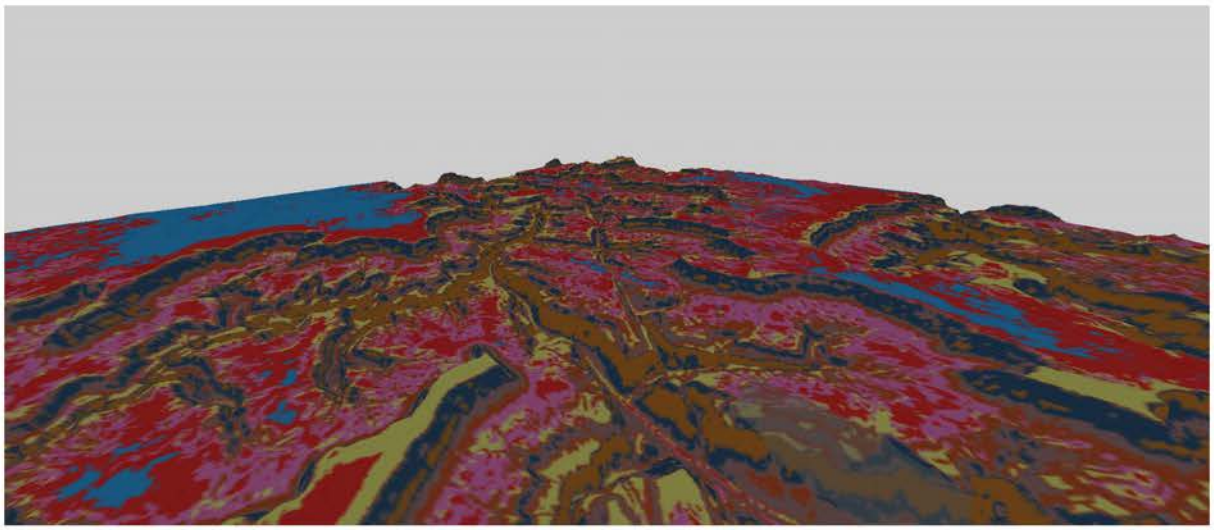

Figure 5.3: *Colour coded cluster overlay. Using this it is possible to easily identify the clusters associated to each terrain vertex.*

5.4 Overlay and Cluster Descriptions

When clustering is complete, each point on the terrain is associated with one of k unique clusters. To make this association apparent to the user a cluster overlay is displayed. The clustering overlay attributes a unique color to each cluster and subsequently to each terrain vertex based on cluster membership (see Figure 5.3). Along with the terrain overlay, a dialogue shows the properties (color, member count, resources) of each cluster (see appendix A).

Month	Rainfall (mm)
Jan	13
Feb	15
Mar	14
Apr	9
May	6
Jun	18
Jul	18
Aug	22
Sep	15
Oct	11
Nov	9
Dec	16

Table 5.3: *Monthly rainfall configured for the clustering tests.*

5.5 Results

To test the clustering algorithm, a terrain is loaded, it's resources edited and five clusters produced. These clusters are subsequently analysed to ensure they successfully detect distinct resource features on which to cluster.

The terrain used is a model of the Grand Canyon using data from the US Geological Survey [1]. This terrain is chosen as its canyons and crevasses make ground illumination vary greatly.

The following resource edits were performed on the terrain:

- *Latitude*: Set to zero degrees (equator)
- *Soil Infiltration*: 5 millimetres for all terrain points with a slope under 30 degrees. All points with a slope over 30 degrees were set to 0 to simulate a cliff.
- *Rainfall*: See Table 5.3.
- *Temperature*: 0 degrees at 0 meters in December. 15 degrees at 0 metres in June. Lapse rate at default value of 6.4 degrees per thousand metres.

The resulting terrain clusters that form are displayed in Figure 5.4 and summarized in appendix A.

[1] `http://www.usgs.gov`

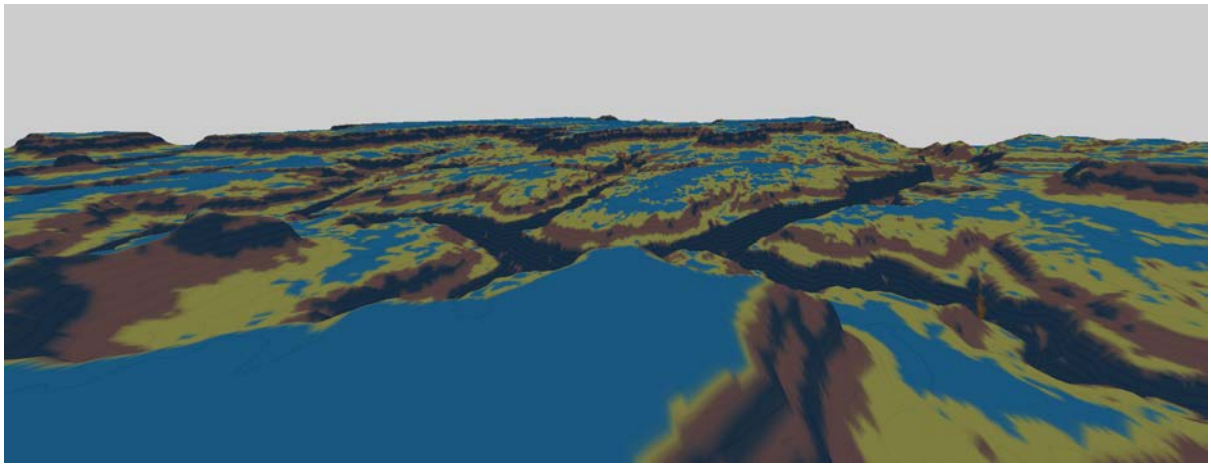

Figure 5.4: *Clustering test: Resulting terrain clusters*

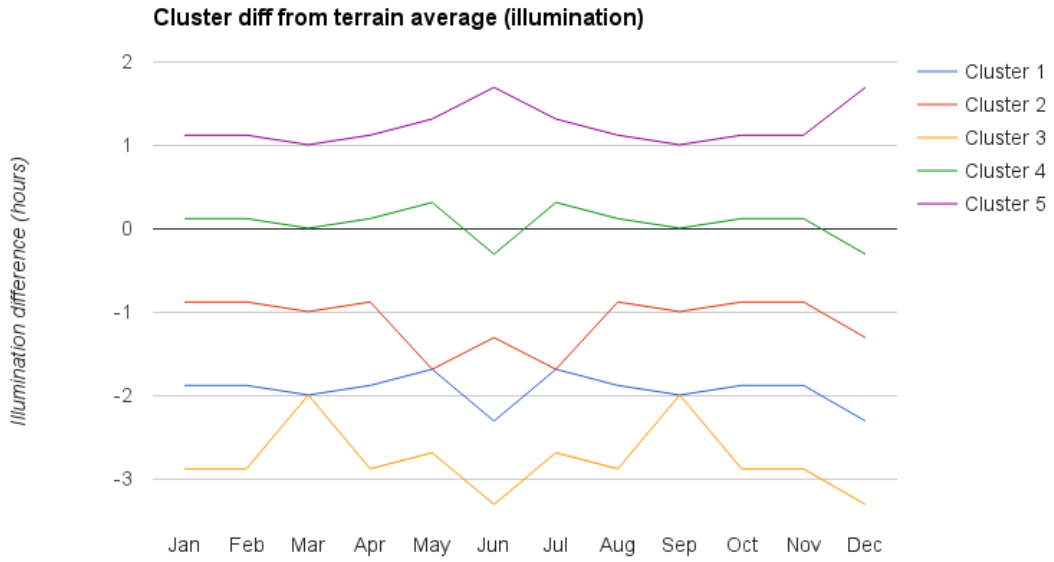

Figure 5.5: *Monthly illumination for each cluster and the average over the whole terrain.*

Cluster ID	**Slope difference (degrees)**
1	-2.5
2	36.7
3	59.5
4	-18
5	-53.5

Table 5.4: *Difference of slope between the means of each cluster and the terrain mean.*

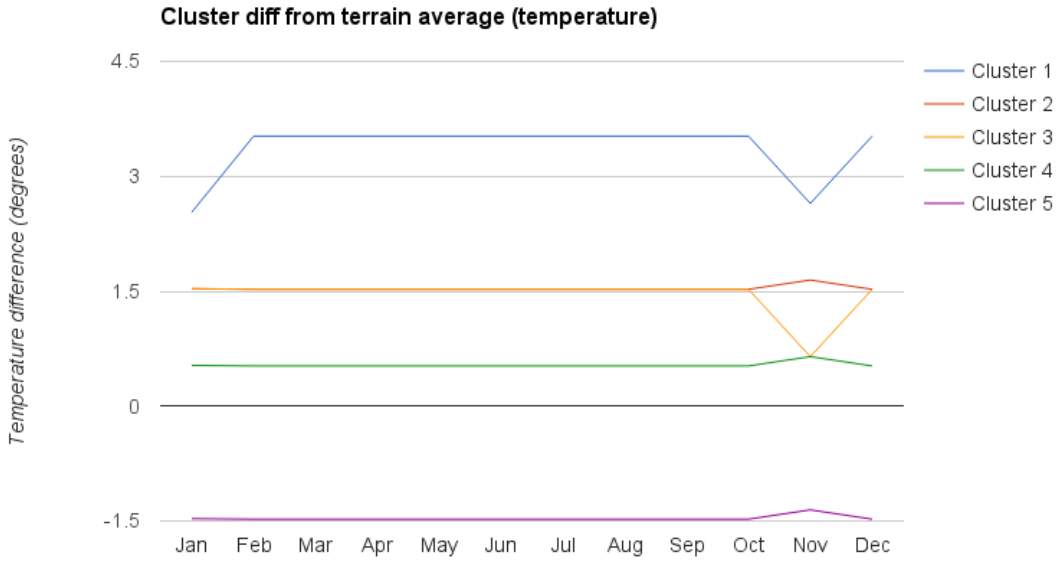

Figure 5.6: *Monthly temperature for each cluster and the average over the whole terrain. Cluster 2 has the same values as cluster 4.*

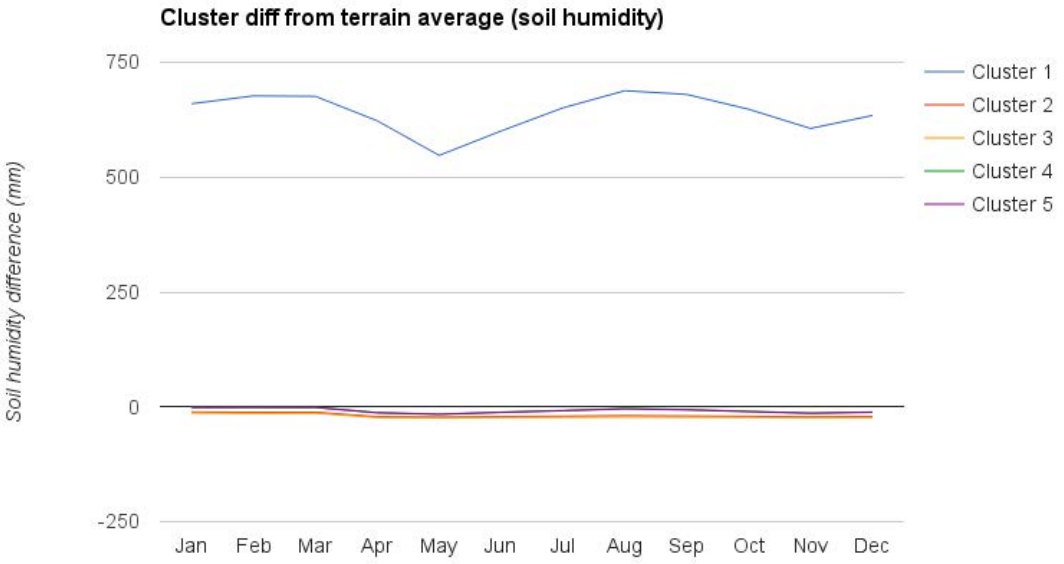

Figure 5.7: *Soil humidity for each cluster (same for every month) and the average over the whole terrain.*

Cluster	Illumination	Temperature	Soil Humidity	Slope
1	4(-)	5(+)*	5(+)*	1(-)
2	2(-)	4(+)	3(-)	3(+)
3	5(-)*	2(+)	4(-)*	5(+)*
4	1(+)	1(+)	1(-)	2(-)
5	3(+)*	3(-)*	2(-)	4(-)*

Table 5.5: *Comparison of cluster feature variance from terrain average on a ranking of 1 (least) to 5 (most). The symbol states whether the variance is positive (+) or negative (-). The minimums and maximums for each resource are represented with a *.*

Figures 5.5, 5.6, 5.7 and Table 5.4 show how much each cluster's illumination, temperature, soil humidity and slope vary from the terrain's average.

From Figure 5.5, which summarises the resource variance of each cluster, it is possible to identify key terrain features the represent, notably: *Cluster 1* is formed of the data points within the flat bottom of the canyon where the rivers form. Hence the low temperature caused by the low altitude, the low illumination caused by the surrounding canyon walls casting shade, the extremely high humidity caused by the river stream passing through and the flat bottom causing the slope to be very low. *Cluster 2* constitutes the points at the top of the canyon cliffs where slope reduces. Hence the high slope and the low humidity (water run-off). *Cluster 3* contains the data points on the cliffs of the terrain, hence the high slopes, low humidity (water run-off) and low illumination (cliffs often in the shade). *Cluster 4* is formed of the data points most similar to the terrain mean, hence its limited variance. *Cluster 5* is formed of the areas of high altitudes where the surface is flat, illumination is high (nothing to shade it) and temperatures are low.

Chapter 6

Vegetation

Vegetation is an essential part of rural terrains is vegetation. Available resources determine which plant species are able to grow and to what extent they strive in a given environment. Reproducing this link between species and climate is essential to determining suitable vegetation and, subsequently, generating plausible terrains.

To determine environments suited for given species, they are configured with associated resource requirements as outlined in Section 6.1. Given these properties, it is possible to automatically filter out ill-suited plants. Information about this automatic filtering is outlined in Section 6.2.

Although a multitude of plants can grow in a given environment, some will naturally flourish more than others. This can be because resources are more suitable or they have a faster, more aggressive growth rate. To model this intra-species battle for resources and determine a suitable vegetation state, an ecosystem simulator is used which models the natural process of plants battling to capture available resources to optimize health and, therefore, growth.
Plant growth is highly complex and is driven by a multitude of processes interacting simultaneously [FZS+08]. As such, any plant growth modeller will be a drastic simplification of the real-world equivalent. The level of detail of existing plant growth modellers depend heavily on there target use. Those used to optimize crop yield, for example, need to model the process in great detail for the results to be trustworthy [SSBR01, SED03, Yan04]. Downsides of such detailed modellers, however, is that they are often fine-tuned to specific plant species and require large amounts of configuration and processing time.
Because this system targets visual realism and not botanical realism, focuses on real-time interactivity and needs to determine plausible distributions for large areas, a minimalistic approach is taken, inspired by the work by Deussen et al. [DHL+98]. Although it extends their work by modelling resource requirements and availability in greater detail, important factors are still discarded, including: soil nutrients, soil depth and plant geometry modelling to accurately

model photosynthesis and light propagation. Details of the ecosystem simulator used here can be found in Section 6.3.

The ecosystem simulator is computationally expensive and can take some time to determine a valid distribution. The simulation time is dependent on the number of plant instances, the simulation area and the timespan. To accelerate the process, the ecosystem simulator is run over a small area and the resulting distribution analysed in order to efficiently reproduce it on larger areas. A caching system is also used to prevent users from having to run the same costly simulation more than once. Information about these features is discussed in Section 6.4.

Property	Value	Units
Slope	Start of decline	Degrees
	Maximum	Degrees
Growth	Maximum canopy	Centimetres
	Maximum root size	Centimetres
	Maximum height	Centimetres
Ageing	Start of decline	Months
	Maximum age	Months
Seeding	Maximum seeding distance	Metres
	Annual seed count	-
Illumination	Start of prime	hours
	End of prime	hours
	Minimum	hours
	Maximum	hours
Humidity	Start of prime	millimetres
	End of prime	millimetres
	Minimum	millimetres
	Maximum	millimetres
Temperature	Start of prime	degrees
	End of prime	degrees
	Minimum	degrees
	Maximum	degrees

Table 6.1: *Summary of the properties which must be configured with each plant species.*

6.1 Plant Species

A database is used to store all plant species and their associated properties, which are used to determine their ability to grow in a given environment and, subsequently, deduce a plausible distribution using the ecosystem simulator. A dedicated tool can be used to interact directly with the database in order to add, remove and edit this data.

When configuring a new species it is necessary to specify a set of associated properties. These properties can be split into two main categories: *simulation-based* and *environment-based*. *Simulation-based* properties are those used only by the ecosystem simulator to simulate the growth and spawning of new plants. *Environment-based* properties are those used by the simulator to calculate the strength of the plant but also to determine whether or not it is suited to given environments. Each are discussed below and summarized in Table 6.1.

6.1.1 Simulation-based Species Properties

To model the growth of a plant species in the ecosystem simulator, it is necessary to specify: *Maximum height*, *maximum canopy width* and *maximum root size*. Using these along with the specie's ageing properties, it is possible to simulate the plants vertical growth (height), horizontal growth (canopy) and root coverage. A plants height and canopy width is also used to determine the shade it projects on other plants during the simulation. Furthermore, the plant's root growth is used to determine how far the plant can reach to fetch soil water. Note that a maximum canopy width of zero can be specified to model plants with no canopy.

Biological life-cycle varies greatly between plant species. Whereas annual and biennials have a fixed lifespan of one and two years, respectively, perennial plant species can live far longer. To model the life-cycle of different plant species they must be configured with an associated *age of start of decline* and *maximum age*. Using these two values, it is possible to simulate a plant getting weaker and, therefore, becoming more susceptible to domination from surrounding plants.

It is necessary to replicate the spawning of offspring in the ecosystem simulator for two core reasons: *Propagation*: Plants propagate on a terrain by producing new offspring which attempt to spawn and invade different areas. *Succession*: New plants spawn to later succeed older and weaker plants of the same specie.
The two most common ways for plants to spawn new offspring is through sexual and asexual reproduction. Asexual reproducing species often spawn cloned offspring through budding (e.g potato). Sexual reproducing species, on the other hand, require chromosome exchange between males and females in order to produce unique offspring often propagated via seeds or spores. Although biologically different, both can be considered identical for the sole purpose of modelling propagation and succession. The reproduction characteristics of a given species which will influence *propagation* and *succession* in the simulation and therefore need to be configured, are the *number of offspring produced annually* and the *maximum distance from source to offspring*.

6.1.2 Environment-based Species Properties

Steep slopes causes essential water and soil nutrients to run-off, making them less rich and, therefore, less suited to plant growth [KD01]. The slope angle can also cause larger species to struggle in supporting their own biomass. For this reason, steeper slopes often cater better to smaller plant species (grass, shrub, etc.). To model the effect of slope on given plant species, when configuring a new plant species, the *slope of start of decline* and *maximum slope* must be

configured.

Illumination, soil humidity and temperature also have a great impact on plant growth and survival [FZS+08].
Whereas some species thrive in shaded undergrowth, others require direct illumination all year round. Soil water deposited into the soil by either rainfall or existing groundwater is absorbed by plant roots and is vital to their development and survival. Some species have evolved to survive in arid climates with very little water, others require frequent downpours of rain. To simplify water requirement specifications for different plant species, we ignore groundwater and consider rainfall as the plants only source of water. Some species are able to withstand extremely low temperatures (e.g at high altitudes), others have the ability to survive in extremely hot temperatures (e.g deserts).
To configure the illumination, soil humidity and temperature requirements of a given species, it is necessary to configure for each the *minimum*, *prime range* and *maximum*. The minimum represents the minimum illumination (hours), soil humidity (millimetres) or temperature (degrees) necessary for the species survival, the prime range are the values at which the resource is deemed optimal and the maximum is the upper limit after which the plant is unable to survive.

6.2 Plant Suitability Filtering

Once the terrain clusters have been generated, the user must specify the plant species to incorporate. The ecosystem simulator is then used to determine a suitable distribution for the species given the resources associated with the individual clusters.

Rather than permit the user to select any plant from the database, including those unable to grow, a filtering pass is performed in order to display only the plants best able to survive. This is useful as it prevents users from triggering an ecosystem simulation run with species that are guaranteed not to survive. To determine whether a given species is suited, a *species suitability score* is calculated for each species based on the resources of each cluster.

As well as being used to filter out ill-suited species, this suitability score also highlights the species best suited to the given environment and could, as a consequence, prove to be useful information for the user when selecting plant species. Various methods are used to effectively communicate the suitability score, details of which are discussed below.

6.2.1 Calculating the Species Suitability Score

The species suitability score associated with a given species, S, for cluster C, illustrates how suited species S is to the environment of cluster C on a range from 0 (completely ill-suited) to 100 (perfect conditions). To calculate this, the resource requirements of species S are matched with the resource availability of cluster C. To determine this score, it is first necessary to determine the specie's suitability to the environment in terms of *slope*, *illumination*, *soil humidity* and *temperature*. A separate score is calculated for each as discussed below. Note that no filtering is based on illumination as it varies during the simulation as the canopy of taller plants shade that of the smaller ones.

The slope suitability score determines how well suited the species is in terms of slope and is calculated as illustrated in Equation 6.1.

$$SS(S,x) = \begin{cases} 100, & \text{if } x \leq S_{sod} \\ 0, & \text{if } x \geq S_{max} \\ (1 - \frac{x - S_{sod}}{S_{max} - S_{sod}}) \times 100, & \text{otherwise} \end{cases} \tag{6.1}$$

Where: *SS(S,x)* is the slope suitability score for species S given slope x; S_{sod} is slope of start of decline configured for species S; S_{max} is the maximum slope configured for species S.

Because the soil humidity and temperature vary on a monthly basis, it is necessary to calculate the score for each month as illustrated in Equation 6.2. The mean of these twelve

months is then calculated as illustrated in Equation 6.3 to represent the overall suitability score for the given resource.

$$MRS(S,r,x) = \begin{cases} 0, & \text{if } x < S_{min}(r) \text{ or } x > S_{max}(r) \\ \frac{x - S_{min}(r)}{S_{ps}(r) - S_{min}(r)} \times 100, & \text{if } x \in [S_{min}(R), S_{ps}(R)] \\ 100, & \text{if } x \in [S_{prime_start}(r), S_{prime_end}(r)] \\ (1 - \frac{x - S_{pe}(r)}{S_{max}(r) - S_{pe}(r)}) \times 100, & \text{if } x \in [S_{pe}(r), S_{max}(r)] \end{cases} \tag{6.2}$$

Where: *MRS(S,r,x)* is the monthly suitability score for species S and resource r given resource value x at the given month; $S_{min}(r)$ is the minimum configured for species S and resource r;$S_{max}(r)$ is the maximum configured for species S and resource r; $S_{ps}(r)$ and $S_{pe}(r)$ constitute the start and end of the prime range configured for species S and resource r, respectively.

$$RSS(S,r) = \begin{cases} \frac{\sum_{m=1}^{m=12} MRS(S,r,value(r,m))}{12}, & \text{if } MRS(S,R,rv(m)) > 0 \text{ for } m \in [1,12] \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

Where: *RSS(S,r)* is the resource suitability score for species S and resource r; *value(r,m)* is the value of resource r at month m; *MRS(S,r,rv)* is the monthly resource score for species S, resource r and resource value rv (see Equation 6.2);

The overall suitability score gives an overview of the species suitability to the environment for all resources and is calculated as described in Equation 6.4

$$OSS(S,sl) = \begin{cases} \frac{SS(S,sl) + \sum_r RSS(S,r)}{4} \text{ for } r \in ITS, & \text{if } SS(sl) > 0 \text{ and } RSS(S,r) > 0 \text{ for } r \in ITS \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

Where: *OSS(S)* is the overall suitability score for species S; AR = {temperature,soil humdity }; *RSS(S,r)* is the resource suitability score for species S and resource r (see Equation 6.3); *SS(S,sl)* is the slope suitability score for species S given slope sl (see Equation 6.1);

6.2.2 Limitations of the Specie Suitability Score

The equations used to calculate the specie suitability scores are linear. That is, the suitability will increase and decrease proportiananlly as it gets closer or further from the optimal range, respectively. This is a simplification, however, as in nature the correlation is not necessarily linear.

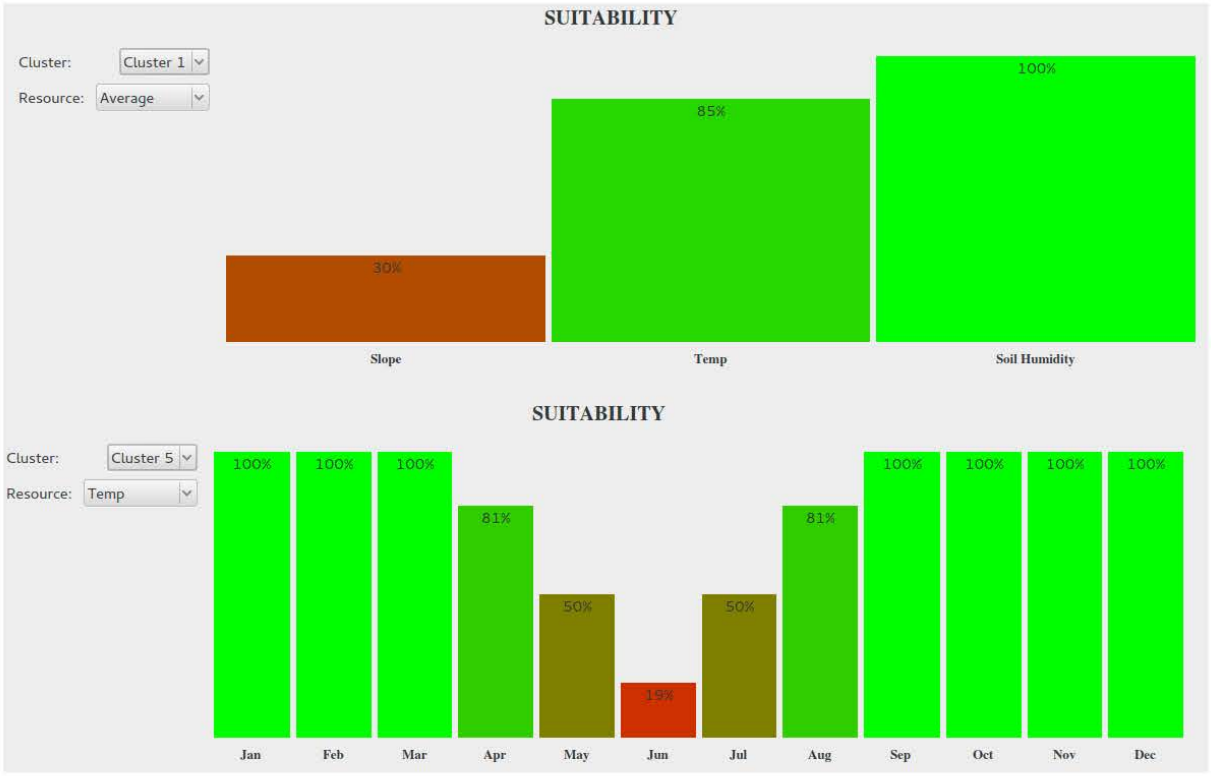

Figure 6.1: *Overall (top) and temperature (bottom) intermediate species suitability histograms. Not displayed but also present are the humidity intermediate species suitability histograms.*

6.2.3 Communicating the Species Suitability Score

When all the terrain clusters have been created, the terrain suitability score for each species in the plant database is calculated in relation to the resources of each individual cluster. If the calculated score is zero for all clusters, the species is automatically filtered out to prevent the user from selecting it.

To further communicate this information, selectable species are sorted in descending order of their overall species suitability score. Colour coding is also used where each species is associated with a colour ranging from red (very ill-suited) to light green (completely suited).

When the user selects a given species, all intermediate scores which were used to calculate the *species suitability score* are communicated to the user in histogram form (see Figure 6.1).

6.3 Ecosystem Simulator

Once the user selects the union of all species to appear in all clusters of the terrain, it is necessary to determine a valid vegetation distribution for each. To do so, an ecosystem simulator is used as in the work of Deussen et al [DHL+98] and Lane and Przemyslaw [LP02]. Unlike these other ecosystem simulators, however, our approach is not based on L-Systems, and models both resource requirements and resource availability in greater detail. The purpose of the ecosystem simulator is to determine, given a vegetation state, S_t at time t, the state S_{t+n} at time $t+n$, for any value of n. To do so, the simulation advances through time at monthly intervals and the strength of all plant instances are iteratively re-calculated (see algorithm 1). This strength of a given plant depends on it's age, available resources and surrounding plants with which it is competing for resources. This calculated value directly influences the plants growth and ability to survive.

6.3.1 Gridded Simulation Area

The simulation area greatly effects the performance of the ecosystem simulator and, therefore, it is necessary to keep it to a minimum. However, too small a simulation area will fail to accurately model the interaction of larger plant species. Given these constraints, a simulation window of one hundred by one hundred meters is used, accurate to the nearest centimetre. This area is deemed conservative, however, as rare are the species which come remotely close to such spatial coverage. An extension to this work would be to adjust the size of the simulation window depending on the species selected. This would ensure optimal simulation speed for all simulation runs.

When iteratively calculating the strength of plant instances, it is necessary to quickly determine the set of plants S = {P_1, P_2, P_3, ...} competing for available resources with P_n. Determining S depends on the spatial reach of P_n. Spatial awareness is therefore a key requirement of the simulation and is achieved by splitting the simulation window into a grid of smaller cells.

The size of individual cells can be configured to increase or decrease the resolution and, therefore, the accuracy of the simulation. As the simulation progresses, plants grow, their spatial coverage increases, and they enter new grid cells. When a plant enters a new grid cell, it becomes a member and cell resources are distributed to it. The information associated with each individual grid cell can be split into two categories: *time-dependent* and *simulation-dependent*. The time-dependent information depends only on the current month, is identical for every grid cell and comprised of: the *soil moisture* and the *illumination*. The simulation-dependent infor-

Algorithm 1 *Ecosystem simulator algorithm.*

Require: Initialize C as the set of all simulation cells.

Require: getRootIntersectPlants(C_n) is a function which returns all plant's which roots intersects cell C_n.

Require: getCanopyIntersectPlants(C_n) is a function which returns all plant's which canopy intersects cell C_n.

Require: allocateSoilMoisture(p, $Cn_{moisture}$) is a function which allocates moisture to plant p given the moisture available $Cn_{moisture}$ in the given cell.

Require: allocateIllumination(p, $Cn_{illumination}$) is a function which allocates illumination to plant p given the available illumination $Cn_{illumination}$ in the given cell.

Require: calculateStrength(p) is a function which calculates the strength of plant p based on the resources distributed to it and it's age.

Require: grow(p) is a function which grows plant p based on it's strength and species growth properties.

Require: killIfNecessary(p) is a function which kills off plant p based if necessary based on its calculated strength.

1: **for** C_n in C **do**
2: rootIntersectingPlants = getRootIntersectPlants(C_n) RootSize(P_n)
3: **for** p in C_n **do**
4: allocateSoilMoisture(p, $Cn_{moisture}$)
5: allocateIllumination(p, $Cn_{illumination}$)
6: calculateStrength(p)
7: **end for**
8: **end for**
9: **for** C_n in C **do**
10: **for** p in C_n **do**
11: grow(p)
12: killIfNecessary(p)
13: **end for**
14: **end for**

mation varies throughout the simulation as plants spawn, die and grow and consists of: the *list of plants whose roots intersect the cell* and the *list of plants whose canopy intersects the cell*. It is important to have both as plant roots and canopies grow at different rates and their cell coverage will therefore differ.
By employing this gridded approach, determining the set of plants which compete for resources is extremely efficient. For example, to determine the set of plants with which P_n is competing for soil moisture, it is only necessary to determine the cells covered by its roots, which depends solely on its position and root size.
Another advantage of this gridded approach, which is discussed in further detail below, is that it splits plants into separate cells, each with unique resource distributions. This permits partial shading, for example, where illumination is zero in some of its cells but more in others.

6.3.2 Soil Moisture Distribution

Plants grow their roots in order to access the nutrients and moisture available in the surrounding soil. As roots of different plant instances overlap, they compete for these resources. A notable simplification in our work is that soil depth is not modelled. Soil depth affects the plants rooting reach and has a significant impact on plant growth [FZS+08]. A a future extension to this work, the soil could be modelled as layers, which would permit plants to battle for different soil resources depending on their root depth. For example, grass and shrub with small root depth would access soil moisture in the upper most layer but large trees would access it in the deeper most layer. They would therefore not be competing for soil moisture.

The strength of each plant in the simulation must be recalculated on a monthly basis. Part of the information required to calculate the overall strength of a given plant is the moisture allocated to it, which is taken as the average of the moisture allocated to it in each cell its roots overlap. To determine the overall moisture allocated to a given plant p, it is first necessary, therefore, to iterate over all incident grid cells and calculate the moisture allocated to each plant with intersecting roots.

When distributing the soil moisture in a given grid cell C_{xy} to the set $S = \{P_1, P_2, P_3, ...\}$ of plants whose roots intersect the cell, one of three distinct scenarios can occur, depending on the available moisture, $M_{available}$, of the cell: *Abundant*, *sufficient* and *insufficient*.

The moisture is deemed abundant if the available moisture, $M_{available}$, surpasses 300 millimetres. To prevent situations where the soil moisture attributed to a given plant is small simply because the majority of the available moisture is distributed to other plants, all plants of S are allocated $M_{available}$. This is important as it could lead to situations where species strive

in areas completely unsuited. The available soil moisture is directly dependent on rainfall. At three hundred millimetres, rainfall can be considered abundant and soil moisture therefore not a limiting factor.

If $M_{available}$ is less than 300 millimetres, it is necessary to determine whether the moisture is sufficient or insufficient by calculating the requested moisture $M_{requested}$, as outlined in Equation 6.5.

$$M_{requested} = \sum MinMoisture(P_n) \text{ for } n \in S \tag{6.5}$$

Where: *MinMoisture(P_n)* is the minimum moisture requirement of the species to which plant P_n belongs; S is the set of plants whose roots intersect the given grid cell.

If $M_{requested}$ is less than $M_{available}$, the moisture is deemed sufficient and the amount allocated to each plant is calculated as described in Equation 6.6. Intuitively, this equation allocates each plant with the minimum amount of humidity it requires to survive plus the resulting overflow. Note that in this equation, the overflow is not distributed amongst plants of the cell but rather allocated to each plant. For the same reason as to why all plants are allocated $M_{available}$ when it is deemed abundant, it is to prevent situation where unsuited plant species are able to grow because the moisture allocated to it is low simply because it is distributed to other plants.

$$\begin{aligned} M_{allocated}(P_n) &= MinMoisture(P_n) + OverFlow \\ OverFlow &= M_{available} - \sum MinMoisture(P_n) \text{ for } n \in S \end{aligned} \tag{6.6}$$

Where: *$M_{allocated}(P_n)$* is the humidity allocated to plant P_n;

If $M_{requested}$ is more than $M_{available}$, however, the humidity is deemed insufficient and the allocation follows algorithm 2 which prioritises water distribution to the more vigorous plants. The vigour of a plant is estimated based on its root size. This ensures stronger plants have better access to water than smaller, weaker ones.

The overall moisture allocated to P_n is calculated using Equation 6.7. Intuitively, it is simply the average of all moisture allocated to it within all cells of S.

$$M_n = \frac{\sum M_{allocated}(C_x) \text{ for } x \in S}{|S|} \tag{6.7}$$

Algorithm 2 *Algorithm to distribute soil moisture within a cell when the quantity is insufficient.*

Require: Initialize $S_{remaining}$ as the total moisture available in the cell.

Require: Initialize S as the set of all plants whose roots intersect the cell, sorted in decreasing order of root size

TotalRootSize = 0
for P_n in S **do**
    TotalRootSize += RootSize(P_n)
end for
for P_n in S **do**
    Vigour = $\frac{RootSize(P_n)}{\sum RootSize(P_x) \text{ for } x \in S}$
    $M_{allocated}(P_n) = min(MinMoisture(P_n), Vigour \times M_{remaining})$
    $M_{remaining}$ -= $M_{allocated}$
    Remove P_n from the set S
end for

Where:M_n is the moisture allocated to plant P_n; $M_{allocated}(C_n)$ is the moisture allocated to plant P_n in grid cell C_n; $\mid S \mid$ is the number of cells in the set S.

6.3.3 Illumination Distribution

Photosynthesis is an essential part of plant development as it permits the creation of fresh matter and, therefore, growth [SSBR01]. Species that are heavily dependent on illumination will often grow large canopies to maximize the leaf coverage area and therefore photosynthesis potential. These large canopies also limit the illumination available in the area underneath the canopy, limiting plant development. To model this, available illumination is calculated for each grid cell based on the height of the plants with canopies intersecting the given cell, as outlined in Equation 6.8. Intuitively, if all plants present in the given cell are canopy-free, the equation allocates them all the available illumination. If not, the equation allocates illumination only to the tallest canopy plant. A canopy-free plant is one which grows more in height than width and for which shade projection can be ignored (e.g grass, cacti, etc.). Note that this is a simplification as some light should still pass through the canopy and the shade projected by the canopy does not always fall directly below but varies throughout the day and the year. A much more detailed approach is taken by Soler et al. [SSBR01], who model light transmittance through the canopy based on plant geometry. This detailed approach is ill-suited here, however, as the growth of a large set of plants needs to be simulated simultaneously. A possible extension to this work would be to associate with each plant species a canopy density parameter, which affects the quantity of light which can pass through its canopy.

$$Illumination(C_{xy}, P_n) = \begin{cases} C_{illumination}, & \text{if } CanopyWidth(P) = 0 for P \in S \\ C_{illumination}, & \text{if } Height(P_n) > height(P) for P \in S : P \neq P_n \\ 0, & \text{otherwise} \end{cases} \tag{6.8}$$

Where: *Illumination*(C_{xy}, P_n) is the illumination allocated to plant P_n whose canopy overlaps grid cell C_{xy}; $C_{illumination}$ is the available illumination for the given month (equal for all cells);*CanopyWidth(P)* is the canopy width of plant P; *Height(P)* is the height of plant P; S is the set of plants whose canopy intersects with the given grid cell C_{xy}.

Calculating the illumination allocated to a plant P_n is identical to calculating the humidity allocated (see Equation 6.7) but the cells considered are those which the plants canopy intersects (and not its roots). Intuitively, the illumination allocated to a given plant is simply the average of the illumination allocated to it in all grid cells its canopy intersect.

By calculating the illumination separately for each cell covered by a plants canopy and then taking the average as the aggregate illumination, it is possible to model partial shade. For example, if half the grid cells covered by a plants canopy are shaded (zero illumination) and the other half receive ten hours of daily illumination, the aggregate would be five hours.

6.3.4 Plant Strength Calculation

Given the humidity and illumination allocated to a given plant P, the temperature, the slope and the age of P, it is possible to calculate its overall strength (vigour), which is subsequently used as a representation of the plants health and directly affects its growth and survival.
The overall strength, of plant P, is taken as the minimum of S_{slope}, S_{age}, $S_{temperature}$, $S_{illumination}$ and $S_{humidity}$, which represent the strength of P in terms of the slope, its age, the temperature, the allocated illumination and humidity, respectively. The minimum is taken rather than the average as the strength of a plant depends heavily which resource is limiting. For example, if a plant is struggling due to a lack of daily illumination, improving the allocated water would not have a big impact on its overall health.
To calculate the individual strength values, in the range [-100,100], a graph is plotted as outlined in Figures 6.2 and 6.3 for each species. These graphs are generate for each resource based on the associated properties (see Section 6.1). Using these, it is possible to calculate the plants strength in terms of slope, age, temperature, illumination and humidity. When a plant has a negative strength it is deemed in *survival*. When in this state, it does not grow and is susceptible

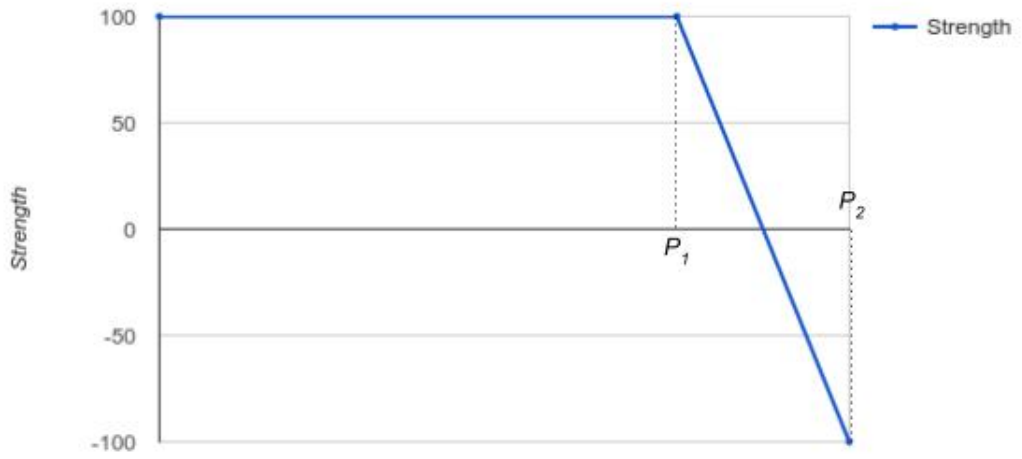

Figure 6.2: *Graph used to calculate the slope and age strength of a given plant instance where:* $\boldsymbol{P}_1$ *represents the value of start of decline and* $\boldsymbol{P}_2$ *is the maximum configured for the given species.*

to be killed off.

6.3.5 Plant Growth

In the simulation, each plant P attempts to grow its roots, its canopy and its height on a monthly basis. Each species has a maximum monthly root growth, canopy growth and height growth which are calculated as outlined in Equation 6.9. The maximum height, canopy and root size, along with the species age-based start of decline (see Section 6.1) are used to calculate the amount it must grow each month to reach these maximums by start of decline. Note that this is a simplification as, in reality, plant growth is non-linear as growth slows with increasing size [PMV+12].

$$MaxGrowth(S) = \frac{Max(S)}{Age_{sod}(S)} \tag{6.9}$$

Where:*MaxGrowth(S)* is the maximum monthly root, canopy or height growth of species S; *Max(S)* is the maximum root size, canopy size or height configured for species S; $Age_{sod}(S)$ is the age of start of decline configured for species S.

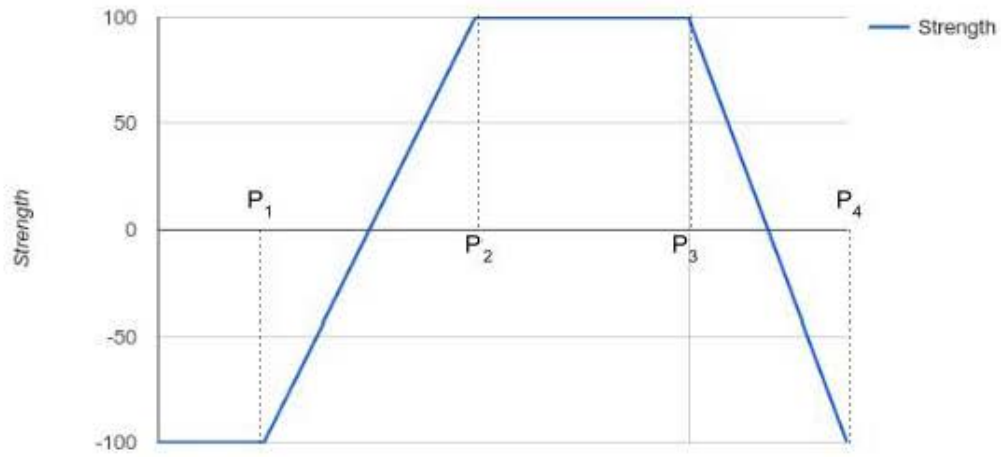

Figure 6.3: *Graph used to calculate the temperature, illumination and humidity strength of a given plant instance where:* $\boldsymbol{P}_1$ *and* $\boldsymbol{P}_4$ *are the minimum and maximum and* $\boldsymbol{P}_2$ *and* $\boldsymbol{P}_3$ *form the prime range configured for the given species.*

The actual root growth, canopy growth and height growth of a plant is directly dependent on its strength (see Section 6.3.4), however, and is calculated using Equation 6.10. This equation only permits growth if the plants strength is positive as it is otherwise deemed too weak to grow and in a state of *survival*. If the strength is positive, the growth is proportional to the plants strength. The maximum growth is therefore only achieved if the plant is at full strength.

$$Growth(P, S) = max(0, Strength(P) \times MaxGrowth(S) \tag{6.10}$$

Where: *Growth(S)* is the monthly root, canopy or height growth of plant P of species S; *Strength(P)* is the current strength of P; *MaxGrowth(S)* is the maximum monthly root, canopy or height growth calculated for species S (see Equation 6.9).

6.3.6 Plant Death

In order for the simulation to be accurate, it is necessary to model plant death. This can be caused by ageing, the slope being ill-suited or resources being inadequate. On a monthly basis, the probability of death of each plant is calculated based on its strength using Equation 6.11 and the plant killed off with the given probability. This equation permits plants to be killed-off only when in a survival state (i.e the strength is negative). If this is the case, the probability of death is proportional to the absolute value of the strength.

$$Probability_{death}(P) = max(0, \frac{-1 \times Strength(P) + counter}{100}) \tag{6.11}$$

Where:*Probabilitydeath(P)* is the probability of death of plant P;*counter* is a value which increases each month the plants strength is negative, and resets to zero when it becomes positive. This prevents plants from surviving in a survival state for too long.

6.3.7 Spawning Plants

In nature, the spawning of new plants ensures species *succession* and *propagation*. In order to accurately model the evolution of an ecosystem it is essential to replicate this spawning mechanism. To do so, seeds are produced annually for each species and are positioned either randomly or at predefined positions. The number of seeds that are produced for a given species is determined by the species configured *annual seed count*. Different seeding mechanisms are used in the simulator depending on the current state of the simulation, as discussed below.

To ensure species propagation, when plants of the given species are already present in the simulation window, they are used to determine the location for new plant instances. To do so, n of these plants are selected at random and seeds placed at random within an annular radius r of each. The value of n is the *annual seed count* configured for the current specie. The value of r is the configured *maximum seeding distance* of the specie. Note that a single plant can be used to spawn multiple seeds if n is greater than the number of plants of the given species present in the simulation.
This technique is effective in ensuring *propagation* until the number of plant instances present far outweighs the number of seeds, at which point, the *propagation* potential decreases. This is because, as the selection pool for the random seeding plants increases in size, the probability of selecting a seeding plant at a location which will permit propagation decreases. For example, if there are 100 plants of species s within a 2 metre square window and 50 are selected for seeding, there is a high probability that a number of them will be selected at the extremities and, therefore, propagate the species further. If there are 10000 plants of the given species, however, the probability of selecting extremity plants and is very low. To overcome this and ensure the initial seeding plants that are selected span a wide area, the simulation window is split into equally sized cells and the seeding plants selected individually from each.

If no instance of the given species is present in the simulation (i.e it is the first month) and therefore no seeding plants can be used, the seeds are placed at random within the simulation window.

A species is deemed shade-loving if its configured minimum daily illumination is zero. Such species thrive in the undergrowth of other plants. Spawning shade-loving species in the same way as other plants would drastically limit their chance of survival because the probability of a random seed location falling in the canopy of an existing plant is very low. For this reason, when there are no instances of the given shade-loving species present in the simulation window, the seed locations are located at random under the canopy of existing plants. If instances of the plant species are already present, however, they are used to propagate the seeds, just like with other, non shade-loving, plant species.

6.3.8 Algorithm Complexity

On a monthly basis, the algorithm must iterate through all plant instances twice for the following purpose: The first pass is to calculate the resources distributed to each plant instance depending on resources available, its associated strength, growth potential and probability of death. The second pass is to actually perform plant growth and kill off weak plants. Note that it is not possible to perform growth and death during the first pass as, doing so will subsequently effect the strength and probability of death of other plants. In other words, doing so would make the ecosimulator produce different results depending on the order in which the plants are iterated over.

Given this information, the complexity of this algorithm is $\mathbf{m} \times \mathbf{2n}$ where: $\mathbf{n}$ is the number of plants present in the simulation at any given month and $\mathbf{m}$ is the number of months for which the simulation must be run.

6.3.9 Performance

The number of plants present in the simulation will heavily influence its performance as the strength of each plant needs to be recalculated on a monthly basis. As can be seen in Figure 6.4, with a linear relationship between plant count and memory usage, this also has a big impact on memory. With a maximum of just under 17 megabytes for over 46 thousand plant instances, this remains manageable, however.

To test the influence of plant count on simulation execution time, a simulation is run with a single species of plant and the monthly processing time analysed alongside the number of plants present. The plant used is grass as is has no canopy and very minimal root coverage, therefore permitting a large number of instances to grow simultaneously (see appendix B for properties of the specie). The resources were set to be optimal for maximizing plant count and minimizing intra-plant competition. The simulation is started with only a single instance and, as the simulation progresses and seeding is performed, the number of instances increases. The results are summarized in Figure 6.5 and show that the processing time increases linearly with plant count. The jump at the 65 thousand plant mark occurred repeatedly over multiple tests.

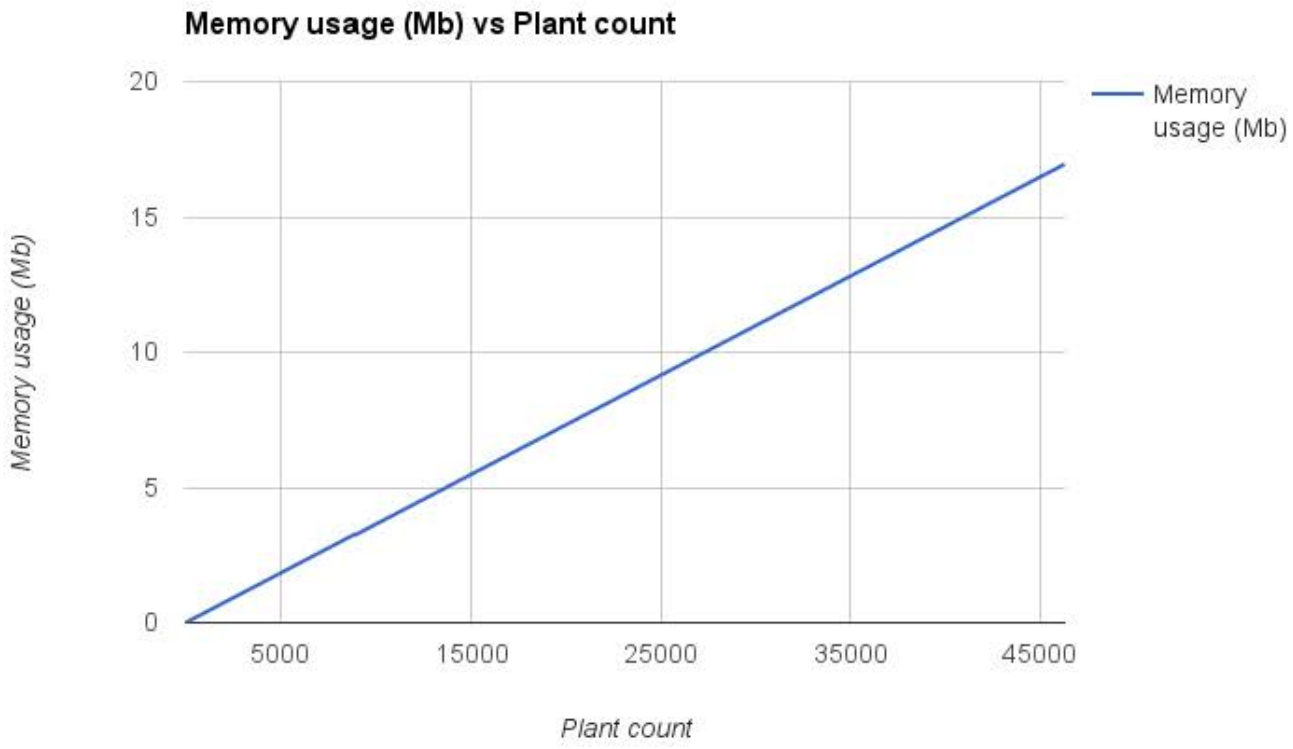

Figure 6.4: *Memory usage based on plant count. It shows a linear relationship between memory usage and plant count*

Although just a hypothesise, the most likely cause is that it is a tipping point at which page swapping starts to occur due to saturated RAM.

Another property that heavily impacts performance is the root and canopy growth. As roots and canopies grow, they will cover more grid cells of the simulation window, and more calculations will be required per individual cell. To analyse the impact of plant growth, a base species S_{base} is created with a given root and canopy growth rate. Then, two species S_{X2} and S_{X3} are created with identical properties to S_{base} but with twice and thrice the growth rates, respectively (see appendix B for species details). Separate simulations are run with each species but with identical available resources and, on a monthly basis, the number of plants present in the simulation, along with the monthly processing time, are analysed. Given this information, it is possible to track the average monthly execution time per plant throughout the simulation. It is important to normalise this based on the number of plants as faster growing plants will naturally reduce the total plant count for the simple reason that they will require and be able to access resources from a larger number of grid cells. As can be seen in the results plotted

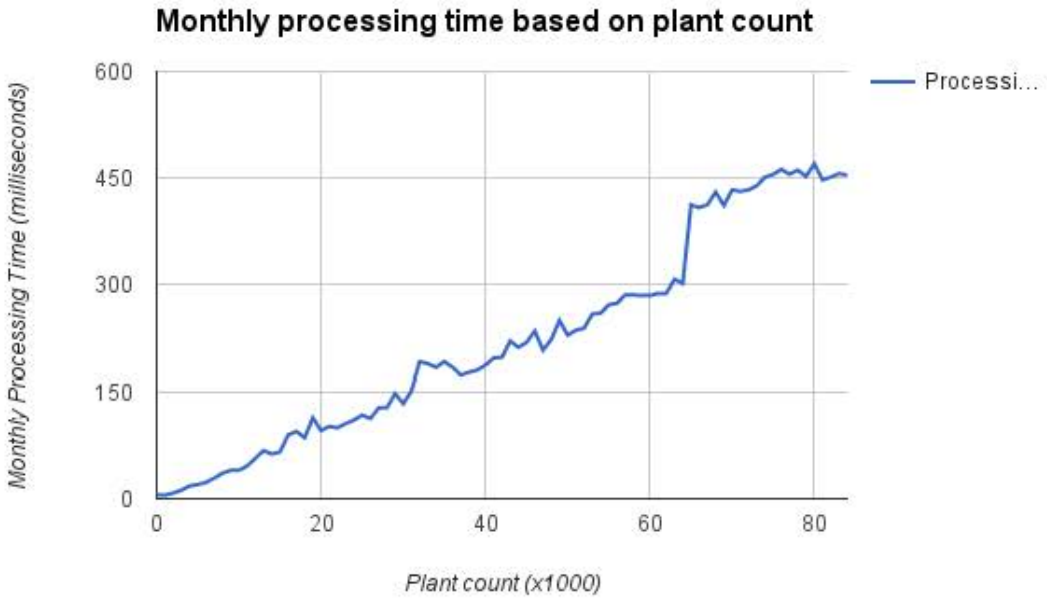

Figure 6.5: *Processing time based on plant count. Total simulation time for 100 years: 271 seconds*

in Figure 6.6, the processing times are similar to begin with and then increase proportionally to the species growth rate. For the fastest growing plant specie, S_{X3}, it took 166 seconds to simulate one hundred years.

6.3.10 Results

To test the resulting spatial distribution of plant communities in their work, Lane and Przemyslaw [LP02] attempt to reproduce three important properties of nature: *Self-thinning*, *succession* and *propagation*. To test the ecosystem simulator, we employ the same methodology as Lane and and Przemyslaw [LP02] and attempt to reproduce these core properties of nature. Other tests are also performed to ensure plant instances thrive better in environments suited to their individual resource requirements.

SELF-THINNING TEST

As plants grow, their resource requirements increase and, as a direct consequence, inter-plant competition for resources increases. Eventually, the competition becomes too intense and resources too scarce, leading to more vigorous plants starving smaller plants. At this point, *self-thinning* begins and plant densities decrease [LP02].
To test whether self-thinning is successfully modelled in the ecosystem simulator, three sim-

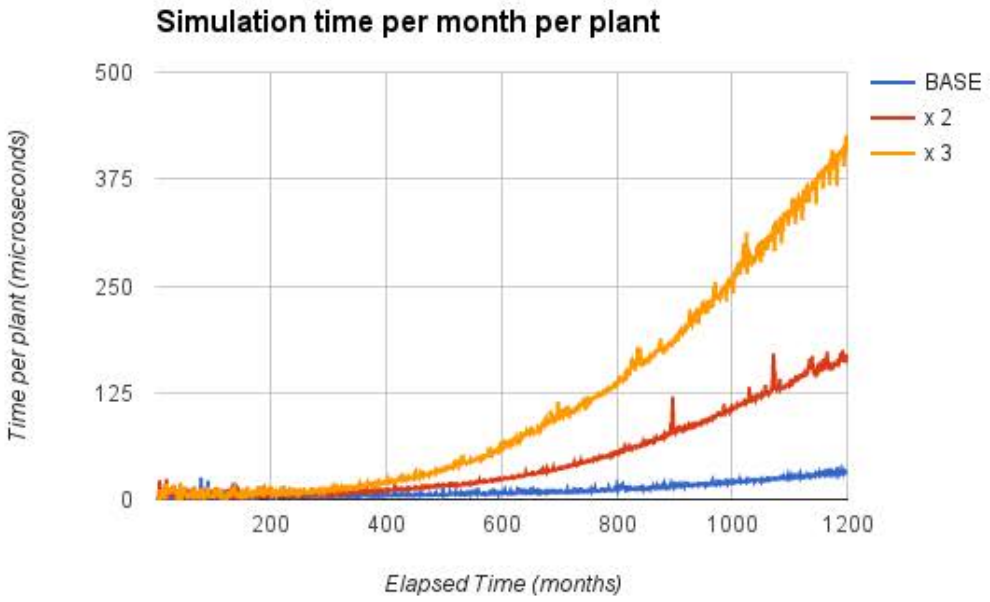

Figure 6.6: *Evolution of the monthly processing time normalised based on plant count. The processing time increases as the plants grow larger since they cover more grid cells. Total simulation time for one hundred years: 49 seconds for* S_{base}*, 122 seconds for* S_{X2} *and 166 seconds for* S_{X3}

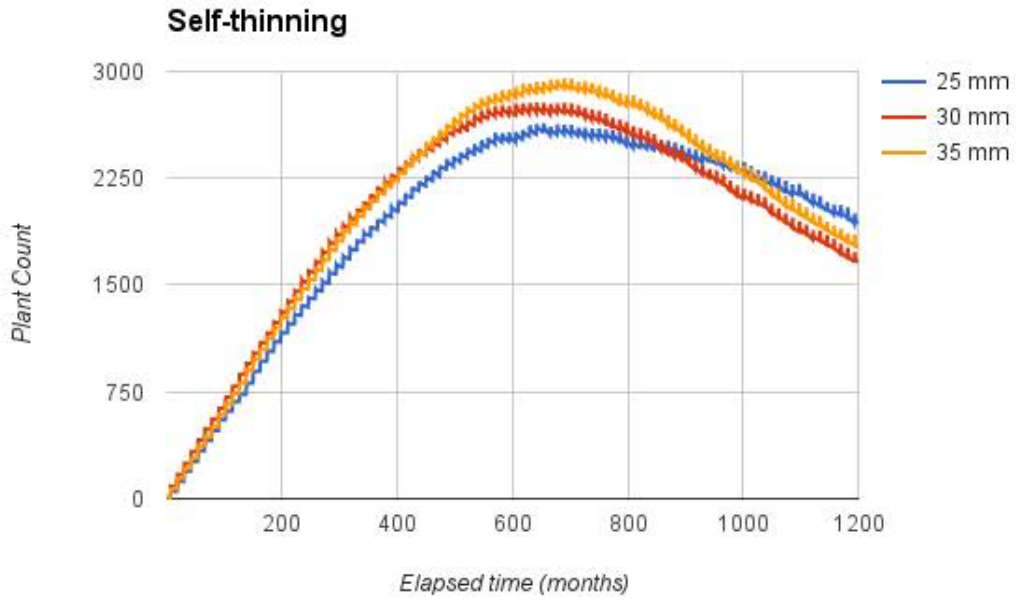

Figure 6.7: *Self thinning test: plant count tracked throughout three separate simulations differing only in available humidity. For all three runs the plant density reaches a maximum tipping-point after which plant density reduces. Note that the more available moisture there is, the more severe the slope of descent is following the tipping point. This is because, with more soil moisture available, plants thrive and reach greater sizes. Because of this increased spatial coverage, the killing off of smaller plants is more severe. So, although the plant count is smaller by the end of the simulations when there is more humidity, the average plant size is larger.*

ulations are run differing only in the configured soil moisture and the plant count tracked throughout. As described previously, self-thinning occurs because of insufficient resources. By modifying only available humidity in each simulation, its affect on self-thinning becomes apparent. As can be seen in the results summarized in Figure 6.7, the plant count increases at first, reaches a maximum and decreases thereafter. This is the expected behaviour of self-thinning. Furthermore, it is apparent that the maximum plant count increases with the humidity available, therefore showing that the tipping point depends on available resources.

SUCCESSION TEST

Given plant species A with a fast growth rate and species B with a slower growth rate but higher shade tolerance. At first, the faster growing species A will dominate and flourish but, with time, the slower growing, but more shade-tolerant species B will flourish and dominate. This is the *succession* property. To test *succession* in the ecosystem simulator, two plant species

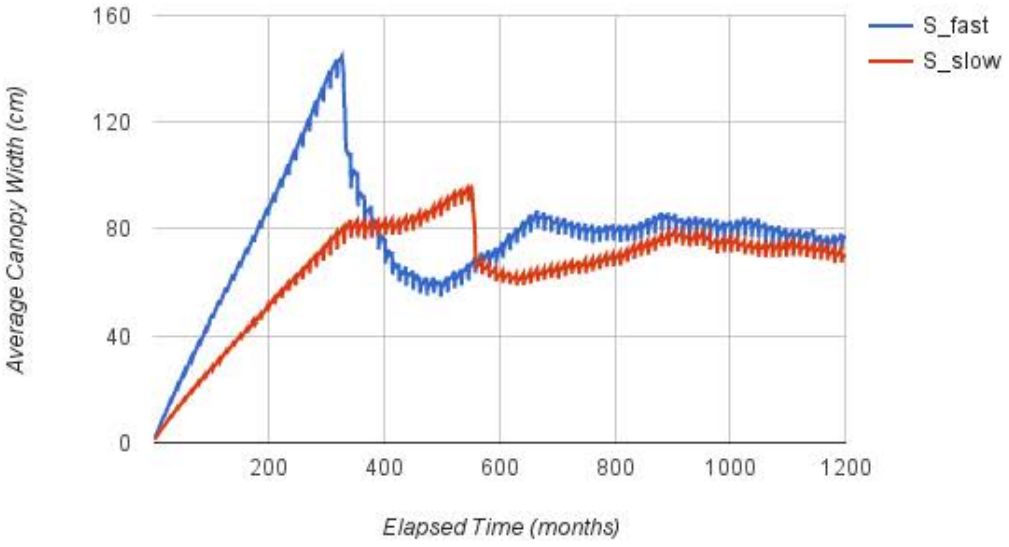

Figure 6.8: *Succession Test: Average size of the slow growing S_{slow} (red) and fast growing S_{fast} (blue) throughout a simulation run in optimal conditions. Note that the plant count drops severely at the 300 and 500 mark for S_{fast} and S_{slow} respectively as resources are configured such that conditions are ideal for these species. This leads to a large quantity of them dying of age and, because the difference between start of decline and maximum age configured for these species is very small, this decrease is very sharp.*

S_{fast} and S_{slow} are created differing only in their growth rate and illumination properties (see appendix B for details) and a simulation run with these two species under optimal conditions. During the simulation, the appearance and average size of the two plant species are monitored to determine the dominating specie. The results are analysed and illustrated in Figure 6.8. A snapshot of the simulation window is taken at ten year intervals and displayed in Figure 6.9. Both these figures show that S_{fast} dominates at first (300 months in) followed by S_{slow} (500 months in). A balance is found thereafter.

PROPAGATION TEST

The *propagation* property simply states that plants *propagate* in clusters surrounding the seed plants. To ensure propagation is modelled, a simulation is run with a single starting grass seed (see appendix B for species details) and its evolution tracked throughout. Figure 6.10 shows that iterative propagation through annual seeding enables a single seed plant to colonize the entirety of the terrain. Note that, although it does show propagation is reproduced, it is

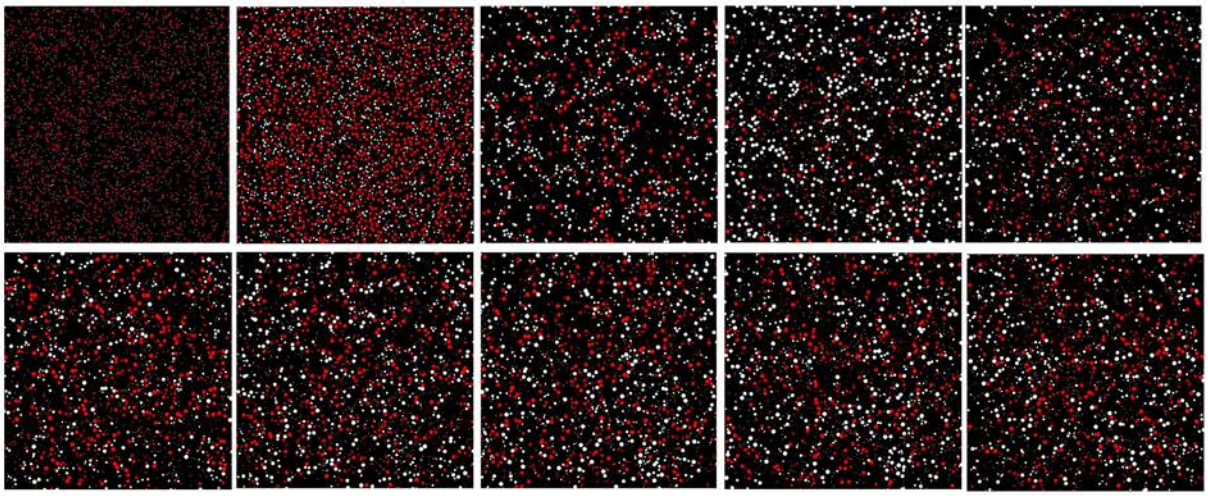

Figure 6.9: *Succession Test: Appearance of the slow growing S_{slow} (white) and fast growing S_{fast} (red) at different times during the simulation. From left-to-right, top-top-bottom: 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 years.*

unrealistically slow. This is caused by the seeding algorithm used. As discussed in Section 6.3.7, in order to promote propagation, the simulation window is split into grids and seeding plants selected separately from each cell. Although this increases the spatial coverage of the seeding plants, it still fails to propagate effectively when the number of grid cells in which the given species appears is large as the probability of selecting a grid cell from the edge (which would lead to seeding) decreases with the grid coverage of the given specie. A worthwhile extension to this work would be to implement the ability to locate border grid cells and use them more extensively during seeding. This could be done by, for example, sorting the grid cells in order of the species count they contain as border cells would naturally be less dense.

VARYING RESOURCE TEST

To ensure a given plant species thrives better when the environment is more suitable, multiple simulations are run with a single species S_{base} (see appendix B for species properties) varying only in available humidity. Throughout the simulation, the average plant canopy width is tracked to monitor the strength of the plants. As can be seen by the results plotted in Figure 6.11, plants thrive better in environments better suited to their resource requirements.

SHADE TEST

Plants that are heavily dependent on illumination struggle to grow in areas shaded by the canopy of larger plants. To ensure this is modelled in the ecosystem simulator, a simulation is run with two species: $S_{smallroots}$ and grass (see appendix B for species details). $S_{smallroots}$ is a

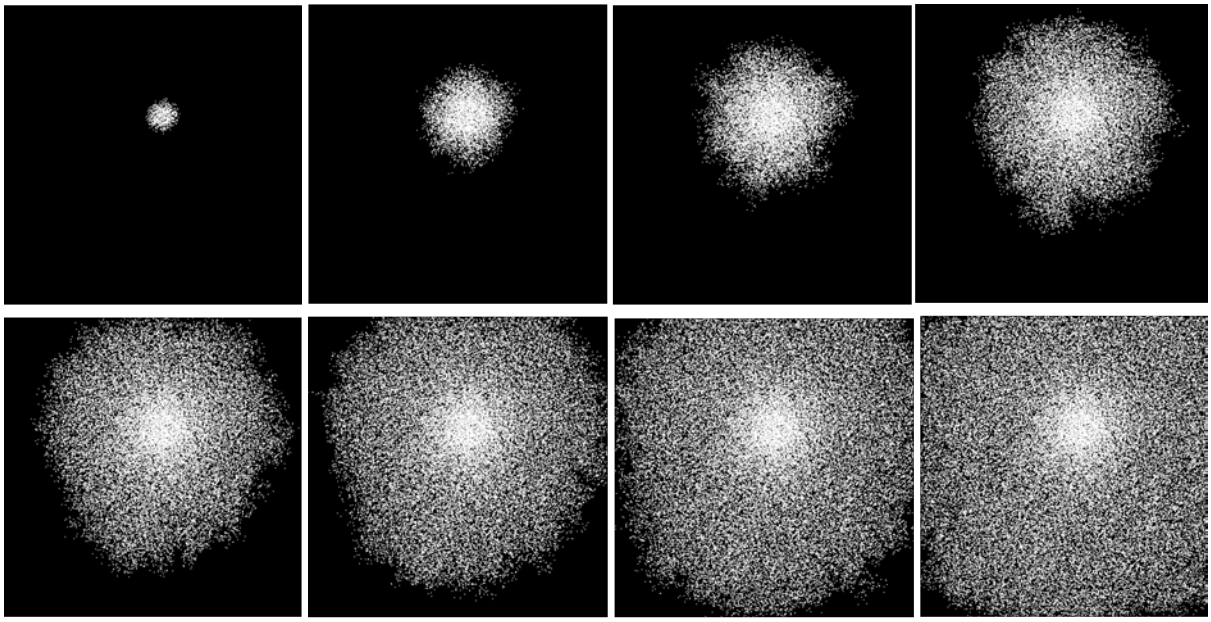

Figure 6.10: *Propagation Test: Evolution through time of a simulation starting from a single seed plant of grass. From left-to-right, top-to-bottom: 2, 10, 20, 30, 40, 50, 60, 70 years in.*

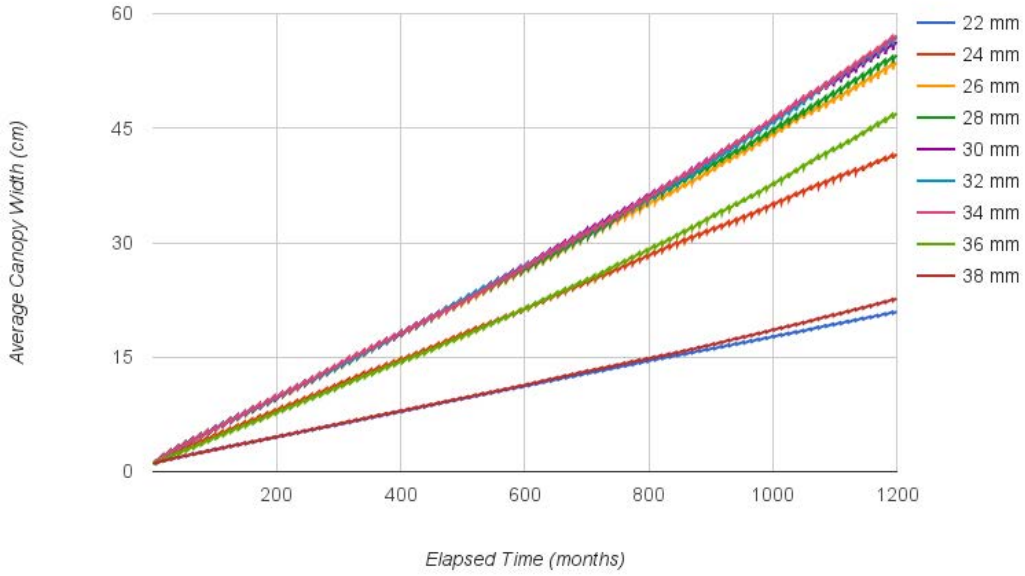

Figure 6.11: *Varying resource test: Average canopy width throughout simulations varying only in available moisture and with only plant species S_{base}. It shows that the average canopy width is low when the configured humidity is outside the species optimal humidity range (22mm and 38mm), improves as it approaches optimal range (24mm and 36mm) and reaches its peak when the humidity is within the optimal range (26mmn, 28mm, 30mm, 32mm and 34mm).*

custom species created for the purpose of this test which has a very small root growth value. This is important so as to focus on the effects of illumination and minimize the influence of drought. Figure 6.12, which illustrates the state of the simulation after ten years, shows the grass struggling to grow in areas directly below the canopies of $S_{smallroots}$.

SHADE-LOVING TEST

As discussed in 6.3.7, species which strive in shaded areas are deemed shade-loving. The shade can be caused by the terrain relief or by the shadow cast by the canopy of taller plants. To test whether the ecosystem simulator successfully caters for such plant species, a simulation identical to the shade test is run but with shade-loving species $S_{shadeloving}$ added (see appendix B for species details). As seen by the snapshot of the simulation after fifteen years illustrated in Figure 6.13, instances of $S_{shadeloving}$ only appear in areas covered by the canopies of $S_{smallroots}$. The number of shade-loving plants at the end of the simulation is reasonably small. This is caused by the plant strength calculation algorithm and the technique employed for plant propagation. At first, shade-loving plants are unable to survive in the simulation as there are no plants large enough to cast shade. On an annual basis, the propagation algorithm attempts to spawn new shade-loving plants at random locations in the simulation window. It can take a while for a randomly selected location to be under the canopy of an existing plant and therefore permit it's survival. Another issue faced by the shade-loving species is that there size is very small when they first spawn and, as they are guaranteed to be within the neighbourhood of existing larger plants (which shade it), their chance of survival is very limited if soil moisture is scarce. This could be improved in future work by implementing the soil in layers as described in Section 6.3.2

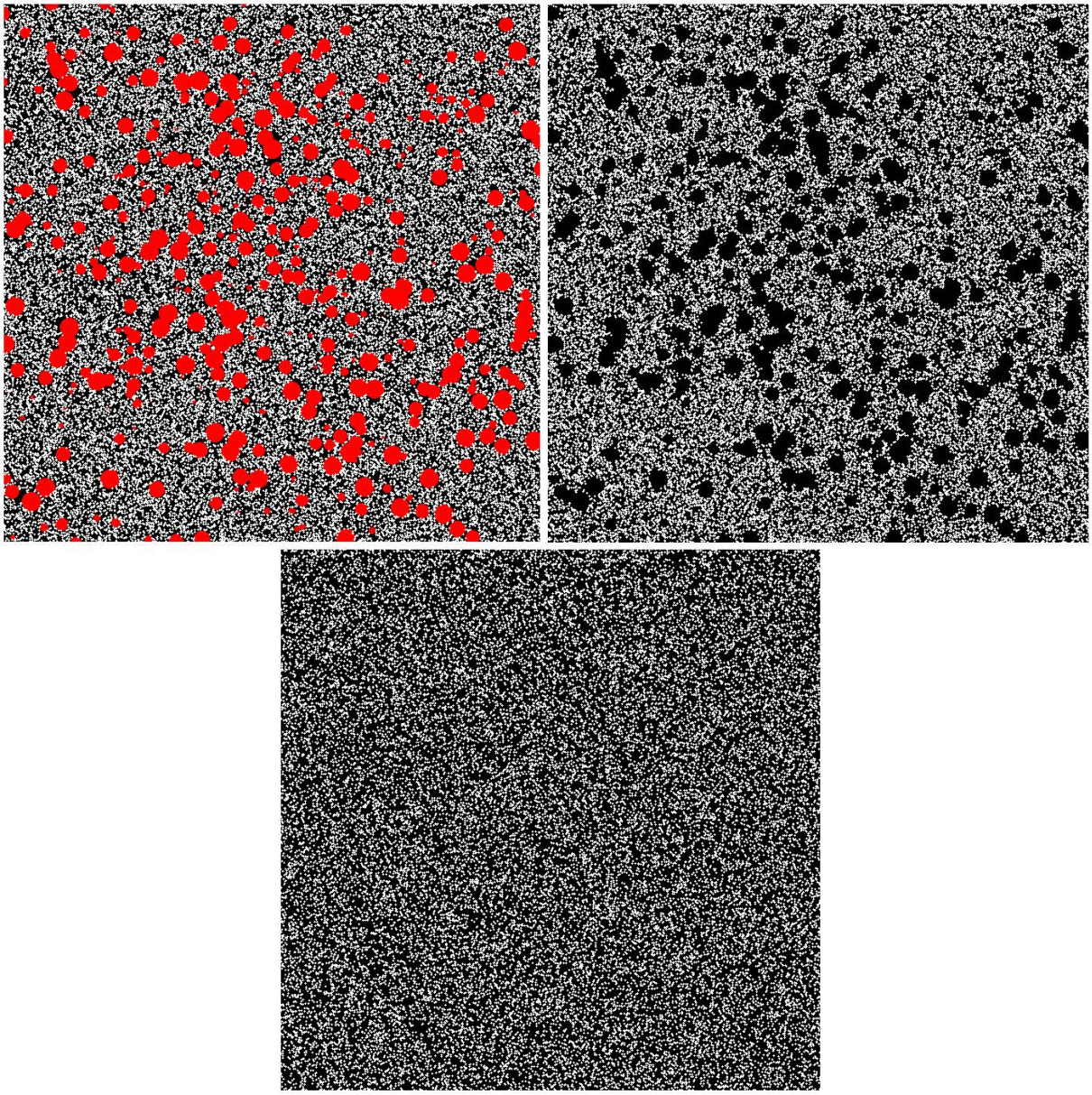

Figure 6.12: *Shade Test: Results of a simulation run with $S_{smallroots}$ (red), grass (white) after 10 years. Top-left: Rendered with both species. Top-right: Only grass rendered in order to clearly visualise the empty areas at the exact locations the canopies of $S_{smallroots}$ appear. Bottom: The same simulation run with only grass, clearly showing the empty areas disappear.*

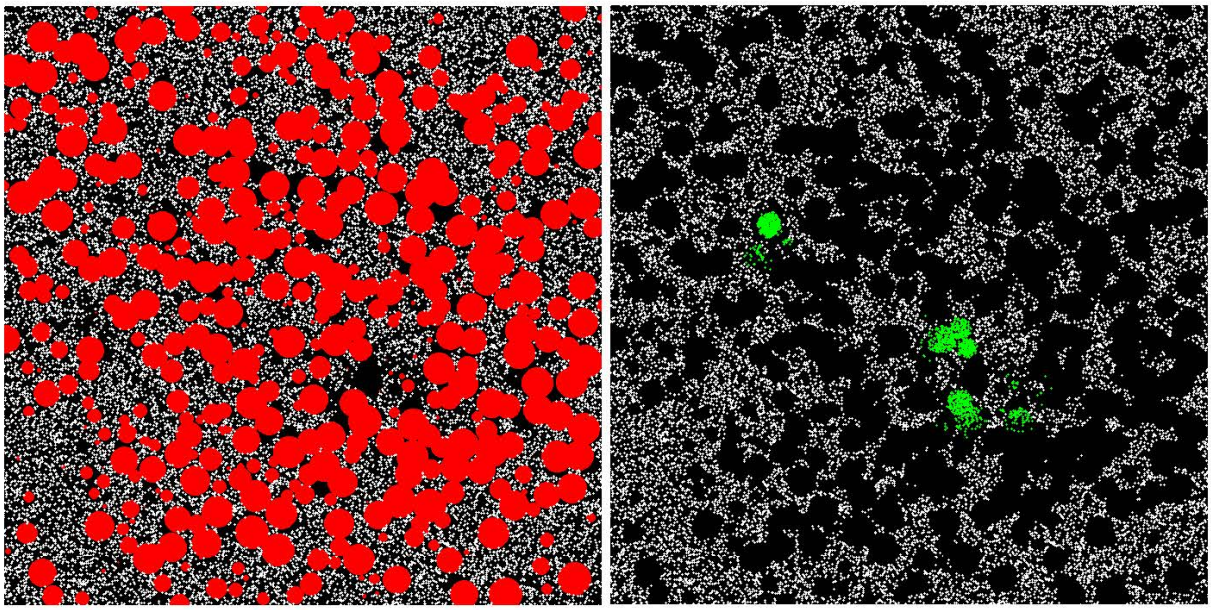

Figure 6.13: *Shade Loving Test: Simulation with $S_{smallroots}$ (red), grass (white) and $S_{shadeloving}$ (green) after 15 years. Left: All species rendered. Right: All except $S_{smallroots}$ rendered. It shows clearly that the only instances of $S_{shadeloving}$ which survive are those under the canopies of $S_{smallroots}$ (red).*

6.4 Plant Distribution Analysis and Reproduction

As mentioned previously, running a simulation using the ecosystem simulator to generate valid plant distributions can be a lengthy process (see Section 6.3.9). This processing time depends on the plant count and the resolution of the simulation window. The simulation illustrated in Figure 6.5, for example, took four and a half minutes to complete.

The area to be covered with vegetation on the terrain, and for which valid plant distributions therefore need to be created, can potentially be much larger than the hundred by hundred metre simulation window used by the ecosystem simulator. There are three obvious ways to extend the ecosystem simulator for this: *Setting the simulation window to the area which must be covered*, *decreasing the resolution of the simulation window* and by *repeating the output of the hundred by hundred metre distribution (tiling)*. Each come with major setbacks: *increasing the simulation* window will further increase the processing time, *decreasing the resolution* will impact the resulting realism and *tiling* will create repetitive vegetation, also impacting the resulting realism.

In this work, *radial distribution analysis* is used [EVC+15] to analyse the statistical characteristics of the vegetation distribution generated by the ecosystem simulator. Some customisations, discussed below, are made to the core algorithm, however, to better suit the requirements of our system (i.e plant distribution analysis).
This analysis data is then used to generate plausible distributions on much larger areas that respect the characteristics of these input exemplars. This method proves to be both efficient, as statistical reproduction runs much faster than the ecosystem simulator, and realistic as it increases the area for which a plausible, non-repeating distribution is created, therefore limiting any tiling on the final terrain.

6.4.1 Radial Distribution Analysis

Radial distribution analysis, as described in detail in Section 2.2.2, is performed on the output of the ecosystem simulator to derive its core characteristics. When performing the analysis on the vegetation distribution output of the ecosystem simulator, each plant instance acts as a single point and the different species represent the individual categories. Customizations to the generic radial distribution analysis algorithm are performed, as discussed below, to better suit the purpose of analysing plant distributions.

Generating the category hierarchy During reproduction, distributions for each category are created sequentially, in order of priority. Once a valid distribution is created for a given category, it is static and **does not** change whilst points of other categories are being plotted.

For this reason, the category hierarchy plays a vital role and has a big impact on the final distribution. Because taller plants will potentially have a canopy that shades and influences the position of smaller plants, it is important these be generated first during reproduction. For this reason, the hierarchy is generated based on the average height of the represented plant species, in descending order.

Category Dependency Analysis Shade-loving plants will appear under the the shaded canopies of taller plants (see Section 6.3.10). To cater for this during radial distribution analysis, a new *negative histogram bin* is created for plants that appear within the canopy radius of others.
During reproduction, taller plants will be placed first because they are classed higher in the category hierarchy. This does not guarantee all shade-loving plants will be placed in the shaded canopy of other plants, however, because if a shade-loving plant is placed at a distance larger than R_{max} from any other plant instance, it is attributed a strength of one by default and is therefore deemed valid. It is essential to attribute a strength of one in such conditions to permit plant propagation. A solution would be to set R_{max} large enough to cover the entire simulation window. This would drastically increase the computational requirements, however, as it would drastically increase the number of destination points that need to be analysed for each source point during the analysis stage. Another solution is used here: When the pairwise histograms have been generated for a given category A, they are subsequently analysed to check whether all instances appear within the *negative-bin* of the other category. The species A is deemed **dependent** on all categories for which this is true and, during reproduction, will have to be placed within the radius of one of them for the distribution to be valid.

Plant-size Analysis in addition to location, plant size is an important output property of the ecosystem simulator. In order to reproduce appropriately sized plants, this must also be analysed. To do so, the *minimum* and *maximum* canopy radius and height for each category are analysed. When placing plants of the given category during the reproduction phase, its size is selected at random within the range [*minimum*,*maximum*].
A valuable addition to this work, which could improve the realism of the results, would be to analyse and store size information in more detail. For example, separate size histograms could be generated for each plant species which tracks the ratio of plant instances who's size are within each size bin. This information would subsequently be used to generate plant sizes matching the input exemplar more closely.

Configuration Parameters The *radial distribution analysis* requires the following configuration parameters: R_{min}, R_{max} and *bin-size*. Details on each parameter can be found in Section

2.2.2.

Increasing the analysis range [R_{min}, R_{max}] and decreasing the *bin-size* will impact performance but potentially increase the accuracy of the analysis. Finding optimal values for these parameters depends on the properties of the points being analysed. A large analysis range is unnecessary as the impact a plant has on its surrounding is finite. This impact radius varies, however, and is dependent on species size. In order to cater for different species of different sizes and therefore with different impact radii, R_{max} is dynamic and limited to **two metres** beyond the extremity of the plants canopy.
The bin-size does not influence performance as severely as the analysis range, however, as it has no impact on the number of points that need to be processed but only the bin in which they will influence. Smaller bin sizes will result in fewer points being processed per bin and, therefore, a less accurate representation of the distribution variation with distance. Because smaller bins will result in a smaller number of points, also, the analysis will be more sensitive to noise. A default bin size of **twenty centimetres** is used as it strikes a good balance between accuracy and point count per bin in the majority of test scenarios. All these parameters can easily be changed, however, if the generated analysis data is considered ill-suited by the user.

As an extension to this work, these parameters could be configured automatically by doing a very basic first-pass analysis beforehand. For example, the analysis range and bin size can change dynamically depending on plant density.

Performance In order to generate the necessary analysis data, each point (plant) must be iterated over and the distance measured from it to all other points within a radius of R_{max}. As a consequence, the analysis time is directly correlated to plant density and, therefore, plant count within the hundred metre analysis window. To determine the correlation between plant density and analysis time, test distributions are generated of various densities using the ecosystem simulator. These are subsequently analysed and the execution time, measured. Because the number of histograms depends on the number of categories, it would be easy to assume that the processing time is correlated to the category count. This is **not** the case, however, as it does not affect the aggregate point count that needs to be processed. Only point density influences analysis performance and to demonstrate this, various plant densities are generated containing one, two and three distinct categories. The results, plotted in Figure 6.14, indicate an exponential correlation between plant count and processing time and even point to a quicker analysis time when points are split into multiple categories. The processing is faster when split into multiple categories because multi-threading is used to generate the different pair correlation histograms in parallel. Although the correlation is exponential, the most extreme scenario (over

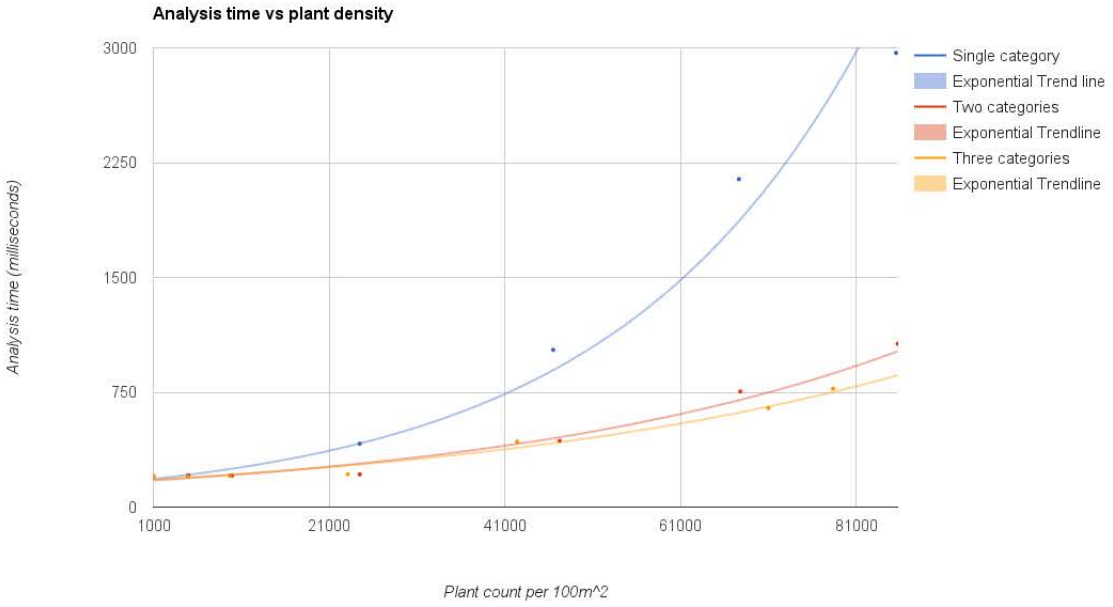

Figure 6.14: *Distribution analysis time based on aggregate plant density for single category (blue), two categories (red) and three categories (yellow).*

ninety thousand points in a single category) is processed in a manageable time of just under two seconds.

6.4.2 Radial Distribution Reproduction

The purpose of the analysed radial distribution data is to later use it to reproduce distributions on larger scales which match the characteristics of the original input exemplars from the ecosystem simulator. To do so, the same reproduction technique described in Section 2.2.2 is employed with slight nuances described below. Pseudocode in Figure 3 provides a summary of the core algorithm employed.

Matched Density Initialization Rather than employ a birth-and-death technique like that described in Section 2.2.2 and employed in the work by Emilien et al. [EVC+15], where a point is added, the aggregate strength of the distribution calculated and the new point accepted with a calculated probability, matched-density initialization is employed. This technique first initializes the distribution with a fixed number of points so that the point density matches that of the input exemplar. The only requirement is for the aggregate strength of the distribution to be non-zero. In other words, the distribution does not need to be strongly matched,

Algorithm 3 *Radial distribution reproduction algorithm.*

Require: Initialize $AllCategories$ as the set of all categories, sorted in decrementing order of their rank.

Require: Initialize $AllPoints_c$ as the set of all points of category c.

Require: $movePoint(c)$ is a function which takes a point c and moves it to a new random location. The new location for the given point is accepted or rejected based on the new strength of the distribution.

Require: matchedDensityInitialize(c) is a function which initializes points of category c to the correct density such as their distribution strength is non-zero.

```
1: for category in AllCategories do
2:    matchedDensityInitialize(category)
3:    for point in AllPoints_c do
4:       movePoint(point)
5:       movePoint(point)
6:    end for
7: end for
```

but valid. Matched-density initialization is performed to ensure the plant density of the reproduction matches that of the input exemplar as this is deemed a vital property of vegetation state.

Iterative Point Moving When points of a given category have been initialized and the required density reached, iterative point-moving is performed where each point is iterated over and moved to two randomly locations. The new distribution strength is calculated after each move and the best scoring move is accepted with probability $P_{acceptance}$, calculated using Equation 6.12. Although more than two random moves could be attempted for each point, it has a big impact on performance. Two has been selected through trial-and-error as it strikes a good balance between performance and resulting realism. Like other configuration parameters, however, this value can easily be modified to improve realism at the cost of processing speed. A possible extension to this work would be to make the number of trial moves per point dynamic depending on the total point count in the distribution.

$$P_{acceptance} = \frac{Strength_{n+1}}{Strength_n} \tag{6.12}$$

Where:$Strength_{n+1}$ is the aggregated distribution strength after the move; $Strength_n$ is the aggregated distribution strength before the move.

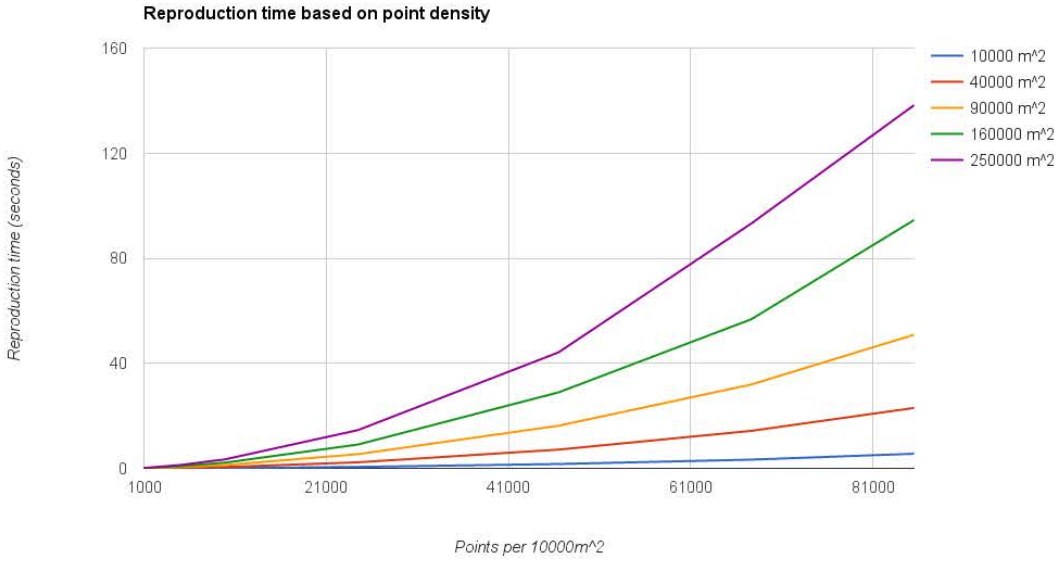

Figure 6.15: *Reproduction time based on point density for different reproduction areas.*

Performance The reproduction area and point density are two properties which greatly affect performance as both effect the number of points to reproduce and, therefore, the reproduction time. Because an decrease in plant density leads to less points within R_{max} of any given source point, perfromance should increase with reproduction area if the plant count is kept fixed and plant locations span the entirety of the analysis window. To determine to what extent and the correlation between plant density, reproduction area and performance, test reproductions are performed using the analysis data generated when performance testing the analysis stage (Section 6.4.1).

Figure 6.15 plots the reproduction performance based on point density for various reproduction areas. It shows the correlation between plant density and reproduction time to be exponential and the exponential increase more severe when reproducing larger areas. This is expected, however, as larger areas will require more points to be added in order to meet the required density. The most extreme test scenario is to reproduce a distribution with an original density of 85530 for 10'000 m^2 over an area of 250'000 m^2. To do so, this test had to place over 50 million points and took 138 seconds to do so.

Using the test data generated to plot Figure 6.15, it is possible to plot the reproduction time based solely on plant count for various densities (see Figure 6.16). It shows the correlation between plant count and reproduction time to be linear and dependent on point density. The

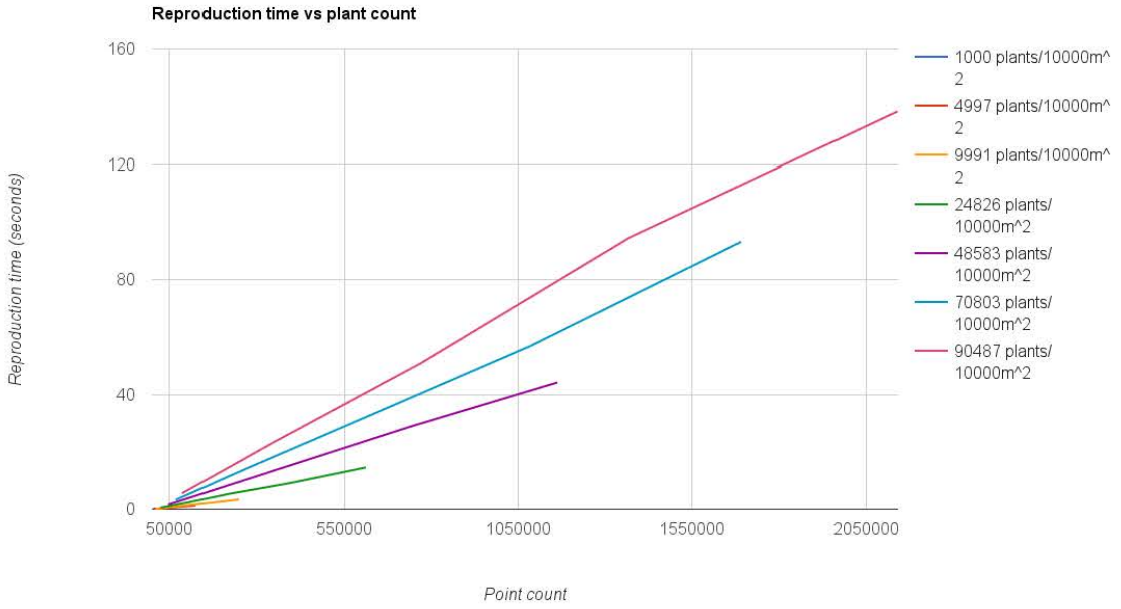

Figure 6.16: *Reproduction time based on point count for different densities.*

reason it is sensitive to plant density is because the denser the points are the the more of them will be within a distance of R_{max} and, therefore, need to be taken into consideration when calculating the strength of the distribution. Based on this, in order to keep reproduction times manageable irrespective of point density (under a minute), the total reproduced plant count is limited to half a million. If large areas need to be reproduced with a plant count higher than this limit, repeating/tiling is performed. Appendix C outlines the maximum reproduction areas that can be achieved, given this limitation, for different plant densities. Although the repetition (tiling) performed will increase for denser distributions, it will not necessarily be more noticeable as denser distributions will tend to have less distinct patterns and be more closely correlated to random.

6.4.3 Caching Distribution Data

In order to prevent repeated costly runs of the ecosystem simulator for identical resource parameters, the analysed distribution data is stored and tracked in a database. This way, if a plant distribution is requested for a simulation which has already been run, the ecosystem simulator is bypassed entirely and the stored distribution data used. A custom binary file format is used in order to save space when storing the necessary distribution analysis data.

6.4.4 Results

The important properties of the input exemplars which must be reproduced are: *inter and intra species separation*, *plant size* and *species densities*. To ensure these are accurately reproduced, an ecosystem simulator run is performed containing *shade-loving*, *shade intolerant* and *canopy plants*. The resulting plant distribution is subsequently used as input exemplar to stress test the distribution analyser and reproducer. Figure 6.17 shows an overview and zoomed subsection of the input exemplar along with it's associated reproduction. From this, along with the point count of individual species, it is possible to conclude that point density and point size is accurately replicated.
To determine whether intra and inter-species spacing is accurately reproduced, the reproduction distribution is re-analysed in order to produce the pair correlation histograms of the reproduced distribution. The original and reproduced histograms are then compared to ensure they follow similar trends (see Figure 6.18). Important properties to note which are accurately reproduced are:

- No plants appear within the radius of the shade-loving (category 6) and shade-intolerant plants (category 5)
- The density of shade-intolerant plants (category 5) drastically decreases within the radius of canopy plants (category 9).
- The density of shade-loving plants (category 6) increases within the radius of canopy plants (category 9). Note that this increase it much severe for the reproduced distribution than for the original. This is caused by the reproduction algorithm for shade-loving plants (see Section 6.4.1). In the ecosystem simulator, new shade-loving plants are spawned based on the location of existing instances of the given species. This naturally leads to a large aggregation of these plants under the same canopy and therefore a high density in the negative bin when radial distribution analysis is performed. During radial distribution reproduction, however, the plants are placed under randomly selected canopies and therefore aggregation under the same canopy is reduced drastically. It is important to note that the reproduction is still valid for the shade-loving species, however, as all instances do appear within the shaded canopies of existing plants.
- The density decreases drastically for canopy plants within the canopy of other canopy plants (category 9) as it blocks access to available illumination.

Note that the pair correlation histogram of the shade-loving species with itself (category 6 and 6) is substantially different between original and reproduction. The reason for this is because during the ecosystem simulator, seeding is performed from existing plant instances which will lead to the clustering of plants under the same canopy, very close to each other. During iterative

point-moving, the probability of shade-loving plants to cluster under neighbouring canopy plants is very slim, however, and dispersion under all present canopy plants is much more likely.

Note also that the aim of radial distribution reproduction is not to reproduce a ditribution matching exactly the input exemplar. This is unfeasible due to to the nature and configuration of the radial distribution reproduction algorithm. What radial distribution reproduction is able to do is prevent plants from being placed at illegal or very weak locations because the strength associated to the location is very low (or zero).

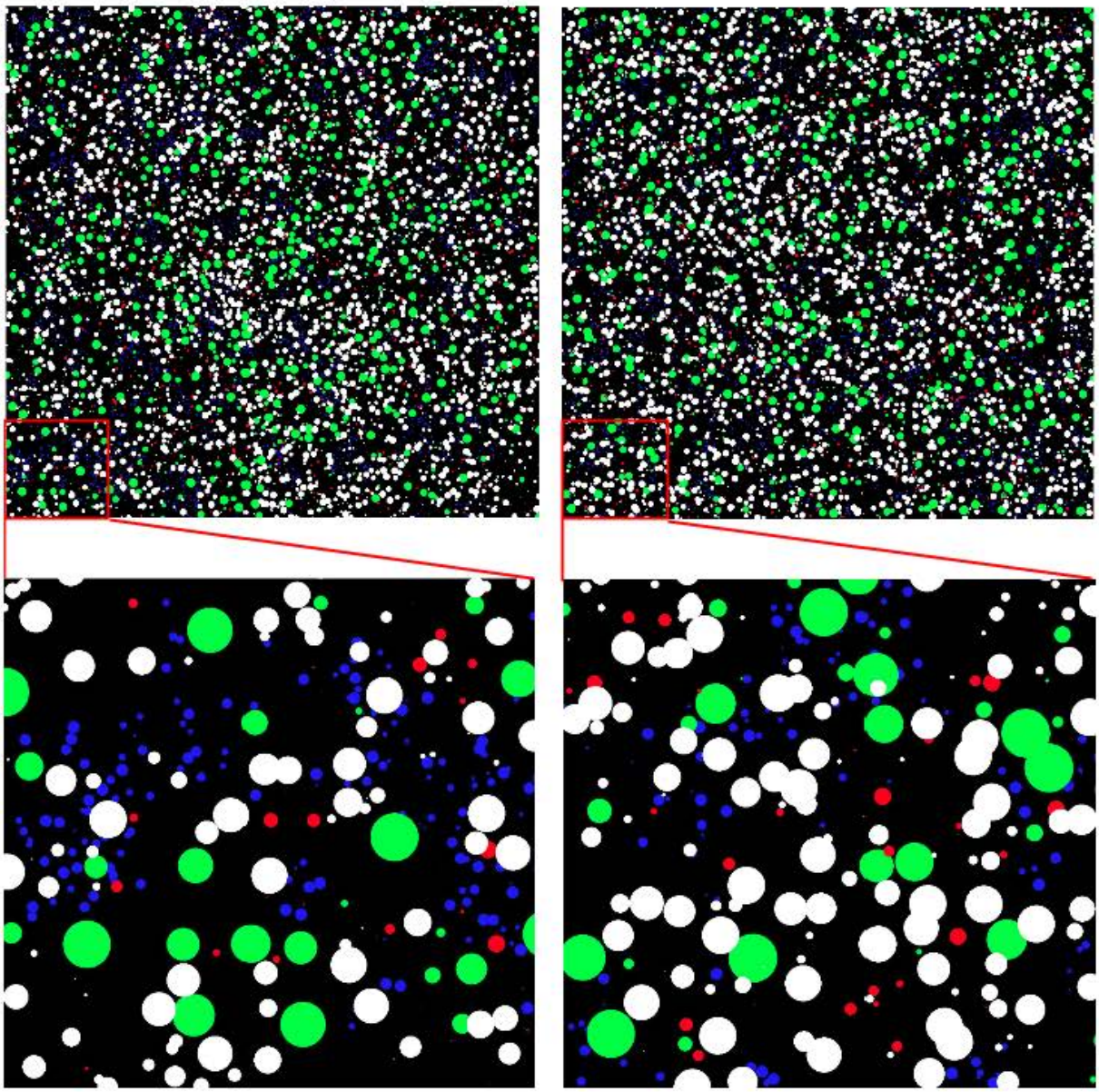

Figure 6.17: *Distribution analysis and reproduction test: Input exemplar (top-left), reproduction (top-right), zoomed (x 5) input exemplar (bottom-left), zoomed (x 5) reproduction (bottom-right).*

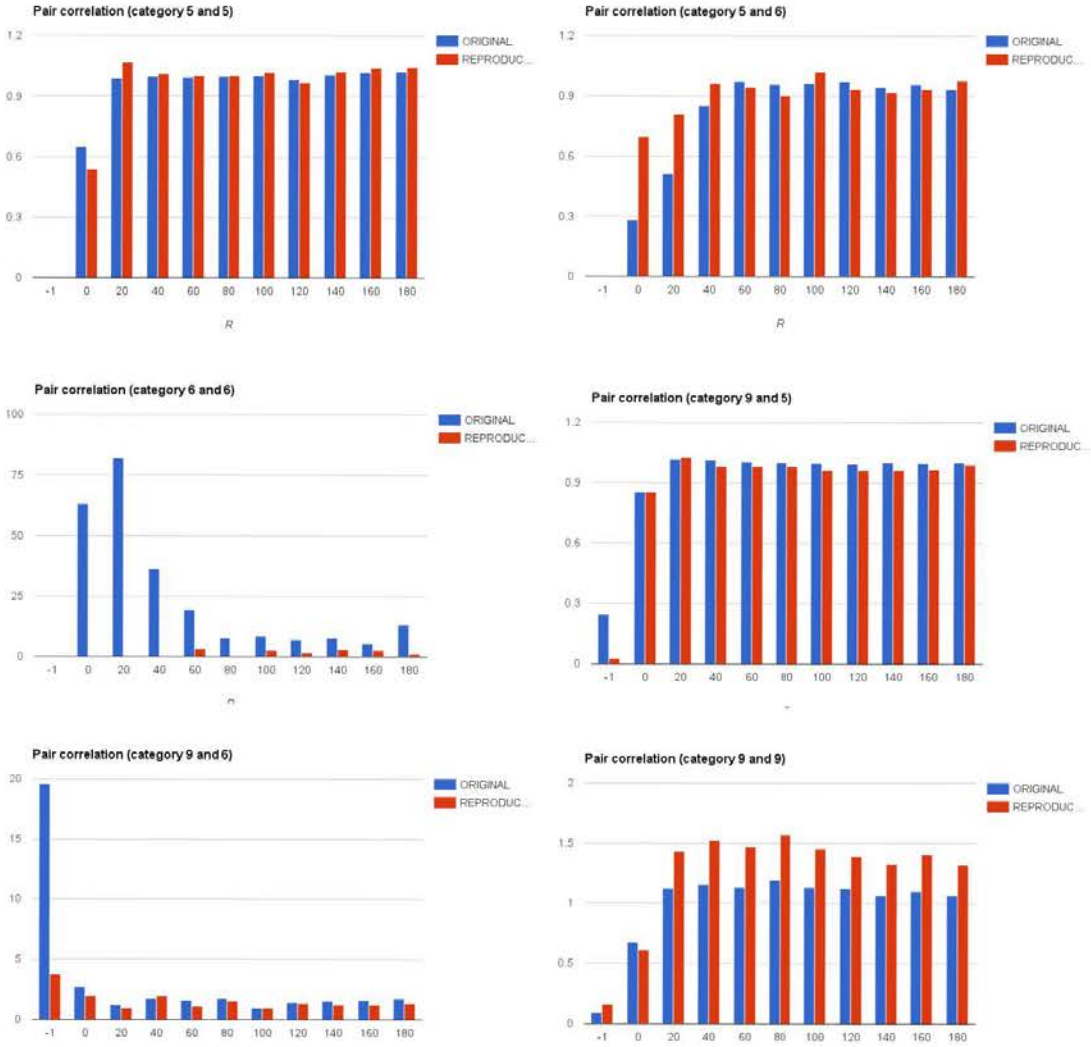

Figure 6.18: *Original (blue) and reproduced (red) pair correlation histograms for different bins where category 6 is a shade-loving, category 5 is shade intolerant and category 9 is a canopy specie. Bin sizes of -1 signify the target category is within the radius of the source.*

Chapter 7

Results and Discussion

This work focuses on river networks (section 4.3), water reserves (section 4.4) and vegetation (section 6) in order to generate realistic virtual rural worlds. Clustering is an achieving this (section 5). Because the results and efficiency of these individual constituents are analysed and discussed in detail in the main body of this document, this chapter focuses more on the aggregate results of all system parts working together. To do so, three terrains taken from the U.S. Geological Survey [1] are loaded and resources specified to model two distinct environments: tropical rainforest and alpine. When modelling these environments, temperature and rainfall data are specified using freely available weather data at locations with such climates [2]. Given this, a set of five species that are suited to the given environment are configured using freely available online species data [3] and a valid distribution created using the ecosystem simulator alongside the clustering algorithm configured with a fixed cluster count of ten. The resulting water networks and cluster plant distributions are subsequently analysed. Specifying resources to generate the virtual worlds in this chapter took less than ten minutes for all scenarios.

It is important to note that these tests do not attempt to reproduce real-world plant distributions given identical resources but rather produce plausible distributions given the properties of the generated clusters and configured species. Plausibility here denotes that there is a clear correlation between a species distribution and it's suitability to the given environment.
In order to do so, suitability analysis is performed on each resulting specie and cluster pair. The cluster count is fixed and the number of species, small, in order to keep this analysis data manageable and efficiently communicable.

In order to show the scalability of the system, a third test is performed similar to the tropical test but with a larger number of species (fifteen). Because of the large amount of data which

[1] `http://www.usgs.gov`
[2] `http://weather-and-climate.com`
[3] `http://davesgarden.com/guides/pf/`

results from this test, however, detailed analysis of each specie and cluster pair is not performed. Instead, we briefly discuss notable characteristics of each cluster.

At the end of this chapter, the strength and weaknesses of the system in terms of the generated results are discussed.

7.1 Tropical Rainforest

Tropical rainforest or equatorial climates are characterised by frequent and heavy rainfall and are situated on or close to the equator. Because of their location they have no seasons and therefore very little temperature and illumination variance throughout the year.

Toamasina, a city in Madagascar at latitude 18°south, is the location on which climate data is based for these tests.

Because of its water absorption capabilities, loamy soil is modelled in this test scenario. To do so, the terrain is filled with a base soil infiltration rate of fifteen millimetres per hour (see Table 4.2 for the correlation between soil type and soil infiltration rate). To model rocky cliff faces, the soil infiltration rate is set to zero when the slope angle surpasses forty degrees.

The configured monthly rainfall, rainfall intensity and resulting soil moisture and weighted soil moisture is summarized in Table 7.1. The resulting water networks that form on the terrain are illustrated in Figure 7.1.

The temperature extremes configured are twenty-one degrees for June and twenty-six degrees for December, resulting in monthly temperatures outlined in Table 7.1. The lapse rate is kept at its default value of 6.5 degrees loss for every thousand metres gained.

Resource	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**
Rainfall (mm)	410	382	478	322	228	259	288	218	121	132	169	357
Rainfall Intensity (mm/h)	18	17	20	16	11	12	14	11	7	7	8	16
Soil Moisture (mm)	341	337	359	302	228	259	288	218	121	132	169	334
Weighted soil moisture (mm)	311	338	349	327	274	256	268	248	181	143	149	246
Temperature (°)	25	24	24	23	22	21	22	23	24	24	25	26

Table 7.1: *Tropical rainforest: Monthly rainfall, rainfall intensity, soil moisture, weighted soil moisture and temperature at zero metres. Green cells represent values explicitly input for this test scenario, all others are procedurally calculated. The soil moisture is valid for terrain vertices with a slope less than forty degrees and therefore for which the soil infiltration rate is 15 mm per hour. All other vertices have a soil moisture of zero (modelled rock cliff faces).*

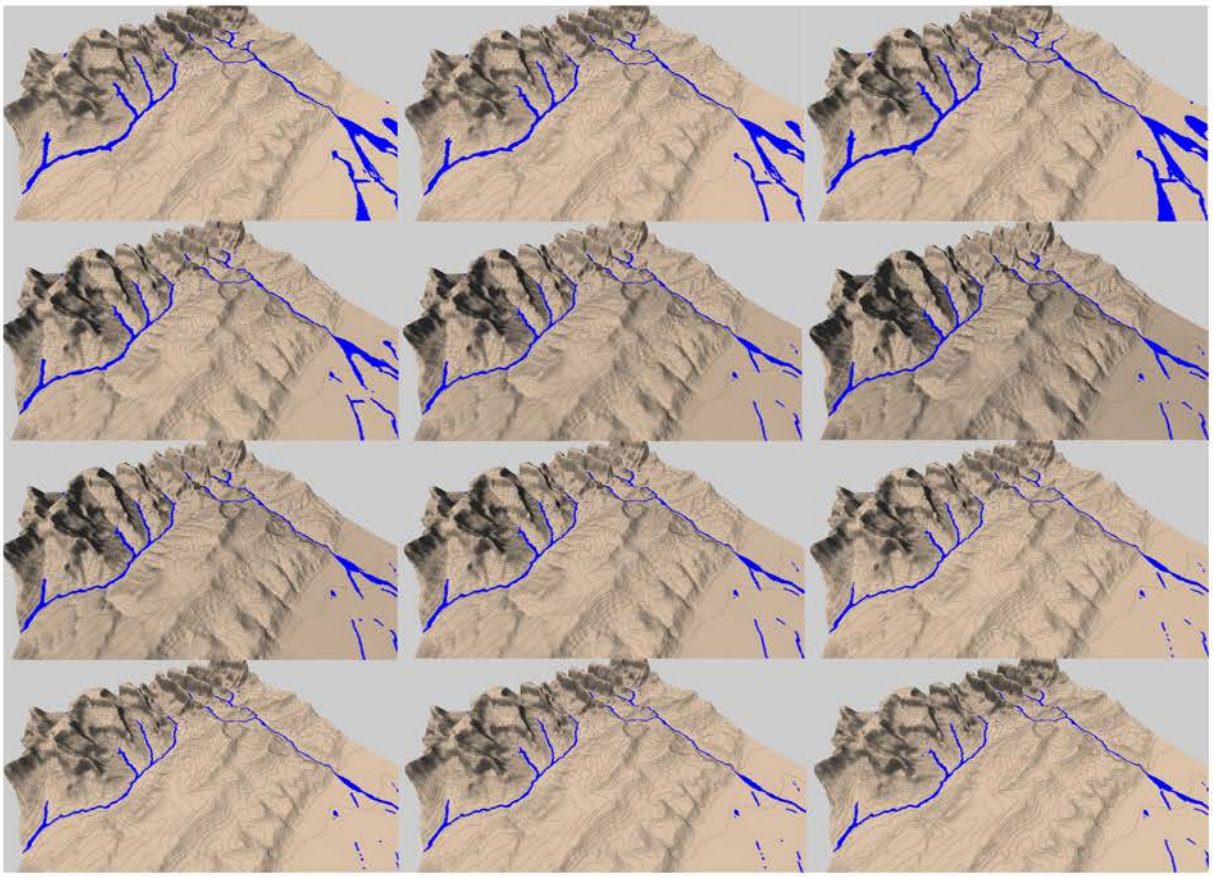

Figure 7.1: *Tropical rainforest: Water networks that form on the terrain in every month. From left to right, top to bottom.*

The number of generated clusters is set to ten. The clustered terrain is shown in Figure 7.2 and the corresponding cluster properties summarized in Figure J.1 and Table 7.2.

Five tropical rainforest plant species are configured for this test: *Brazil nut*, *Cavendish banana*, *Heliconia*, *King of Bromeliads* and *Orchid*. The properties associated with each are summarized in appendix E.

The species suitability filtering (see Section 6.2) automatically filters out ill-suited species from being inserted in some clusters as the slope, soil moisture and/or temperature do not meet the requirements of the given species. Note that species are not filtered out due to unsuited illumination during specie suitability filtering as the sun exposure can vary during a simulation as canopy plants grow and project shade. Table 7.3 summarizes this information and illustrates the species which are suited to individual clusters and, if not, which resource acts as a bottleneck. From this, it is possible to conclude that no plants are able to grow, and therefore no ecosystem simulator needs to be run, for clusters 3, 5, 7 and 8. Clusters 3, 5 and 7 have to steep a slope and too little soil humidity to cater for these species. *Cluster 8* has soil that is

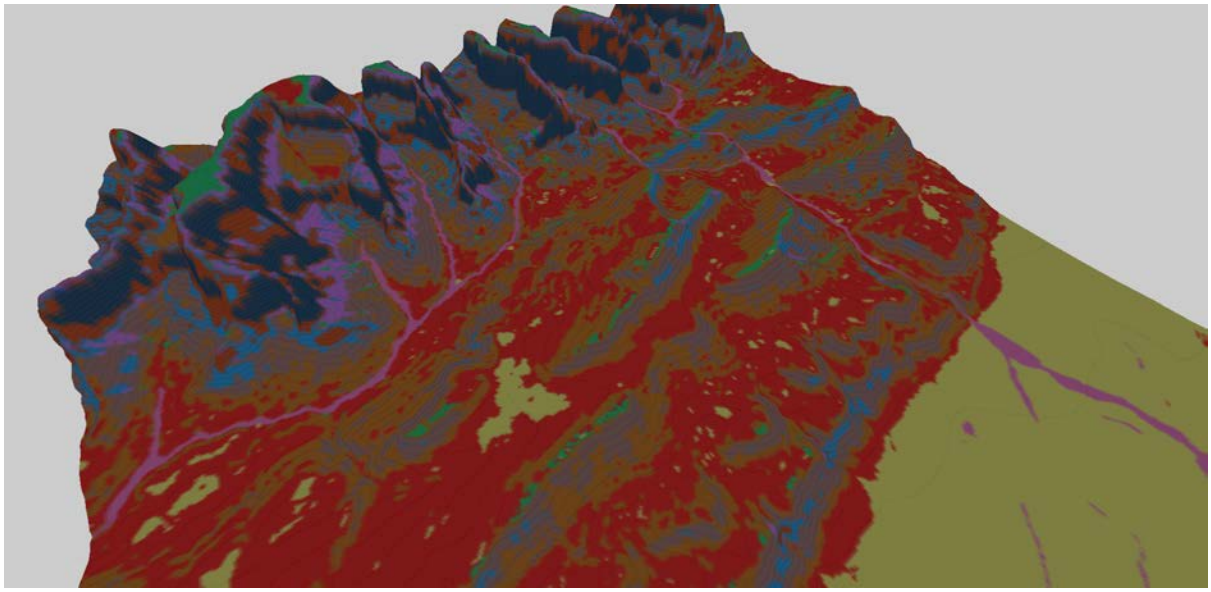

Figure 7.2: *Tropical rainforest: Resulting terrain clusters.*

Cluster	**Slope (degrees)**	**Coverage area (% of terrain)**
Cluster 1	22.3	19.3
Cluster 2	32.2704	16.2
Cluster 3	69.092	4.7
Cluster 4	2.95581	18.1
Cluster 5	42.9369	4.8
Cluster 6	11.8319	22.1
Cluster 7	54.1674	6.0
Cluster 8	8.0803	2.1
Cluster 9	14.0102	2.5
Cluster 10	32.1679	3.8

Table 7.2: *Tropical rainforest: Slope and coverage area of each cluster.*

too moist as it represents the points on the terrain with standing water.

Figures J.2, J.3, J.4, J.5, J.6 and Table 7.4 illustrate how suited the remaining clusters are for each plant species. This information proves especially useful for determining how suited given species are in terms of illumination, as the species suitability scoring ignores this resource since it varies throughout a simulation as plants grow and project shade.
From this, it is possible to conclude that: *Clusters 9 and 10* prevent *Brazil Nut*, *Cavendish Banana* and *Heliconia* from growing as their minimum illuminations falls below these species

	Orchid	**Cavendish banana**	**Heliconia**	**Brazil Nut**	**King of Bromeliads**
Cluster 1	Y	Y	Y	Y	Y
Cluster 2	Y	N (S^+)	Y	N (S^+)	Y
Cluster 3	N (S^+, SH^-)	N (S^+, SH^-)	N (S^+, SH^-)	N (S^+, SH^-)	N (S^+, SH^-)
Cluster 4	Y	Y	Y	Y	Y
Cluster 5	N (SH^-)	N (S^+, SH^-)	N (S^+, SH^-)	N (S^+, SH^-)	N (SH^-)
Cluster 6	Y	Y	Y	Y	Y
Cluster 7	N (SH^-)	N (S^+, SH^-)	N (S^+, SH^-)	N (S^+, SH^-)	N (S^+, SH^-)
Cluster 8	N(SH^+)	N (SH^+)	N (SH^+)	N (SH^+)	N (SH^+)
Cluster 9	Y	Y	Y	Y	Y
Cluster 10	Y	N (S^+)	Y	N (S^+)	Y

Table 7.3: *Tropical rainforest: Summary of the species suitability filter pass on each cluster. This step filters out ill-suited species from being inserted into a given cluster is the slope, soil humidity or temperature is ill-suited. If a species is not suited, the cell is highlighted red and the reason is stated in brackets where S is the slope, T is the temperature and SH is the soil humidity, $^+$ signifies too much and $^-$ too little of the given resource.*

lower limits. These clusters also prevent shade-loving *King of Bromeliads* and *Orchids* from growing as their maximum illumination surpasses the species upper limits. Clusters 9 and 10 are therefore unsuited to all species.
Cavendish Banana and *Brazil Nut* plants are unsuited to clusters 1 and 2 as the clusters minimum illumination falls below the species lower limit.
King of Bromeliads and *Orchids* are universally unsuited as the clusters maximum illuminations surpass the species upper-limit. However, as these species are shade-loving, there is a possibility they survive in the undergrowth of striving plant instances. Given all this, the clusters to which plant species are suited and plausible distributions created are clusters 1, 2, 4 and 6. Together, these clusters make up only 70% of the terrain.

Table 7.5 summarizes the plant distributions generated by the ecosystem simulator for each cluster by stating the associated plant instance count, average, minimum and maximum size. We discuss each cluster separately below.

Cluster One Cluster one only permits Heliconia species to grow. As stated previously, *Brazil Nut* and *Cavendish Banana* plants are unable to grow in this cluster due to unsuitable illumination. *King of Bromeliads* and *Orchids* are also unable to grow as, without *Brazil Nut* and

	Brazil Nut	Cavendish Banana	Heliconia	King of Bromeliads	Orchid
Start of decline	15	20	25	35	35
Max	25	30	40	45	55
Cluster 1	22.3	22.3	22.3	22.3	22.3
Cluster 2	32.2	32.2	32.2	32.2	32.2
Cluster 4	2.9	2.9	2.9	2.9	2.9
Cluster 6	11.8	11.8	11.8	11.8	11.8
Cluster 9	14	14	14	14	14
Cluster 10	32.1	32.1	32.1	32.1	32.1

Table 7.4: *Tropical rainforest: Species suitability to clusters 1, 2, 4, 6, 9 and 10 in terms of slope, where: Green means the species is completely suited, orange means the species is negatively impacted by the slope, and red that the species is completely ill-suited. All values are in degrees.*

Cavendish Banana plants, these species lack the essential canopy shade necessary to grow. Although Heliconia thrives well in this cluster, because this species does not live long (it starts to decline after 3 years), it does not provide cover for shade-loving species long enough for them to develop and survive in the long term.

Cluster Two For the same reasons as for cluster one, all but Heliconia species are unable to grow in this cluster. However, as shown by its average size, this species fares poorly in this cluster, mostly because at 32 °, the clusters slope is outside the species optimal range.

Cluster Four Unlike clusters one and two, cluster four permits *Brazil Nut* and *Cavendish Banana* plants to grow which, in turn, permit shade-loving *King of Bromeliads* and *Orchids*. This cluster is close to optimal for all species and, as such, all species come close to reaching there maximum sizes. It covers 18% of the terrain.

Cluster Six The distribution generated for cluster six is very similar to that of cluster four. With a bit less illumination and humidity available, however, the resource intensive *Brazil Nut* and *Cavendish Banana* thrive a bit less which, in turn, also provides fewer shade-loving plants. Unlike other plant species, *Heliconia* reaches a greater average size in this cluster compared to cluster five. Because this species has a short lifespan, the plant instances iteratively spawn and die. When they spawn, they are competing for resources against much more vigorous (large) plants and therefore struggle to get the resources necessary for there development. In this cluster the most vigorous plants (*Brazil Nut* and *Cavendish Banana*) survive less well and the competition for available resources is that much less intense, making it slightly easier for these

plant species.

Specie	Property	Cluster 1	Cluster 2	Cluster 4	Cluster 6
Brazil Nut	Count	0	0	324	342
	Min height (cm)	-	-	11	19
	Max height (cm)	-	-	1879	1984
	Avg height (cm)	-	-	396	357
CB	Count	0	0	371	408
	Min height (cm)	-	-	2	4
	Max height (cm)	-	-	215	215
	Avg height (cm)	-	-	70	64
Heliconia	Count	625	1456	766	793
	Min height (cm)	5	1	5	9
	Max height (cm)	297	22	298	298
	Avg height (cm)	132.8	10.2	136.5	140
KOB	Count	0	0	612	281
	Min height (cm)	-	-	6	16
	Max height (cm)	-	-	86	78
	Avg height (cm)	-	-	41	38
Orchid	Count	0	0	557	260
	Min height (cm)	-	-	1	2
	Max height (cm)	-	-	36	45
	Avg height (cm)	-	-	8.5	6.2

Table 7.5: *Tropical rainforest: Vegetation content of the hundred by hundred metre simulation window for each cluster. CB and KOB represent species Cavendish Banana and King of Bromeliads, respectively.*

7.2 Alpine

We refer to an alpine climate, here, as one that experiences below zero temperatures during certain periods of the year. These extreme temperatures ensure that only plant species able to survive in sub-zero temperatures will persist in the long term. It is possible, however, for annuals to sprout during periods when the temperature increases.

In this section resources are configured, clusters generated and plausible vegetation distributions computed and analysed in the same manner as done previously in Section 7.1. These tests differ in their input resource configurations and plant species, however, in order to better model an alpine climate.
Verbier, a Swiss alpine city at a latitude of 43°north, is the location on which input climate data is based. See Table 7.6 for details concerning this input data. To more accurately represent the drastic change in temperature that occurs in alpine climates, the lapse rate is changed from its default and set to a ten degree drop in temperature for every thousand metres altitude gain. To model a drier loamy soil as for the previous tests, the soil infiltration rate is set to ten millimetres per hour (see Figure 4.2). Rocky cliff faces are modelled with the soil infiltration rate set to zero for slope angles surpassing forty-five degrees. Figure 7.3 shows the monthly water networks that form as a result of the configured rainfall and soil properties.
Spruce, *Maple*, *Moss Campion*, *Daisy* and *Beech* are the five species which are used for these tests. Refer to appendix F for the resource requirements associated with each.

Resource	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**
Rainfall (mm)	57	49	41	27	42	51	42	42	46	48	52	63
Rainfall Intensity (mm/h)	16	12	11	7	11	14	11	11	12	13	14	17
Soil Moisture (mm)	36	41	37	27	38	36	38	38	38	37	37	37
Weighted soil moisture (mm)	36	38	38	32	34	35	38	38	38	38	37	37
Temperature (°)	5	10	15	20	25	30	25	20	15	10	5	0

Table 7.6: *Alpine: Monthly rainfall, rainfall intensity, soil moisture, weighted soil moisture and temperature at zero metres. Green cells represent values explicitly input for this test scenario, all others are procedurally calculated. The soil moisture is valid for terrain vertices with a slope less than forty-five degrees and therefore for which the soil infiltration rate is 10 mm per hour. All other vertices have a soil moisture of zero (modelled rock cliff faces).*

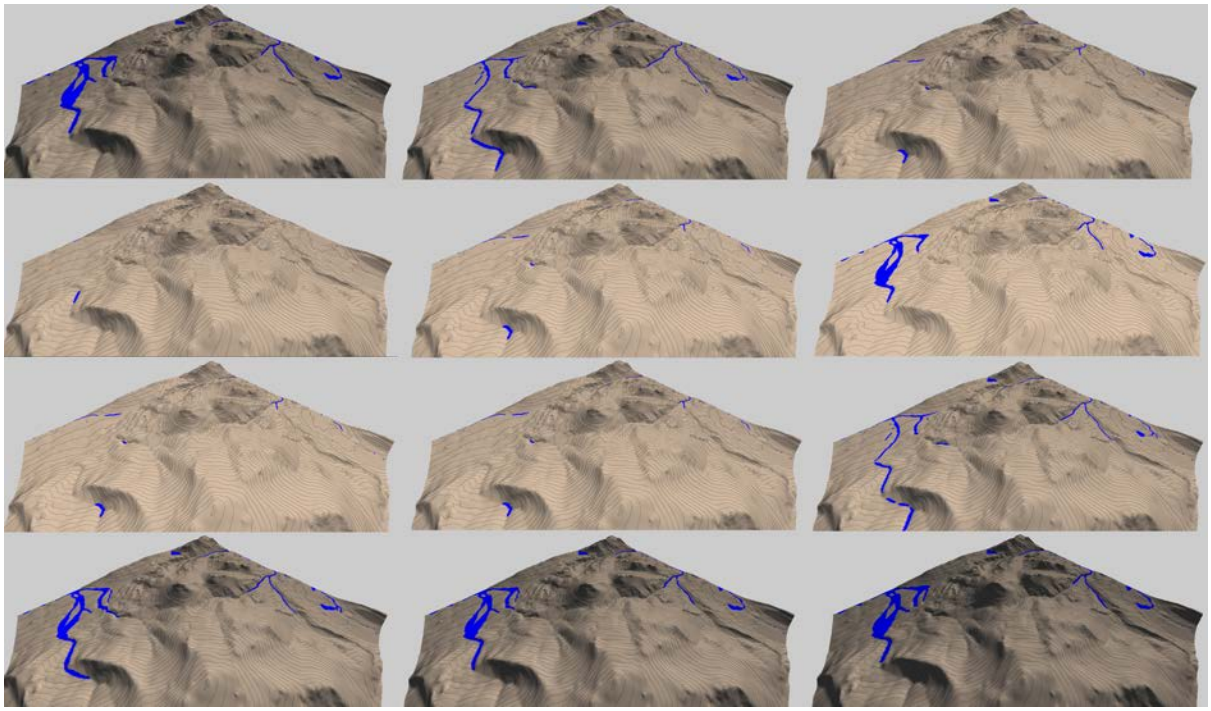

Figure 7.3: *Alpine: Water networks that form on the terrain in every month. From left to right, top to bottom.*

Figure 7.4 illustrates the clustered terrain. Corresponding cluster properties are summarized in Figure K.1 and Table 7.7.

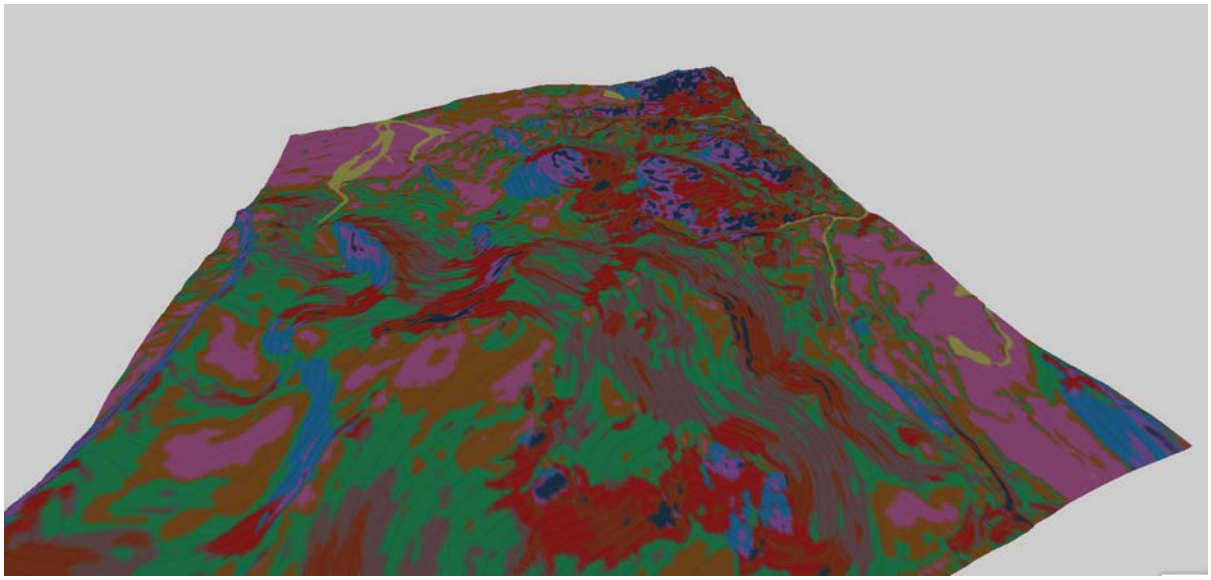

Figure 7.4: *Alpine: Resulting terrain clusters.*

The results of the specie suitability filtering step is summarized in Figure 7.8. As stated previously, this step filters out species from being inserted into given clusters because the tem-

Cluster	**Slope (degrees)**	**Coverage area (% of terrain)**
Cluster 1	10.2	18
Cluster 2	21.1	13.5
Cluster 3	35.8	3.7
Cluster 4	5.3	2
Cluster 5	19.9	4.2
Cluster 6	22.3	10.4
Cluster 7	28.2	9.3
Cluster 8	5	15
Cluster 9	15.7	19.4
Cluster 10	28.7	4.2

Table 7.7: *Alpine: Slope and coverage area associated with each cluster*

perature, slope or soil moisture is ill-suited. From this, it is possible to conclude that no plants are able to grow, and therefore no ecosystem simulator needs to be run, for cluster 4. This is caused by excess moisture as this cluster represents areas on the terrain covered in water.
This data also shows that Daisies are only suited to grow in cluster 8. In all other clusters, the temperature drops below its minimum. It is important to note that, even though Daisies are a season plant, in order for them to be considered for growth at any time in the year, the temperature must be above the species absolute minimum all year round. If the temperature drops below this absolute minimum, it is considered too cold for even the seeds to survive through the winter. Cluster 8 is suited for this seasoned plant as the minimum annual temperature is -6 degrees, one degree higher than the species absolute minimum. At this temperature, it is too cold for the given species to grow but is sufficiently high for the seeds to survive in order to sprout in the warmer summer months.

To visualise each species suitability to the given clusters in terms of temperature, soil moisture and sun exposure, graphs similar to those plotted in the tropical tests (Section 7.1) are plotted (see Figures K.2, K.3, K.4, K.5 and K.6). Table 7.9 summarizes the species suitability in terms of slope. From this data, it is possible to conclude that: Available illumination drops beneath the minimum permitted for all species in clusters *3, 5, 6* and *10*; Illumination in clusters *1, 8* and *9* also falls beneath the lower limit for *Moss Campion, Daisy* and *Beech*; Illumination in cluster 7 also prevents *Daisy* and *Beech* from surviving. In summary, only clusters one, two, seven, eight and nine are able to support plant development. Together, they make up seventy-five percent of the terrain.

	Spruce	Maple	Moss Campion	Daisy	Beech
Cluster 1	Y	Y	Y	N (T^-)	Y
Cluster 2	Y	Y	Y	N (T^-)	Y
Cluster 3	N (S^+)	Y	Y	N (T^-)	Y
Cluster 4	N (SH^+)	N (SH^+)	N (SH^+)	N (SH^+)	N (SH^+)
Cluster 5	Y	Y	Y	N (T^-)	Y
Cluster 6	Y	Y	Y	N (T^-)	Y
Cluster 7	Y	Y	Y	N (T^-)	Y
Cluster 8	Y	Y	Y	Y	Y
Cluster 9	Y	Y	Y	N (T^-)	Y
Cluster 10	Y	Y	Y	N (T^-)	Y

Table 7.8: *Alpine: Summary of the species suitability filter pass on each cluster. This step filters out ill-suited species from being inserted into a given cluster is the slope, soil humidity or temperature is ill-suited. If a species is not suited, the cell is highlighted red and the reason is stated in brackets where S is the slope, T is the temperature and SH is the soil humidity, $^+$ signifies too much and $^-$ too little of the given resource.*

	Spruce	Maple	Moss Campion	Daisy	Beech
Start of decline	25	15	66	50	10
Max	50	30	90	75	40
Cluster 1	10.2	10.2	10.2	10.2	10.2
Cluster 2	21.1	21.1	21.1	21.1	21.1
Cluster 3	35.8	35.8	35.8	35.8	35.8
Cluster 5	19.9	19.9	19.9	19.9	19.9
Cluster 6	22.3	22.3	22.3	22.3	22.3
Cluster 7	28.2	28.2	28.2	28.2	28.2
Cluster 8	5	5	5	5	5
Cluster 9	15.7	15.7	15.7	15.7	15.7
Cluster 10	28.7	28.7	28.7	28.7	28.7

Table 7.9: *Alpine: Species suitability to clusters 1 to 10 (excluding 4) in terms of slope, where: Green means the species is completely suited, orange means the species is negatively impacted by the slope, and red that the species is completely ill-suited. All values are in degrees.*

Specie	Property	Cluster 1	Cluster 2	Cluster 7	Cluster 8	Cluster 9
Spruce	Count	3112	3242	3916	3272	3395
	Min height (cm)	1	1	1	1	1
	Max height (cm)	137	166	130	131	143
	Avg height (cm)	70	87	67	66	73
Maple	Count	556	850	0	24	528
	Min height (cm)	3	1	-	2	3
	Max height (cm)	481	146	-	3	520
	Avg height (cm)	248	72	-	2	289
MC	Count	0	3239	2432	0	0
	Min height (cm)	-	1	1	-	-
	Max height (cm)	-	16	14	-	-
	Avg height (cm)	-	9	7	-	-
Daisy	Count	0	0	0	2	0
	Min height (cm)	-	-	-	8	-
	Max height (cm)	-	-	-	8	-
	Avg height (cm)	-	-	-	8	-
Beech	Count	0	914	0	0	0
	Min height (cm)	-	1	-	-	-
	Max height (cm)	-	173	-	-	-
	Avg height (cm)	-	85	-	-	-

Table 7.10: *Alpine: Vegetation content of the hundred by hundred metre simulation window for each cluster. MC is Moss Campion.*

Table 7.10 gives an overview of the suitability of each specie and cluster pair by providing the plant instance count along with the minimum, maximum and average plant size. We discuss each cluster below.

Cluster one A low monthly illumination of five hours in December prevents *Moss Campion* and *Beech* plants from surviving in this cluster.
With just over five hundred and fifty instances, *Maple* plants are able to survive in this cluster. With an average height of just over a fifth of the maximum, resources are not optimal. Many of these plants are still growing, however, as at one hundred year (simulation time), only 60% of the growth period has been completed. With this factored in, these plants are at just over a third of their maximum size. The factors preventing optimal growth of *Maple* plants in this cluster are too much rainfall in the winter months and too much sun exposure and heat exposure

in the summer months.
The growth potential of *Spruce* in this cluster is very similar to that of *Maple*. Although humidity and sun exposure are better suited to the *Spruce*, they are more sensitive to and ill-affected by the high summer temperatures.

Cluster two This is the cluster with the most biodiversity, with four out of the five available species able to survive.
As with cluster one, *Spruce* struggles to grow during the summer months due to high temperatures. However, as these highs are a bit less severe in this cluster, growth does improve. With an average height of just under 90 centimetres, they reach just over a third of there maximum size.
Similarly, *sun exposure* and *temperature* are more suitable in this cluster for *Maple* plants. Soil humidity is also optimal all year round. With an average height of 146 centimetres, approximately 20% of the species maximum, however, *Maple* plants are severely affected by the slope of this cluster (21 °).
With over three thousand instances and an average size of over half of the species maximum, Moss Campion plants are the most suited to this cluster. December is the only month in which Moss Campion growth is ill-affected, caused by low sunlight exposure.
Beech plants are also able to survive in this cluster, but, because of low winter temperatures and limited sunlight exposure in December, these plants only reach 20% of their maximum size.

Cluster seven A combination of high summer temperatures and a slope very close to the species upper-limit prevents *Maple* plants from developing in this cluster.
Low sunlight exposure in December also prevents this cluster from being suitable for the growth of *Beech* plants.
Spruce is able to grow but, with an average size of only a fourth of the species optimum, are unable to reach their maximum growth potential. High summer temperatures and slope are the limiting factor here.
The environment is also suited to *Moss Campion* growth. Low sun exposure during winter months and high peak summer temperatures limits growth to just under 50% of the maximum, however.

Cluster eight Although *Daisies* are able to grow in this cluster as it is the only one for which the temperature stays above this species lower limit, because the temperature is below its prime range for seven months of the year, growth is limited. In addition, illumination is also below this species optimal range for five months. Because of this, the average daisy height is only a fifth of its maximum size, clearly showing that this habitat is from from optimal.

A very small number of Maple plants (twenty-four) are present in this cluster and with an average height of two centimetres, very far from its maximum of twelve metres, it is evident this species struggles. A combination of too much water in the winter months along with too much heat and sun in the summer months proves to be a complete bottleneck for *Maple* growth in this cluster. Given how much this species is struggling, it is clear that the habitat is inadequate. So much so that its presence in this cluster is questionable altogether. As an extension to this work, a final filtering could be performed on the output of the ecosystem simulator to remove species which don't reach a minimum ratio of their maximum.
The species that strives best is *Spruce*. However, with an average height of just over a quarter of the species maximum, available resources are far from ideal. Although humidity and illumination is optimal all year round, high summer temperatures have a negative impact on growth.

Cluster nine Low sunlight exposure in the month of December prevents *Beech* and *Moss Campion* growth in this cluster.
Spruce plants are able to grow but are affected by high summer temperatures and low sun exposure in December. The same factors negatively impact the growth of *Maple* within this cluster.

7.3 Tropical rainforest with fifteen plant species

Sections 7.1 and 7.2 focus on analysing, in detail, the species content of each individual cluster. Because of the level of detail necessary for this analysis and to keep information communicable, the number of species used is limited to five. The system should be scalable, however, and able to cater for a larger number of species. This is the focus of this chapter. To do so, climate data identical to the tropical test run is configured (see Table 7.1), the same number of clusters produced (ten), but a total of fifteen tropical plant species are configured and used when generating the vegetation distributions (refer to appendix E and G for species details). Because the configuration data and cluster count are identical to that of the tropical test, the clusters remain unchanged and are summarized in Figure J.1 and Table 7.2.

The instance count along with the minimum, maximum and average height of each specie and cluster pair is summarized in appendix H. Similarly to the previous tropical test, these results show that: no plants are able to survive in clusters 3, 5 and 7 primarily because there is too little soil moisture; cluster 8 is ill-suited to all species because of too much soil moisture; clusters 9 and 10 are unsuited because the sun exposure is too little to enable canopy plants to grow but too high to permit shade-loving species to grow without vital shade cover of these canopy plants.

When discussing the species content of each cluster below, those present in the tropical test discussed previously (section 7.1) are ignored as their cluster suitability remains unchanged. A property made apparent by these new tests with added species, however, is that although the suitability of the pre-existing species to each cluster does not change, their instance count and size properties often do. This is a direct consequence of adding more species to the simulation and, therefore, having to redistribute resources. This can work in the species favour, as is the case for Heliconia in cluster one for example. It can also be a disadvantage, as is the case for King of Bromeliads in cluster four. It is difficult to pinpoint why adding species works as an advantage or disadvantage to other species, however, as the cause can be indirect. In clusters five and six, for example, shade-loving species King of Bromeliads and Orchids thrive much less even though the number of canopy plants increases. This is caused by the added competition for resources causing a 75% reduction in the number of Brazil Nut instances. With a maximum canopy width of eight metres, the Brazil Nut is, by a margin, the plant with the largest canopy. Therefore, even though the aggregate number of canopy plants increases, the actual area of shade decreases, which, as a consequence, causes shade-loving plants to flourish less.

Because the soil moisture requirements of the Kapok Tree is too high, it is unable to grow in any of these remaining clusters as their minimum drops below the species configured absolute minimum. The opposite is true for Queens Tear, Poinciana, Fern Begonia and Bahama Wild

Coffee as the maximum soil moisture of these clusters is above the maximum allowed for these species and, as such, prevents them from surviving.

Cluster One Bengal Bamboo, Durian and Bougain Villea are unable to grow in this cluster because the average sunlight exposure drops below the minimum permitted for these species during winter. Although still negatively impacted by the limited illumination during winder, the more shade-tolerant Coconut Palm is able to develop in this cluster. Limited soil moisture in October and November also slows this species growth. Because of these bottlenecks, the average plant height reaches only thirty percent of the species optimum. Although, with a maximum of just under five metres, some plant instances do reach over eighty percent of this optimum. This points towards inter-plant competition as being a key bottleneck, also.

Cluster Two Because of low illumination during winter and a slope of over thirty degrees, no new species are able to grow in this cluster. Note the Heliconia doesn't perform as well in this cluster, even though there are no new species present in the final results. This is because, although they don't survive, a larger set of plant species seed on an annual basis and compete for resources in an attempt to grow.

Cluster Four Bougain Villea is unable to grow in this cluster because, at 414 millimetres, the soil moisture surpasses the species maximum in March.
With an average height of over eighty percent of its optimum, Bengal Bamboo thrives extremely well in this cluster. Only slightly too moist soil during the months of February, March and April prevent it from growing to this optimum.
With the exception of the soil being insufficiently moist for a couple of months, Coconut Palm and Durian are also well suited to the resources of this cluster. With an average size of only eighteen and thirteen percent of their optimum respectably, this suitability is not reflected in their size. This is caused by intense competition for both soil moisture and sun exposure through the canopies of thriving Bengal Bamboo and Brazil Nut plants.

Cluster Six The resources of this cluster is extremely similar to that of cluster six and, as a result, so are the plant distributions. A notable difference, however, is that a decrease in the maximum soil moisture of this cluster permits Bougain Villea to grow. A decrease in the minimum sun exposure of this cluster also causes the growth of Bengal Bamboo to be negatively impacted, resulting in an average size decrease of over ten percent.

7.4 Discussion

Weather stations and satellites continuously monitor weather across the globe and the resulting data is made freely available. Out system employs the same data as input, which makes it very easy to model any type of environment. Finding and specifying the input data to model these two environments, for example, took less that ten minutes. The terrains used in this chapter are not taken from real world data of the location on which climate data is being modelled, however, but from a pool of pre-generated terrains. Finding and specifying the necessary input data increases when factoring in the time it takes to find and generate this necessary height-map data.
The least intuitive input is the soil infiltration rate as this necessitates that the user knows the correlation between soil type and soil infiltration rate. This could be improved, however, by providing a list of soils to the user and automating the correlation.

The water networks that the system generates accurately reflect the variation in precipitation quantity and intensity. This is apparent as the networks vary in size and depth on a monthly basis as the amount of calculated standing water varies. In addition, the water networks are generated very quickly for all twelve months and the water flow simulation is rendered in real-time, promoting user interaction.

Similarly, the clustering algorithm also runs in real-time, permitting users to quickly visualise the impact of varying the number of clusters. However, as the number of clusters increases, identifying the different clusters on the terrain becomes difficult and the associated cluster data displayed to the user, overwhelming. A solution to this would be to automatically determine a suitable number of clusters based on terrain resource variability and a more intuitive "level of realism" configuration parameter. By doing so, the clustering element can be bypassed entirely from a user perspective and processed in the background.

In terms of vegetation, the system accurately determines the species suitability to each cluster in terms of slope, temperature and soil moisture and effectively communicates this to the user during the species selection phase (see Section 6.2). Note that sun exposure is not considered at this stage as it varies throughout a simulation as plants grow and project shade on the undergrowth. This suitability data is subsequently used to prevent ill-suited species from being inserted into a given ecosystem simulator run. This accelerates the time it takes to generate vegetation distributions as simulations in which all species are ill-suited are identified beforehand and skipped entirely.
By analysing the average size and instance count of individual species produced by the ecosystem simulator for each cluster, it is apparent that their vigour reflects their suitability to each

cluster environment. Species present, as well as available resources, have a big impact on resulting distributions. This is made apparent in the tropical rainforest tests, for example, where shade-loving plants only grow in clusters suited to canopy plants, on which they depend for vital shade coverage. It is also illustrated when comparing the results of the two rainforest tests (sections 7.1 and 7.3). More species are present in the second test and, as a consequence, inter plant competition is more intense. This has a big impact on resulting plant distributions. In cluster four, for example, the added competition of other large canopy plants causes the number of Brazil Nut plants to decrease by approximately eighty percent. Their average size increases by over forty percent, however, as the fierce competition makes it difficult for smaller, less vigorous plants to survive. The system reproduces the concept of survival of the fittest.

Finding the necessary configuration data for plant species can be difficult. Although optimal water, sun exposure and temperature requirements can be found relatively easily, determining the lower and upper limits is more challenging. A solution to this would be to model plant functional types [MSSH15] rather than individual plant species. Plant functional types encompass a number of species and corresponding resource requirement data can be preconfigured with the help of biologists. Modelling individual plant species could then be done by the user using the parent functional type as a template for parameter specification.

Chapter 8

Conclusion

In this work we design, implement and test a new ecosystem generation pipeline, drawing inspiration from the strengths and weaknesses of existing work. The contribution of this work is the concurrent use of clustering, a simulator and a statistical analysis tool to efficiently and realistically place vegetation on any given terrain. Indeed, simulators are computationally expensive and their processing time heavily dependent on level of detail and simulation area. Using radial distribution analysis permits application of an ecosystem simulator solely as an intermediary tool to derive detailed distribution characteristics. Using clustering permits a single ecosystem simulation to realistically populate a large number of areas on the terrain that are deemed similar in terms of resources. Moreso, clustering permits the user to configure, based on their requirements, the balance between efficiency and realism.
Extensibility is also a strength of this system; By storing species and their properties in a database, by performing plant suitability filtering and by providing a graphical tool to intuitively add species, the system is, by design, able to cater for any number and type of plant species.
Linking back to the research questions elaborated in the introduction of this thesis, we show that procedural methods can indeed be used to automate the generation of vegetation and water networks for virtual worlds. By permitting users to input terrain, resource data, clustering configurations and specie selection, we show it is possible to generate a wide variety of virtual worlds, ranging from tropical rainforrests to alpine resorts. We also found that, by implementing algorithms to take advantage of the parallelism of GPU's, it is possible to optimize the core algorithms sufficiently to obtain near real-time interactions.

8.1 Applications

The main application of this tool would be to quickly and easily create realistic virtual rural worlds for use in video games, simulators or animated movies. Because the inputs are simple

and intuitive, it targets not only professional computer generated imagery (CGI) artists but also hobbyist developers. Limited scope did not permit the implementation of a rendering component to generate the final plant renders. The system acts more as a plug-in to existing game engines or rendering software, therefore, to configure vegetation and water content on the terrain.

With accurate vegetation and resource configurations, it could also be used to determine suitable vegetation distributions given a modelled environment. This could prove useful for biologists, for example, to estimate plant growth. With the ever-increasing issue of climate change and, as a consequence, investment in climate change predictions, this could be a useful tool to help predict the associated affects on vegetation. Although the tool could potentially be used as is for rough estimation, modelling other resources and existing ones in more detail would dramatically improve accuracy and, therefore, applicability (see Section 8.2)

8.2 Limitations and Future Work

The primary limitation of this work is that it cannot be used as an independent virtual world generator but acts more as a tool to be used alongside existing game engines or rendering software. Notably, an essential aspect missing from this work is the ability to generate final renders of the scene with realistic three dimensional plant models. Unfortunately, the time-frame and scope of this work did not permit this.

This work focuses heavily on using procedural methods to generate virtual worlds. Unfortunately, a consequence of this is that fine-control over the final content is lost. A useful extension would be to integrate functionality that provides more user control. Examples include: Permitting users to define areas on the terrain on which they want specific species to appear, an eraser tool to remove all vegetation in selected areas and an override tool to manually place plant instances.

Procedural methods could be extended upon also to improve the resulting realism. A notable example of this is the ability to automatically place snow. Rainfall and temperature information is already known for each terrain vertex. The altitude at which the temperature drops below zero can therefore easily be determined and used to define the snowline. More so, using the sunlight exposure for each terrain vertex, snow meltage can also be simulated accordingly. This would permit slopes with less sun exposure to retain snow for longer, as they do in reality.

Although this work focuses strongly on efficiency in order to maintain continuous interactive feedback, some areas warrant further improvements. As the ecosystem simulator is currently the bottleneck in terms of processing time, much would be gained from improving its efficiency. As simulation area has a strong influence on processing time, one idea would be to make the area vary based on the maximum size attributes of the species it contains. Similarly, the timespan of the simulation could be calculated dynamically based on the ageing properties of contained species.
Another way to improve efficiency would be to enhance the radial distribution caching system. As is, the probability that an ecosystem simulator run has already been performed and, therefore, that it can be bypassed entirely, is very slim. Techniques could be used to improve the cache hit, including: Using a range rather than a fixed value for the resources when searching for valid radial distribution data and a background thread which runs simulations during CPU down-time and fills the radial distribution database accordingly.

The ecosystem simulator models resources and resource availability in great detail in an attempt to generate plausible results. The accuracy of these simulations could be further improved however and, as a consequence, its application domain broadened. For example, some vital elements that have a big impact on plant development are neglected, including: wind, fire, flooding and soil nutrients.
As discussed previously (see Section 6.3.2), soil moisture could be simulated more realistically by modelling the soil with multiple layers. In this way, small plants would not compete for the same soil resources as bigger plants with bigger roots and, therefore, soil reach.

Bibliography

[ACV+14] C. Andújar, A. Chica, M. a. Vico, S. Moya, and P. Brunet. Inexpensive Reconstruction and Rendering of Realistic Roadside Landscapes. *Computer Graphics Forum*, 33(6):101–117, feb 2014.

[BF98] Paul S Bradley and Usama M Fayyad. Refining Initial Points for K-Means Clustering. *ICML*, 98:91–99, 1998.

[BKK+97] FS Berezovskaya, GP Karev, OS Kisliuk, RG Khlebopros, and Yu L Tsel'niker. A fractal approach to computer-analytical modelling of tree crowns. *Trees*, 11(6):323–327, 1997.

[Bli77] James F Blinn. Models of light reflection for computer synthesized pictures. *ACM SIGGRAPH Computer Graphics*, 11(2):192–198, 1977.

[BM07] F Boudon and G Le Moguédec. Déformation asymétrique de houppiers pour la génération de représentations paysageres réalistes. *Revue Electronique Francophone d'Informatique*, 1(1):9–19, 2007.

[BMJ+11] Bedich Beneš, Michel Abdul Massih, Philip Jarvis, Daniel G. Aliaga, and Carlos A. Vanegas. Urban ecosystem design. *Symposium on Interactive 3D Graphics and Games on - I3D '11*, 1(212):167, 2011.

[BPC+12] Frédéric Boudon, Christophe Pradal, Thomas Cokelaer, Przemyslaw Prusinkiewicz, and Christophe Godin. L-py: an L-system simulation framework for modeling plant architecture development based on a dynamic language. *Frontiers in plant science*, 3(May):76, jan 2012.

[DCSD02] Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. *Proceedings of the conference on Visualization '02 (VIS '02)*, 1:219–226, 2002.

[DGGK11] E. Derzapf, Björn Ganster, M. Guthe, and Reinhard Klein. River Networks for Instant Procedural Planets. *Computer Graphics Forum*, 30(7):2031–2040, 2011.

[DHL+98] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomir Měch, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1:275—-286, 1998.

[DP10] 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, jun 2010.

[DSS93] Rudy P Darken, John L Sibert, and Computer Science. A Toolset for Navigation in Virtual Environments. *Proceedings of the 6th annual ACM symposium on User interface software and technology*, 1:157–165, 1993.

[EL99] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 2:1033–1038, 1999.

[Emi14] Arnaud Emilien. *Création interactive de monde virtuels.* PhD thesis, 2014.

[EPCV14] Arnaud Emilien, Pierre Poulin, Marie-Paule Cani, and Ulysse Vimont. Interactive Procedural Modelling of Coherent Waterfall Scenes. In *Computer Graphics Forum*, pages 22–35, 2014.

[EVC+15] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds. In *Siggraph '15, TO BE PUBLISHED*, 2015.

[FP05] Belhadj Fares and Audibert Pierre. Modeling landscapes with ridges and rivers: bottom up approach. *ACM Symposium on Virtual Reality Software and Technology*, 151:1–4, 2005.

[FZS+08] Thierry Fourcaud, Xiaopeng Zhang, Alexia Stokes, Hans Lambers, and Christian Körner. Plant growth modelling and applications: the increasing importance of plant architecture in growth models. *Annals of botany*, 101(8):1053–63, may 2008.

[GFJ+11] Y. Guo, T. Fourcaud, M. Jaeger, X. Zhang, and B. Li. Plant growth and architectural modelling and its applications. *Annals of Botany*, 107(5):723–727, apr 2011.

[GGG+13] Jean-David Génevaux, Éric Galin, Eric Guérin, Adrien Peytavie, and Bedich Beneš. Terrain generation using procedural models based on hydrology. *ACM Transactions on Graphics*, 32(4):1, 2013.

[GMN14] James Gain, Patrick Marais, and Rudolph Neeser. City Sketching. *WSCG 2014*, 1:1–10, 2014.

[GMS09] James Gain, Patrick Marais, and Wolfgang Straßer. Terrain sketching. *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D '09*, 1(212):1–8, 2009.

[Ham01] Johan Hammes. Modeling of Ecosystems as a Data Source for Real-Time Terrain Rendering. *Digital Earth Moving*, 1:98–111, 2001.

[HLT+09] Thomas Hurtut, P-E Landes, Joelle Thollot, Yann Gousseau, Remy Drouillhet, and J-F Coeurjolly. Appearance-guided synthesis of element arrangements by example. *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, 1:51–60, 2009.

[Jai10] Anil K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[KD01] N M Kapolka and D J Dollhopf. Effect of slope gradient and plant growth on soil loss on reconstructed steep slopes. *International Journal of Surface Mining, Reclamation and Environment*, 15(2):86–89, 2001.

[KM07] George Kelly and Hugh McCabe. Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology*, volume 2007, pages 8–16, 2007.

[KMN88] Alex D. Kelley, Michael C. Malin, and Gregory M. Nielson. Terrain simulation using a model of stream erosion. *ACM SIGGRAPH Computer Graphics*, 22(4):263–268, 1988.

[KMN+02] Tapas Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, Ruth Silverman, and a.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[Lau10] Eddie Lau. Visually appealing water flow over a terrain. 2010.

[Lew99] Philip Lewis. Three-dimensional plant modelling for remote sensing simulation studies using the Botanical Plant Modelling System. *Agronomie, EDP Sciences*, 19:185–210, 1999.

[LLX+01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150, jul 2001.

[LP02] Brendan Lane and Przemyslaw Prusinkiewicz. Generating Spatial Distributions for Multilevel Models of Plant Communities. *Interface*, 2002:69–80, 2002.

[LRBP12] Steven Longay, Adam Runions, Frédéric Boudon, and Przemyslaw Prusinkiewicz. TreeSketch : Interactive Procedural Modeling of Trees on a Tablet. *The proceedings of the Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 107–120, 2012.

[MSSH15] G R Moncrieff, S Scheiter, J A Slingsby, and S I Higgins. Understanding global change impacts on South African biomes using Dynamic Vegetation Models. *South African Journal of Botany*, 101(2015):16—-23, 2015.

[PHL+09] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch, and Przemyslaw Prusinkiewicz. Self-organizing tree models for image synthesis. *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, 1(212):1–10, 2009.

[PHM93] Przemyslaw Prusinkiewicz, Mark Hammel, and Eric Mjolsness. Animation of Plant Development. In *SIGGRAPH '93 Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, volume 93, pages 351–360, 1993.

[PI15] Michael Peter and Perrone Ibm. K-Means Clustering for Hidden Markov Models. Number 2000, 2015.

[PL90] ALP Prusinkiewicz and A Lindenmayer. *The Algorithmic Beauty of Plants*. 1990.

[PM01] Yoav I. H. Parish and Pascal Müller. Procedural Modeling of Cities. In *28th annual conference on Computer graphics and interactive techniques*, number 2001, pages 301–308, 2001.

[PMV+12] C. E. Timothy Paine, Toby R. Marthews, Deborah R. Vogt, Drew Purves, Mark Rees, Andy Hector, and Lindsay a. Turnbull. How to fit nonlinear plant growth models and calculate growth rates: an update for ecologists. *Methods in Ecology and Evolution*, 3(2):245–256, apr 2012.

[PNH+14] S. Pirk, T. Niese, T. Hadrich, B. Benes, and O. Deussen. Windy Trees : Computing Stress Response for Developmental Tree Models. *ACM Trans. Graph(ACM Transactions on Graphics)*, 33(6):1–11, 2014.

[Pru96] Przemyslaw Prusinkiewicz. Visual Models of Plants Interacting with Their Environment. In *SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, volume 1, pages 397–410, 1996.

[ŠBBK08] Ondej Šťava, Bedich Beneš, Matthew Brisbin, and Jaroslav Kivánek. Interactive terrain modeling using hydraulic erosion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 201–210, 2008.

[SED03] Cyril Soler, F R Ed, and Philippe Dereffye. An Efficient Instantiation Algorithm for Simulating Radiant Energy Transfer in Plant Models. *ACM Transactions on Graphics (TOG)*, 22(2):204–233, 2003.

[SJM80] TJ Schmugge, TJ Jackson, and HL McKim. Survey of Methods for Soil Moisture Determination. *Water Resources Research*, 16(6):961–979, 1980.

[SKG+09] Ruben M Smelik, Klaas Jan De Kraker, Saskia A Groenewegen, The Hague, Tim Tutenel, and Rafael Bidarra. A Survey of Procedural Methods for Terrain Modelling. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, pages 25—-34, 2009.

[SSBR01] Cyril Soler, FX Sillion, F Blaise, and Philippe De Reffye. A physiological plant growth simulation engine based on accurate radiant energy transfer. Technical report, 2001.

[SWF12] Wei Sun, Junhui Wang, and Yixin Fang. Regularized k-means clustering of high-dimensional data and its asymptotic consistency. *Electronic Journal of Statistics*, 6(April 2011):148–167, 2012.

[Teo08] Soon Tee Teoh. River and coastal action in automatic terrain generation. In *CGVR - Computer Graphics and Virtual Reality*, number 1, pages 3–9, 2008.

[VBHS11] Juraj Vanek, Bedrich Benes, Adam Herout, and Ondrej Stava. Large-Scale Physics-Based Terrain Editing. *Computer Graphics and Applications, IEEE*, 31(0):35—-44, 2011.

[WLKT09] Li-yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the Art in Example-based Texture Synthesis. In *In proceedings of Eurographics 09*, number 2, pages 1–25, 2009.

[Xu05] Rui Xu. Survey of Clustering Algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.

[Yan04] H.-P. Yan. A Dynamic, Architectural Plant Model Simulating Resource-dependent Growth. *Annals of Botany*, 93(5):591–602, mar 2004.

Appendices

Appendix A

Cluster Summary

	1	**2**	**3**	**4**	**5**
Color					
Member count	7233 (0.7%)	203043 (19%)	119853 (11%)	271116 (26%)	447331 (43%)
Slope	18.5967	36.6987	56.9893	18.7228	5.9857
Temperature (Jan)	0	-1	-1	25	23
Temperature (Feb)	3	1	1	23	21
Temperature (Mar)	5	3	3	2	0
Temperature (Apr)	8	6	6	5	3
Temperature (May)	10	8	8	7	5
Temperature (Jun)	13	11	11	10	8
Temperature (Jul)	10	8	8	7	5
Temperature (Aug)	8	6	6	5	3
Temperature (Sep)	5	3	3	2	0
Temperature (Oct)	3	1	1	0	-2
Temperature (Nov)	0	-1	-2	-2	-4
Temperature (Dec)	-2	-4	-4	-5	-7
Illumination (Jan)	7	8	6	9	10
Illumination (Feb)	7	8	6	9	10
Illumination (Mar)	7	8	7	9	10
Illumination (Apr)	7	8	6	9	10
Illumination (May)	7	7	6	9	10
Illumination (Jun)	6	7	5	8	10
Illumination (Jul)	7	7	6	9	10
Illumination (Aug)	7	8	6	9	10

Illumination (Sep)	7	8	7	9	10
Illumination (Oct)	7	8	6	9	10
Illumination (Nov)	7	8	6	9	10
Illumination (Dec)	6	7	5	8	10
Soil Humidity (Jan)	674.9	3.9	1.8	14.3	13.6
Soil Humidity (Feb)	693.0	4.2	1.9	15.5	14.8
Soil Humidity (Mar)	691.8	4.1	1.9	15.1	14.4
Soil Humidity (Apr)	647.6	3.5	1.6	12.6	11.9
Soil Humidity (May)	572.3	2.6	1.3	9.1	8.5
Soil Humidity (Jun)	625.4	3.7	1.8	13.5	12.8
Soil Humidity (Jul)	675.8	4.6	2.1	17.2	16.4
Soil Humidity (Aug)	713.1	5.8	2.6	21.6	20.6
Soil Humidity (Sep)	705.0	5.1	2.3	19.2	18.3
Soil Humidity (Oct)	672.8	4.2	2.0	15.2	14.5
Soil Humidity (Nov)	630.9	3.2	1.6	11.5	10.9
Soil Humidity (Dec)	659.2	3.8	1.8	13.8	13.1

Table A.1: *Clustering test: Cluster summary.*

Appendix B

Species Properties

Name	**Grass**	$\mathbf{S}_{base}$	$\mathbf{S}_{X2}$	$\mathbf{S}_{X3}$	$\mathbf{S}_{slow}$	$\mathbf{S}_{fast}$	$\mathbf{S}_{smallroots}$	$\mathbf{S}_{shadeloving}$
Max Height (cm)	60	1500	1500	1500	1500	1500	1500	50
Max canopy width (cm)	0	1000	2000	3000	2000	2000	3000	0
Max root size (cm)	20	1000	2000	2000	2000	2000	50	30
Age of start of decline (months)	8000	1000	1000	1000	500	300	1000	1000
Maximum age (months)	9000	2000	2000	2000	600	350	2000	2000
Start of prime illumination (h)	8	8	8	8	8	8	8	0
End of prime illumination (h)	12	12	12	12	12	12	12	4
Minimum illumination (h)	5	6	6	6	3	6	6	0
Maximum illumination (h)	15	12	12	12	12	12	12	6
Start of prime humidity (mm)	25	25	25	25	25	25	25	10
End of prime humidity (mm)	45	35	35	35	35	35	35	25

Minimum humidity (mm)	10	15	15	15	15	15	15	5
Maximum humidity (mm)	60	45	45	45	45	45	45	40
Start of prime temperature (degrees)	15	10	10	10	10	10	10	10
End of prime temperature (degrees)	25	20	20	20	20	20	20	20
Minimum temperature (degrees)	0	-5	-5	-5	-5	-5	-5	-10
Maximum temperature (degrees)	40	30	30	30	30	30	30	30
Maximum seeding distance (m)	3	50	50	50	50	50	50	3
Annual seed count	1000	100	100	100	100	100	100	1000

Appendix C

Maximum Distribution Reproduction Areas

Points per 10000m^2	**Maximum reproduction area (m^2)**
5000	1000000
10000	500000
15000	333333
20000	250000
25000	200000
30000	166666
35000	142857
40000	125000
45000	111111
50000	100000
55000	90909
60000	83333
65000	76923
70000	71428
75000	66666
80000	62500
85000	58823
90000	55555
95000	52631
100000	50000

Table C.1: *Maximum reproduction area for given plant densities*

Appendix D

Machine Specifications

CPU	Intel Core i7-4770 CPU @ 3.40GHz
GPU	Nvidia GeForce GTX 770
RAM	
Storage	1TB HDD + 60GB SDD

Appendix E

Modeled rainforest plant species

Name	Brazil nut	Cavendish banana	Heliconia	King of Bromeliads	Orchid
Max Height (cm)	1800	200	280	80	50
Max canopy width (cm)	800	150	200	70	30
Max root size (cm)	200	25	100	100	20
Age of start of decline (months)	1000	600	360	48	400
Maximum age (months)	2500	800	500	60	480
Start of prime illumination (h)	8	8	7	0	0
End of prime illumination (h)	13	13	14	6	7
Minimum illumination (h)	7	7	6	0	0
Maximum illumination (h)	14	15	16	8	8
Start of prime humidity (mm)	150	100	150	75	80
End of prime humidity (mm)	450	500	450	200	350
Minimum humidity (mm)	100	80	100	50	50

Maximum humidity (mm)	600	550	500	420	500
Start of prime temperature (degrees)	15	15	15	15	15
End of prime temperature (degrees)	35	35	35	35	35
Minimum temperature (degrees)	5	-5	5	-1	5
Maximum temperature (degrees)	40	40	45	40	40
Slope start of decline (degrees)	15	20	25	35	35
Max slope(degrees)	25	30	40	45	55
Maximum seeding distance (m)	100	100	100	10	2
Annual seed count	50	50	100	1000	500

Appendix F

Modeled alpine plant species

Name	Spruce	Maple	Moss Campion	Daisy	Beech
Max Height (cm)	250	1200	15	45	900
Max canopy width (cm)	200	500	40	10	900
Max root size (cm)	60	150	10	5	300
Age of start of decline (months)	1500	2000	240	6	1500
Maximum age (months)	2000	3000	360	8	2500
Start of prime illumination (h)	6	6	8	9	8
End of prime illumination (h)	13	10	13	13	13
Minimum illumination (h)	4	4	5	6	6
Maximum illumination (h)	14	13	15	14	15
Start of prime humidity (mm)	30	20	15	15	20
End of prime humidity (mm)	70	40	55	50	45
Minimum humidity (mm)	20	15	10	10	15

Maximum humidity (mm)	85	70	65	70	65
Start of prime temperature (degrees)	-20	-5	-30	10	0
End of prime temperature (degrees)	10	15	20	30	25
Minimum temperature (degrees)	-40	-25	-50	-7	-15
Maximum temperature (degrees)	35	35	35	40	35
Slope start of decline (degrees)	25	15	66	50	10
Max slope(degrees)	50	30	90	75	40
Maximum seeding distance (m)	100	100	2	2	100
Annual seed count	100	25	250	500	50

Appendix G

Extra rainforest plant species modeled

Name	Bahama wild coffee	Bengal Bamboo	Bougain villea	Coconut Palm	Durian
Max Height (cm)	150	1200	220	600	1200
Max canopy width (cm)	100	200	60	200	400
Max root size (cm)	40	300	70	100	400
Age of start of decline (months)	100	1000	60	1100	2000
Maximum age (months)	125	1500	80	1600	3000
Start of prime illumination (h)	0	9	8	8	8
End of prime illumination (h)	6	13	13	13	13
Minimum illumination (h)	0	7	7	6	8
Maximum illumination (h)	7	14	14	14	14
Start of prime humidity (mm)	100	100	100	170	450
End of prime humidity (mm)	270	350	325	475	450
Minimum humidity (mm)	75	70	50	130	100

Maximum humidity (mm)	330	500	400	520	600
Start of prime temperature (degrees)	15	10	20	15	25
End of prime temperature (degrees)	35	30	40	40	40
Minimum temperature (degrees)	5	-5	7	5	15
Maximum temperature (degrees)	45	40	45	45	45
Slope start of decline (degrees)	35	20	60	25	10
Max slope(degrees)	45	30	70	35	20
Maximum seeding distance (m)	15	100	5	100	100
Annual seed count	45	70	250	80	30

Name	**Fern Begonia**	**Kapok Tree**	**Passion flower**	**Poinciana**	**Queens tear**
Max Height (cm)	80	1300	350	150	45
Max canopy width (cm)	30	500	100	100	25
Max root size (cm)	40	600	180	60	10
Age of start of decline (months)	35	3000	90	60	30
Maximum age (months)	45	4500	110	80	40
Start of prime illumination (h)	4	9	9	6	4
End of prime illumination (h)	6	13	13	11	6
Minimum illumination (h)	2	8	8	4	2
Maximum illumination (h)	7	14	14	12	7

Start of prime humidity (mm)	100	200	80	80	80
End of prime humidity (mm)	250	500	200	270	250
Minimum humidity (mm)	50	150	60	50	50
Maximum humidity (mm)	300	600	280	310	350
Start of prime temperature (degrees)	20	10	15	5	0
End of prime temperature (degrees)	40	34	35	30	35
Minimum temperature (degrees)	10	0	5	-5	-5
Maximum temperature (degrees)	45	40	45	35	40
Slope start of decline (degrees)	65	9	35	50	70
Max slope(degrees)	75	18	50	65	80
Maximum seeding distance (m)	5	100	8	20	5
Annual seed count	100	10	75	75	250

Appendix H

Tropical Rainforest with fifteen species: Resulting Cluster Vegetation

Specie	Property	Cluster 1	Cluster 2	Cluster 4	Cluster 6
Brazil Nut	Count	0	0	57	80
	Min height (cm)	-	-	11	11
	Max height (cm)	-	-	1907	2006
	Avg height (cm)	-	-	565 (31%)	710 (39%)
CB	Count	0	0	63	73
	Min height (cm)	-	-	2	2
	Max height (cm)	-	-	215	211
	Avg height (cm)	-	-	76 (38%)	72 (36%)
Heliconia	Count	938	845	83	113
	Min height (cm)	5	1	5	5
	Max height (cm)	298	21	297	293
	Avg height (cm)	141 (50%)	6 (2%)	100 (36%)	118 (42%)
KOB	Count	0	0	214	298
	Min height (cm)	-	-	8	7
	Max height (cm)	-	-	85	86
	Avg height (cm)	-	-	27 (34%)	39 (49%)
Orchid	Count	0	0	161	200
	Min height (cm)	-	-	1	1
	Max height (cm)	-	-	18	33
	Avg height (cm)	-	-	3 (6%)	4 (8%)
BB	Count	0	0	484	542

	Min height (cm)	-	-	8	8
	Max height (cm)	-	-	1329	1219
	Avg height (cm)	-	-	1000 (83%)	839 (70%)
BV	Count	0	0	0	413
	Min height (cm)	-	-	-	15
	Max height (cm)	-	-	-	214
	Avg height (cm)	-	-	-	115 (52%)
CP	Count	451	0	63	67
	Min height (cm)	3	-	3	3
	Max height (cm)	477	-	517	529
	Avg height (cm)	186 (31%)	-	108 (18%)	88 (15%)
Durian	Count	0	0	164	103
	Min height (cm)	-	-	3	1
	Max height (cm)	-	-	496	501
	Avg height (cm)	-	-	150 (13%)	109 (9%)
KT	Count	0	0	0	0
	Min height (cm)	-	-	-	-
	Max height (cm)	-	-	-	-
	Avg height (cm)	-	-	-	-
QT	Count	0	0	0	0
	Min height (cm)	-	-	-	-
	Max height (cm)	-	-	-	-
	Avg height (cm)	-	-	-	-
Poinciana	Count	0	0	0	0
	Min height (cm)	-	-	-	-
	Max height (cm)	-	-	-	-
	Avg height (cm)	-	-	-	-
FB	Count	0	0	0	0
	Min height (cm)	-	-	-	-
	Max height (cm)	-	-	-	-
	Avg height (cm)	-	-	-	-
BWC	Count	0	0	0	0
	Min height (cm)	-	-	-	-
	Max height (cm)	-	-	-	-
	Avg height (cm)	-	-	-	-
PF	Count	0	0	0	0
	Min height (cm)	-	-	-	-

	Max height (cm)	-	-	-	-
	Avg height (cm)	-	-	-	-

Table H.1: *Tropical rainforest: Instance count, minimum height, maximum height and average height for each specie and cluster pair, where: CB is Cavendish Banana; KOB is King of Bromeliads; BB is Bengal Bamboo; BV is Bougain Villea; CP is Coconut Palm; KT is Kapok Tree; QT is Queens Tear; FB is Fern Begonia; BWC is Bahama Wild Coffee and PF is Passion Flower. Note, the percentage following the average height represents the proportion of this values over the species maximum height. It is a good indicator of how well suited the species is to the given cluster.*

Appendix I

Water Flux Compute Shader

```
#version 430

layout (local_size_x = 32, local_size_y = 32) in;

layout(binding = 0, r32f) uniform coherent image2D water_heightmap;
layout(binding = 1, r32f) uniform coherent image2D terrain_heightmap;
layout(binding = 2, r32f) uniform coherent image2D water_heightmap_horizontal_overlaps;
layout(binding = 3, r32f) uniform coherent image2D water_heightmap_vertical_overlaps;
layout(binding = 4, r32f) uniform coherent image2D water_heightmap_corner_overlaps;

ivec2 to_local(in ivec2 global_coordinate_to_translate, in ivec2 global_reference_cell)
{
    return ivec2(global_coordinate_to_translate.x-global_reference_cell.x+1,
                 global_coordinate_to_translate.y-global_reference_cell.y+1);
}

ivec2 to_global(in ivec2 local_coordinate_to_translate, in ivec2 global_reference_cell)
{
    return ivec2(global_reference_cell.x+(local_coordinate_to_translate.x-1),
                 global_reference_cell.y+(local_coordinate_to_translate.y-1));
}

float depth_reduce( in float[9] to_reduce)
{
    float reduced = 0;
    for(int i = 0; i < 9; i++)
```

```
    {
        reduced += to_reduce[i];
    }
    return reduced;
}

void depth_init( inout float[9] to_init)
{
    for(int i = 0; i < 9; i++)
    {
        to_init[i] = 0;
    }
}

const int _BORDER_INSERT_ALL_WATER = -1;

shared float water_to_add_to_cell[gl_WorkGroupSize.x+2][gl_WorkGroupSize.y+2][9];

void main()
{
    uvec2 heightmap_size = imageSize(water_heightmap);
    // WATER HEIGHTMAP IMAGE INDICES
    ivec2 global_idx;
    global_idx.x = int(gl_WorkGroupID.x * gl_WorkGroupSize.x + gl_LocalInvocationID.x); // +
    global_idx.y = int(gl_WorkGroupID.y * gl_WorkGroupSize.y + gl_LocalInvocationID.y); // +

    bool valid = global_idx.x < int(heightmap_size.x) && global_idx.y < int(heightmap_size.y
    bool cell_contains_water = imageLoad(water_heightmap, global_idx).x > 0;

    float water_to_add[3][3];
    float terrain_heights[3][3];
    float water_heights[3][3];
    float aggregate_heights[3][3];
    for(int i = 0; i < 3; i++)
    {
        for(int ii = 0; ii < 3; ii++)
        {
            terrain_heights[i][ii] = 0;
```

```
        water_heights[i][ii] = 0;
        aggregate_heights[i][ii] = 0;
        water_to_add[i][ii] = 0;
    }
}
float water_to_remove = 0;
vec2 global_min;
vec2 global_max;

bool dismiss_all_water = false;

/*************
 * LOAD DATA *
 *************/
if(valid && cell_contains_water)
{
    for(int global_x = global_idx.x-1; global_x <= global_idx.x+1 && !dismiss_all_water
    {
        for(int global_y = global_idx.y-1; global_y <= global_idx.y+1 && !dismiss_all_wa
        {
            uvec2 local_coordinate = to_local(ivec2(global_x, global_y), global_idx);

            /******************
             * TERRAIN HEIGHT *
             ******************/
            float terrain_height = imageLoad(terrain_heightmap, ivec2(global_x+1, global
            terrain_heights[local_coordinate.x][local_coordinate.y] = terrain_height;

            /****************
             * WATER HEIGHT *
             ****************/
            if(global_x < 0 || global_x >= heightmap_size.x || global_y < 0 || global_y
            {
                if(terrain_height < 0)
                {
                    water_to_remove = imageLoad(water_heightmap, global_idx).x;
                    dismiss_all_water = true;
                }
```

```
                else // _BORDER_INSERT_NO_WATER
                {
                    terrain_heights[local_coordinate.x][local_coordinate.y] = 1e20; // N
                    aggregate_heights[local_coordinate.x][local_coordinate.y] = 1e20;
                }
            }
            else
            {
                float water_height = imageLoad(water_heightmap, ivec2(int(global_x), int
                water_heights[local_coordinate.x][local_coordinate.y] = water_height;
                aggregate_heights[local_coordinate.x][local_coordinate.y] = water_height
            }
        }
    }
}

/*****************************
 * CALCULATE WATER MOVEMENT *
 *****************************/
if(valid && cell_contains_water && !dismiss_all_water)
{
    // Now check whether or not the surrounding cells can take all water from current ce
    float aggregate_height_diff = 0;
    float height_diffs[3][3]; // Diff between this cells *TERRAIN* height and surroundin
    uvec2 lowest_surrounding_cell = uvec2(1,1);
    float maximum_height_diff = 0;
    for(uint local_x = 0; local_x < 3; local_x++)
    {
        for(uint local_y = 0; local_y < 3; local_y++)
        {
            float height_diff = max(0, terrain_heights[1][1] - aggregate_heights[local_x
            height_diffs[local_x][local_y] = height_diff;
            aggregate_height_diff += height_diff;
            if(height_diff > maximum_height_diff)
            {
                maximum_height_diff = height_diff;
                lowest_surrounding_cell = uvec2(local_x, local_y);
            }
```

```
    }
}


/*
Two possible scenarios:
    1. There is space to evacuate all water into surrounding cells - Split water cor
    2. There is not enough space to evacuate water into surrounding cells - Add most
*/
if(aggregate_height_diff >= water_heights[1][1]) // Scenario 1
{
    for(uint local_x = 0; local_x < 3; local_x++)
    {
        for(uint local_y = 0; local_y < 3; local_y++)
        {
            float movement_percentage = height_diffs[local_x][local_y]/aggregate_hei
            float water_movement = movement_percentage * water_heights[1][1];
            water_to_add[local_x][local_y] = water_movement;
        }
    }
    water_to_remove = water_heights[1][1]; // Remove all water from cell
}
else // Scenario 2
{
    bool cells_in_which_to_insert_water[3][3];
    // First set all cells with lower height to take water
    for(uint local_x = 0; local_x < 3; local_x++)
    {
        for(uint local_y = 0; local_y < 3; local_y++)
        {
            cells_in_which_to_insert_water[local_x][local_y] = aggregate_heights[loc
        }
    }

    bool balance_found = false;
    while(!balance_found)
    {
        float total_height = aggregate_heights[1][1];
```

```
                uint cell_count = 1;
                water_to_remove = 0;
                // First calculate the target balance_height based on cells which are curren
                for(uint local_x = 0; local_x < 3; local_x++)
                {
                    for(uint local_y = 0; local_y < 3; local_y++)
                    {
                        water_to_add[local_x][local_y] = 0;
                        if(cells_in_which_to_insert_water[local_x][local_y])
                        {
                            total_height += aggregate_heights[local_x][local_y];
                            cell_count++;
                        }
                    }
                }

                // Calculate the balance height
//                uint dividable_total_height = total_height;
//                while(dividable_total_height > 0 && dividable_total_height % cell_count !=
//                {
//                    dividable_total_height--;
//                }
                float target_balance_height = total_height/cell_count;
                balance_found = true; // Start positive
                for(uint local_x = 0; local_x < 3; local_x++)
                {
                    for(uint local_y = 0; local_y < 3; local_y++)
                    {
                        if(cells_in_which_to_insert_water[local_x][local_y])
                        {
                            if(target_balance_height > aggregate_heights[local_x][local_y])
                            {
                                float water_movement = target_balance_height-aggregate_heigh
                                water_to_remove += water_movement;
                                water_to_add[local_x][local_y] = water_movement;
                            }
                            else
                            {
```

```
                            cells_in_which_to_insert_water[local_x][local_y] = false;
                            balance_found = false;
                        }
                    }
                }
            }
        }
    }
}

/*************************
 * INIT NON-BORDER CELLS *
 *************************/
depth_init(water_to_add_to_cell[gl_LocalInvocationID.x+1][gl_LocalInvocationID.y+1]);
/**********************
 * INIT BORDER CELLS *
 **********************/
// VERTICAL BORDER INIT
if(gl_LocalInvocationID.x == 0)
{
    depth_init(water_to_add_to_cell[0][gl_LocalInvocationID.y+1]); // LEFT
    depth_init(water_to_add_to_cell[gl_WorkGroupSize.x+1][gl_LocalInvocationID.y+1]); //
}
// HORIZONTAL BORDER INIT
if(gl_LocalInvocationID.y == 0)
{
    depth_init(water_to_add_to_cell[gl_LocalInvocationID.x+1][0]); // TOP
    depth_init(water_to_add_to_cell[gl_LocalInvocationID.x+1][gl_WorkGroupSize.y+1]); //
}
// CORNERS
if(gl_LocalInvocationID.x == 0 && gl_LocalInvocationID.y == 0)
{
    depth_init(water_to_add_to_cell[0][0]); // TOP LEFT
    depth_init(water_to_add_to_cell[gl_WorkGroupSize.x+1][0]); // TOP RIGHT
    depth_init(water_to_add_to_cell[0][gl_WorkGroupSize.y+1]); // BOTTOM LEFT
    depth_init(water_to_add_to_cell[gl_WorkGroupSize.x+1][gl_WorkGroupSize.y+1]); // BO
}
// Sync threads before writing to ensure all cells addition data has been zeroified
```

```
barrier();
memoryBarrierShared();

/***********************
 * WRITE ADDITION DATA *
 ***********************/
if(valid && cell_contains_water)
{
    // Add the water to surrounding cells
    for(int local_x = 0; local_x < 3; local_x++)
    {
        for(int local_y = 0; local_y < 3; local_y++)
        {
            int dest_x = int(gl_LocalInvocationID.x) + local_x;
            int dest_y = int(gl_LocalInvocationID.y) + local_y;
            int depth = 0;
            {
                int dest_x_depth = 2 - local_x;
                int dest_y_depth = 2 - local_y;
                depth = (dest_y_depth * 3) + dest_x_depth;
            }
            water_to_add_to_cell[dest_x][dest_y][depth] = water_to_add[local_x][local_y]
        }
    }
}
// Ensure its all written
barrier();
memoryBarrierShared();

// Perform addition data on this thread
float water_balance = 0;
if(valid)
{
    water_balance = depth_reduce(water_to_add_to_cell[gl_LocalInvocationID.x+1][gl_Local
}

// Perform the actual write for this thread
/****************************************
```

```
 * WRITE REMOVAL DATA IN NON-BORDER CELLS *
 ****************************************/
if(valid)
{
    imageStore(water_heightmap, global_idx, vec4(imageLoad(water_heightmap, global_idx)
}

/*************************************
 * WRITE REMOVAL DATA IN BORDER CELLS *
 *************************************/
// VERTICAL BORDERS
if(gl_LocalInvocationID.x == 0 && gl_WorkGroupID.x > 0) // LEFT
{
    water_balance = depth_reduce(water_to_add_to_cell[0][gl_LocalInvocationID.y+1]);
    ivec2 border_index = ivec2(gl_WorkGroupID.x*2, global_idx.y);

    imageStore(water_heightmap_vertical_overlaps, border_index, vec4(water_balance,0,0,0
}
if(gl_LocalInvocationID.x == gl_WorkGroupSize.x-1 && global_idx.x < heightmap_size.x-gl_
{
    water_balance = depth_reduce(water_to_add_to_cell[gl_WorkGroupSize.x+1][gl_LocalInvo
    ivec2 border_index = ivec2(gl_WorkGroupID.x*2+1, global_idx.y);

    imageStore(water_heightmap_vertical_overlaps, border_index, vec4(water_balance,0,0,0
}
// HORIZONTAL BORDERS
if(gl_LocalInvocationID.y == 0 && gl_WorkGroupID.y > 0) // TOP
{
    water_balance = depth_reduce(water_to_add_to_cell[gl_LocalInvocationID.x+1][0]);
    ivec2 border_index = ivec2(global_idx.x, gl_WorkGroupID.y*2);
    imageStore(water_heightmap_horizontal_overlaps, border_index, vec4(water_balance,0,0
}
if(gl_LocalInvocationID.y == gl_WorkGroupSize.y-1 && global_idx.y < heightmap_size.y-gl_
{
    water_balance = depth_reduce(water_to_add_to_cell[gl_LocalInvocationID.x+1][gl_WorkG
    ivec2 border_index = ivec2(global_idx.x, gl_WorkGroupID.y*2+1);
    imageStore(water_heightmap_horizontal_overlaps, border_index, vec4(water_balance,0,0
}
```

```
    // TOP CORNERS
    if(gl_LocalInvocationID.x == 0 && gl_LocalInvocationID.y == 0 &&
         gl_WorkGroupID.x > 0 && gl_WorkGroupID.y > 0) // TOP LEFT
    {
        water_balance = depth_reduce(water_to_add_to_cell[0][0]);

        ivec2 border_index = ivec2(gl_WorkGroupID.x*2, gl_WorkGroupID.y*2);
        imageStore(water_heightmap_corner_overlaps, border_index, vec4(water_balance,0,0,0)
    }
    if(gl_LocalInvocationID.x == gl_WorkGroupSize.x-1 && gl_LocalInvocationID.y == 0 &&
         global_idx.x < heightmap_size.x-gl_WorkGroupSize.x && gl_WorkGroupID.y > 0) // TOP
    {
        water_balance = depth_reduce(water_to_add_to_cell[gl_WorkGroupSize.x+1][0]);

        ivec2 border_index = ivec2(gl_WorkGroupID.x*2+1, gl_WorkGroupID.y*2);
        imageStore(water_heightmap_corner_overlaps, border_index, vec4(water_balance,0,0,0)
    }
    // BOTTOM CORNERS
    if(gl_LocalInvocationID.x == 0 && gl_LocalInvocationID.y == gl_WorkGroupSize.y-1 &&
         gl_WorkGroupID.x > 0 && global_idx.y < heightmap_size.y-gl_WorkGroupSize.y) // BOTT
    {
        water_balance = depth_reduce(water_to_add_to_cell[0][gl_WorkGroupSize.y+1]);
        ivec2 border_index = ivec2(gl_WorkGroupID.x*2, gl_WorkGroupID.y*2+1);
        imageStore(water_heightmap_corner_overlaps, border_index, vec4(water_balance,0,0,0)
    }
    if(gl_LocalInvocationID.x == gl_WorkGroupSize.x-1 && global_idx.x < heightmap_size.x-gl_
        gl_LocalInvocationID.y == gl_WorkGroupSize.y-1 && global_idx.y < heightmap_size.y-gl
    {
        water_balance = depth_reduce(water_to_add_to_cell[gl_WorkGroupSize.x+1][gl_WorkGroup
        ivec2 border_index = ivec2(gl_WorkGroupID.x*2+1, gl_WorkGroupID.y*2+1);
        imageStore(water_heightmap_corner_overlaps, border_index, vec4(water_balance,0,0,0)
    }

    // Sync threads to ensure all writes have been performed
    barrier();
    memoryBarrierImage();
}
```

Appendix J

Results: Tropical Species Suitability Graphs

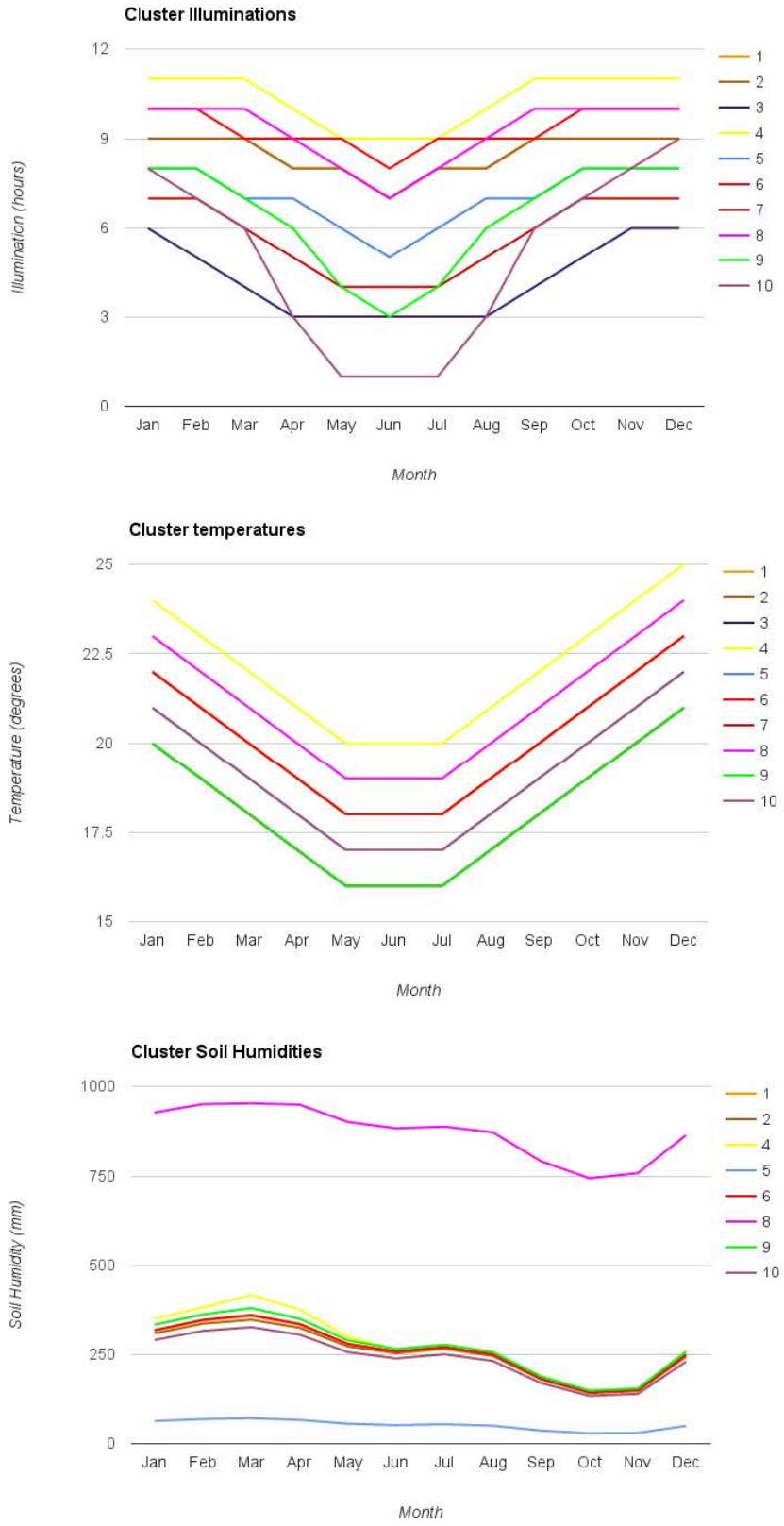

Figure J.1: *Tropical rainforest: Monthly sun exposure (top), temperatures (middle) and soil moistures (bottom) for each terrain cluster. Note that soil humidity data is removed for clusters (3 and 7) as the corresponding values are too small.*

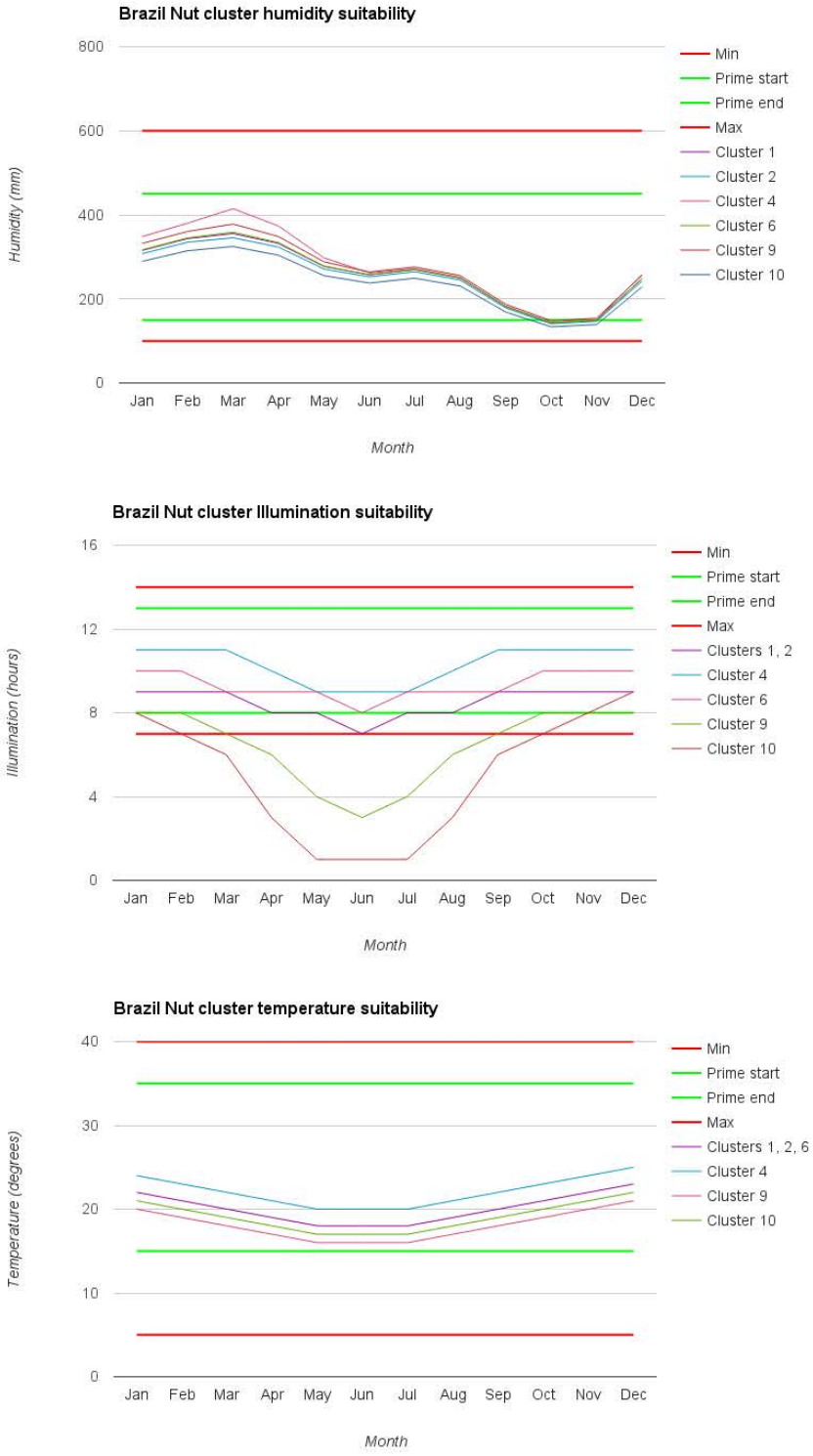

Figure J.2: *Tropical rainforest: Brazil nut suitability to clusters 1, 2, 4, 6, 9 and 10 in terms of soil moisture (top), illumination (middle) and temperature. The thick green lines and red lines delimit the species prime range and absolute limits, respectively.*

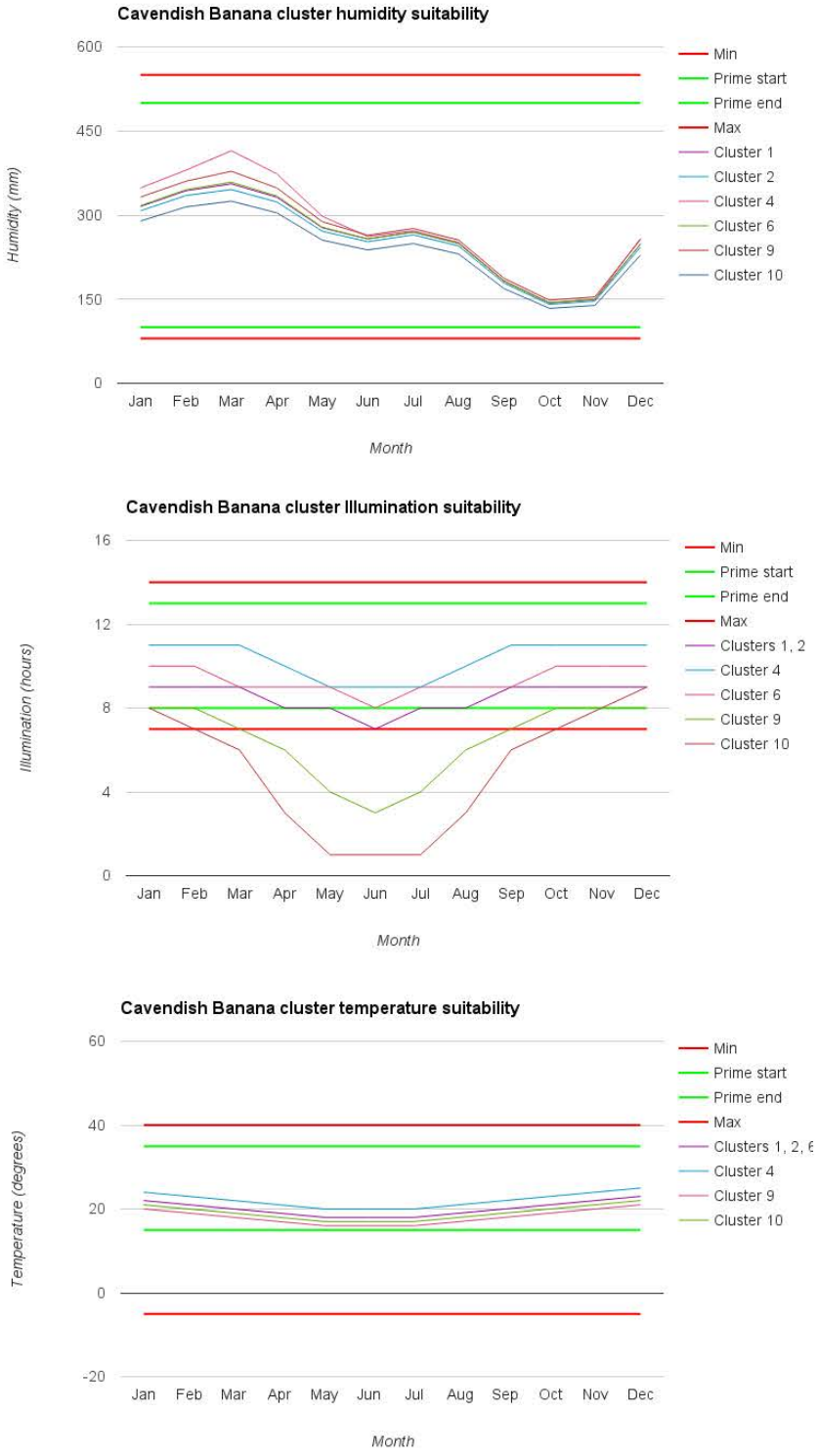

Figure J.3: *Tropical rainforest: Cavendish Banana suitability to clusters 1, 2, 4, 6, 9 and 10 in terms of soil moisture (top), illumination (middle) and temperature. The thick green lines and red lines delimit the species prime range and absolute limits, respectively.*

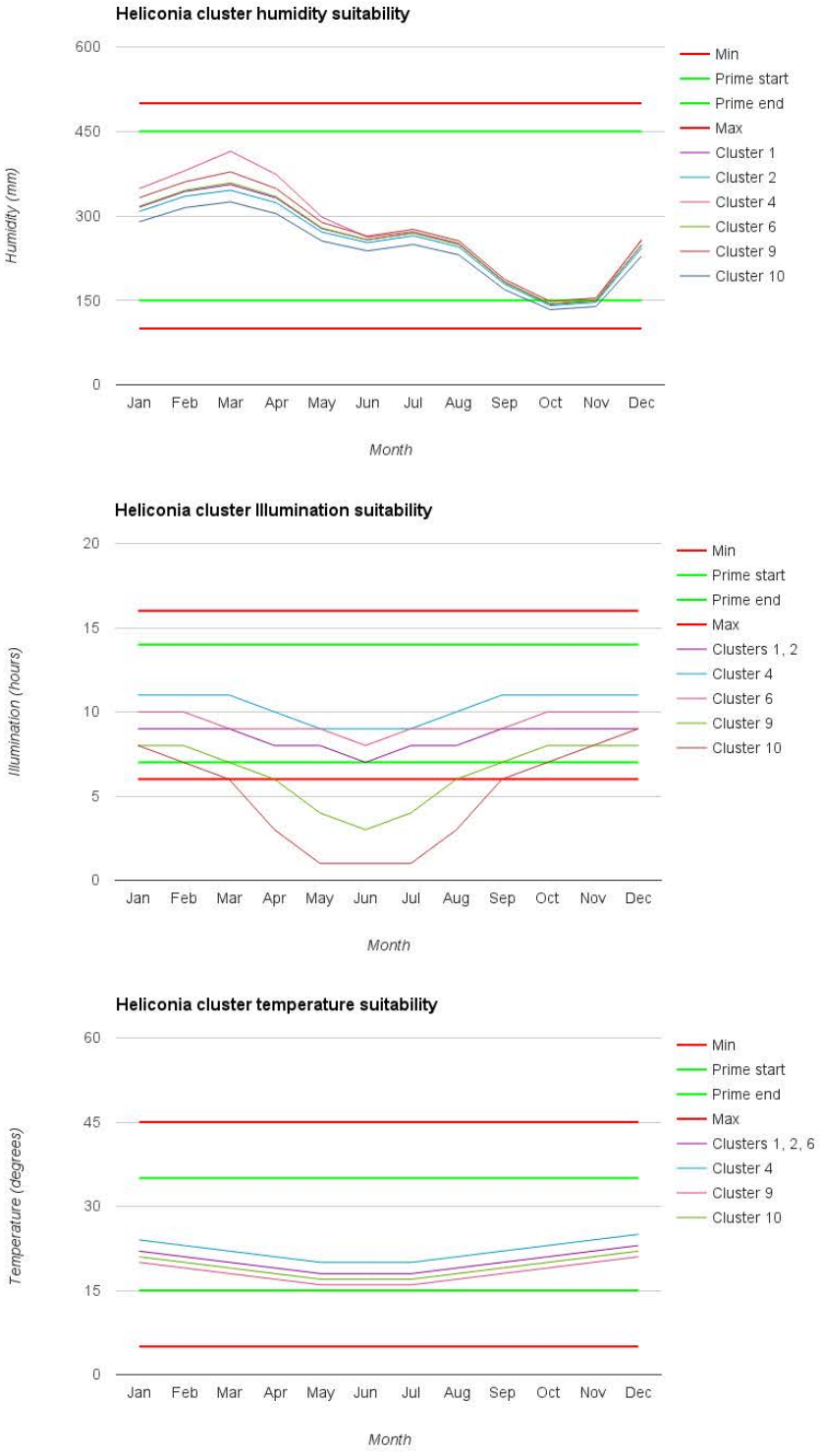

Figure J.4: *Tropical rainforest: Heliconia suitability to clusters 1, 2, 4, 6, 9 and 10 in terms of soil moisture (top), illumination (middle) and temperature. The thick green lines and red lines delimit the species prime range and absolute limits, respectively.*

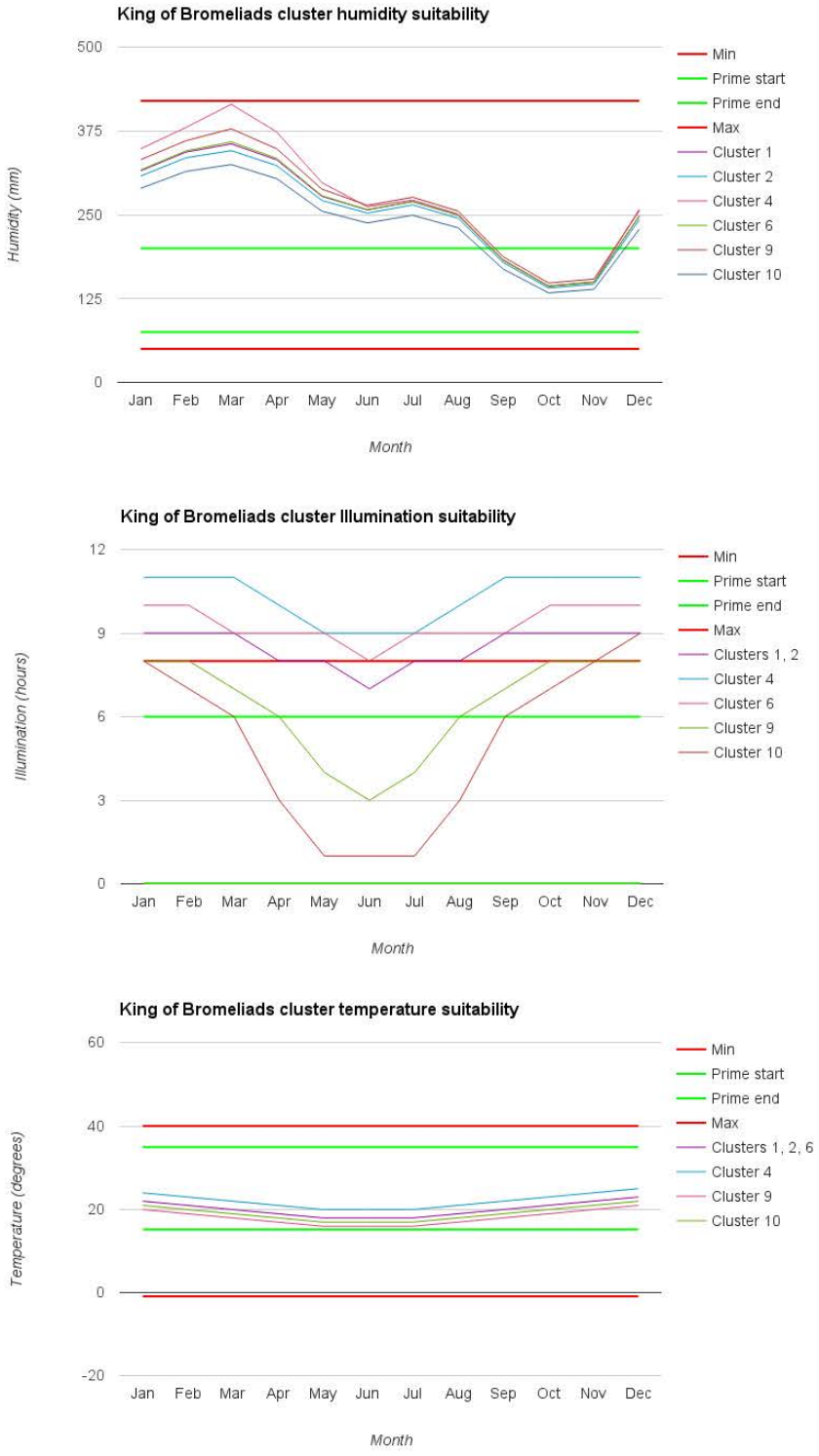

Figure J.5: *Tropical rainforest: King of Bromeliads suitability to clusters 1, 2, 4, 6, 9 and 10 iin terms of soil moisture (top), illumination (middle) and temperature. The thick green lines and red lines delimit the species prime range and absolute limits, respectively. Note that because the king of bromeliads is a shade-loving species, the minimum illumination line is not present as it is overlapped by the start of prime range line.*

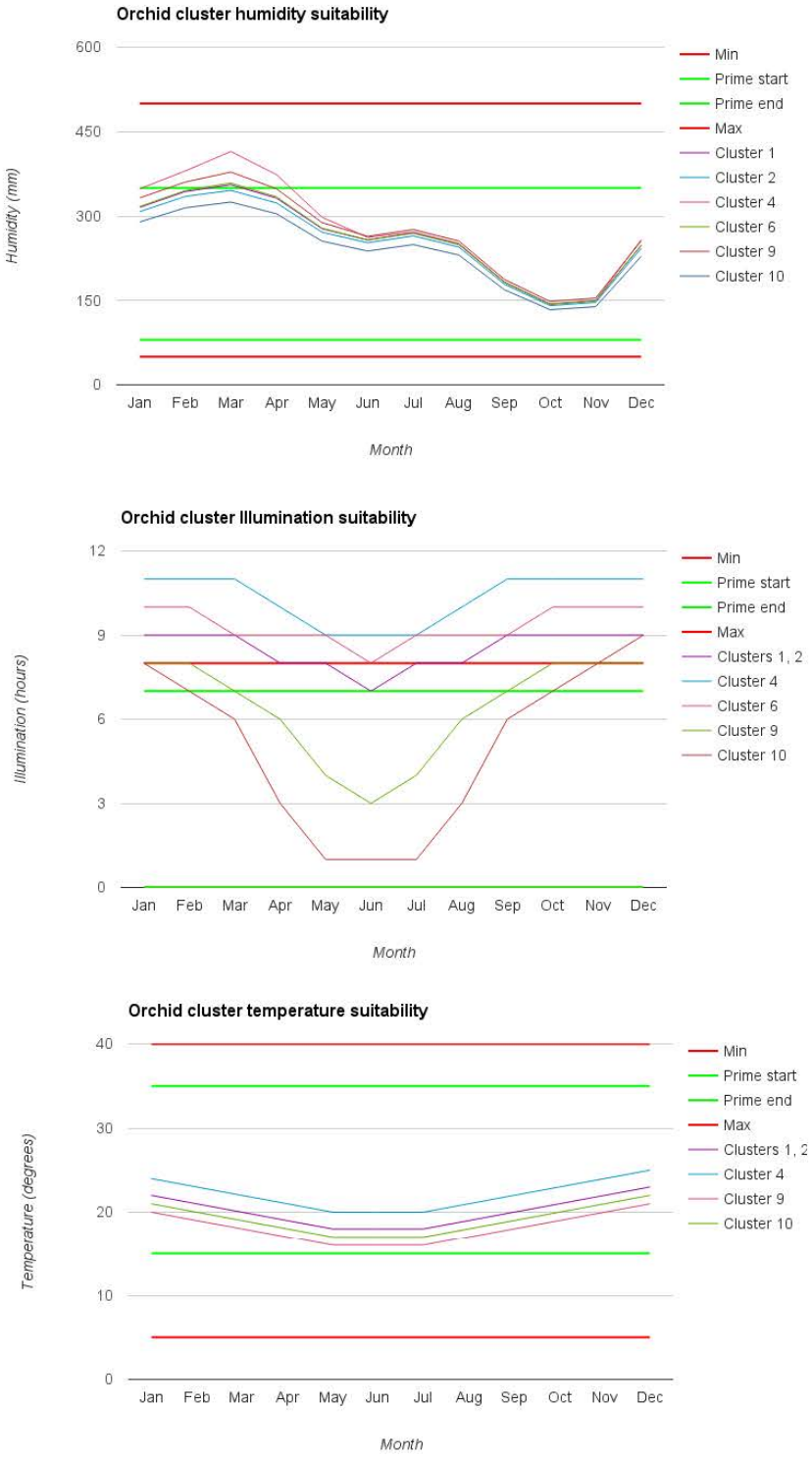

Figure J.6: *Tropical rainforest: Orchid suitability to clusters 1, 2, 4, 6, 9 and 10 in terms of soil moisture (top), illumination (middle) and temperature. The thick green lines and red lines delimit the species prime range and absolute limits, respectively. Note that because the king of bromeliads is a shade-loving specie, the minimum illumination line is not present as it is overlapped by the start of prime range line.*

Appendix K

Results: Alpine Species Suitability Graphs

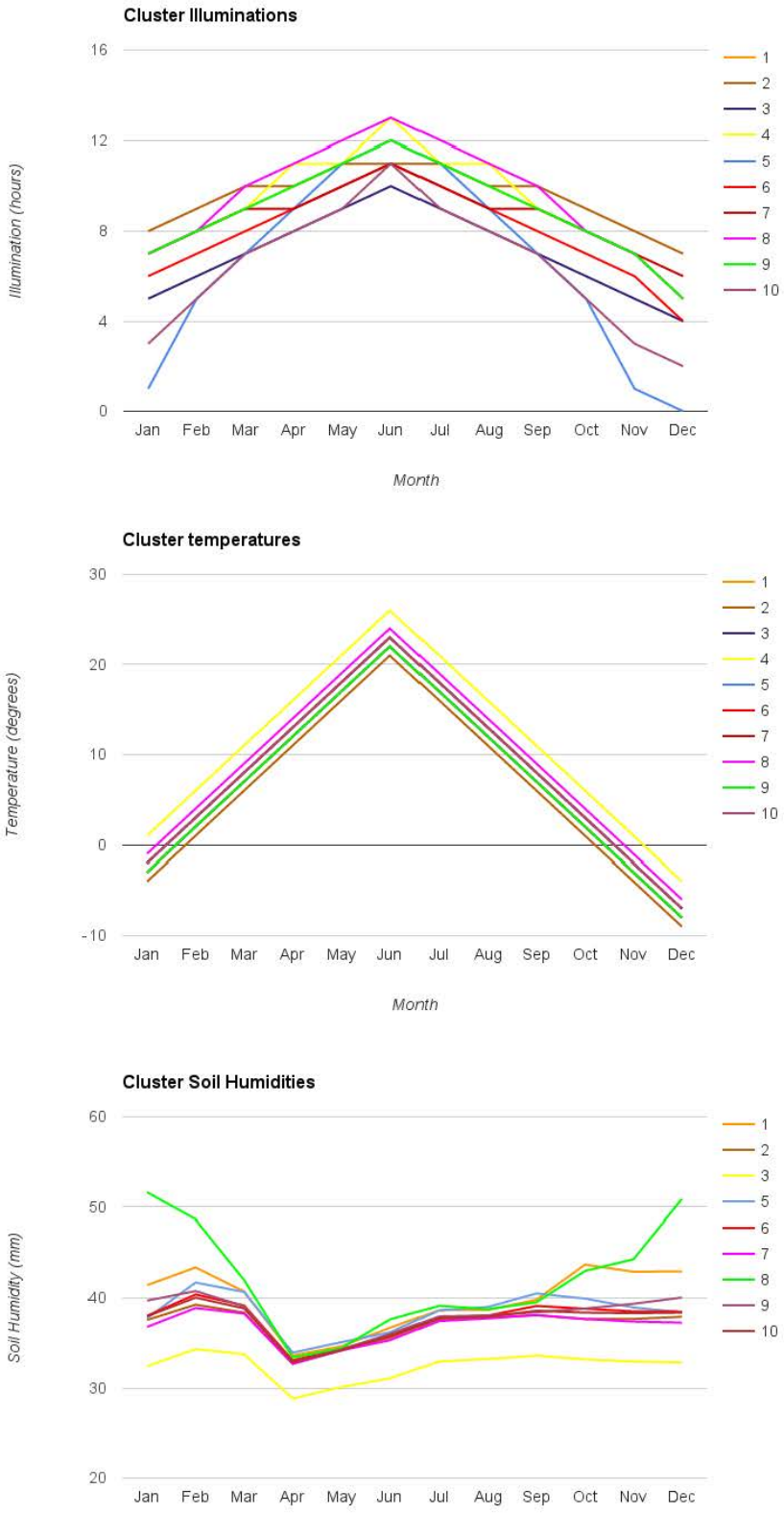

Figure K.1: *Alpine: Mean monthly sun exposure (top), temperatures (middle) and soil moistures (bottom) for each terrain cluster. Note that soil humidity data is removed for cluster 4 because the values are too high.*

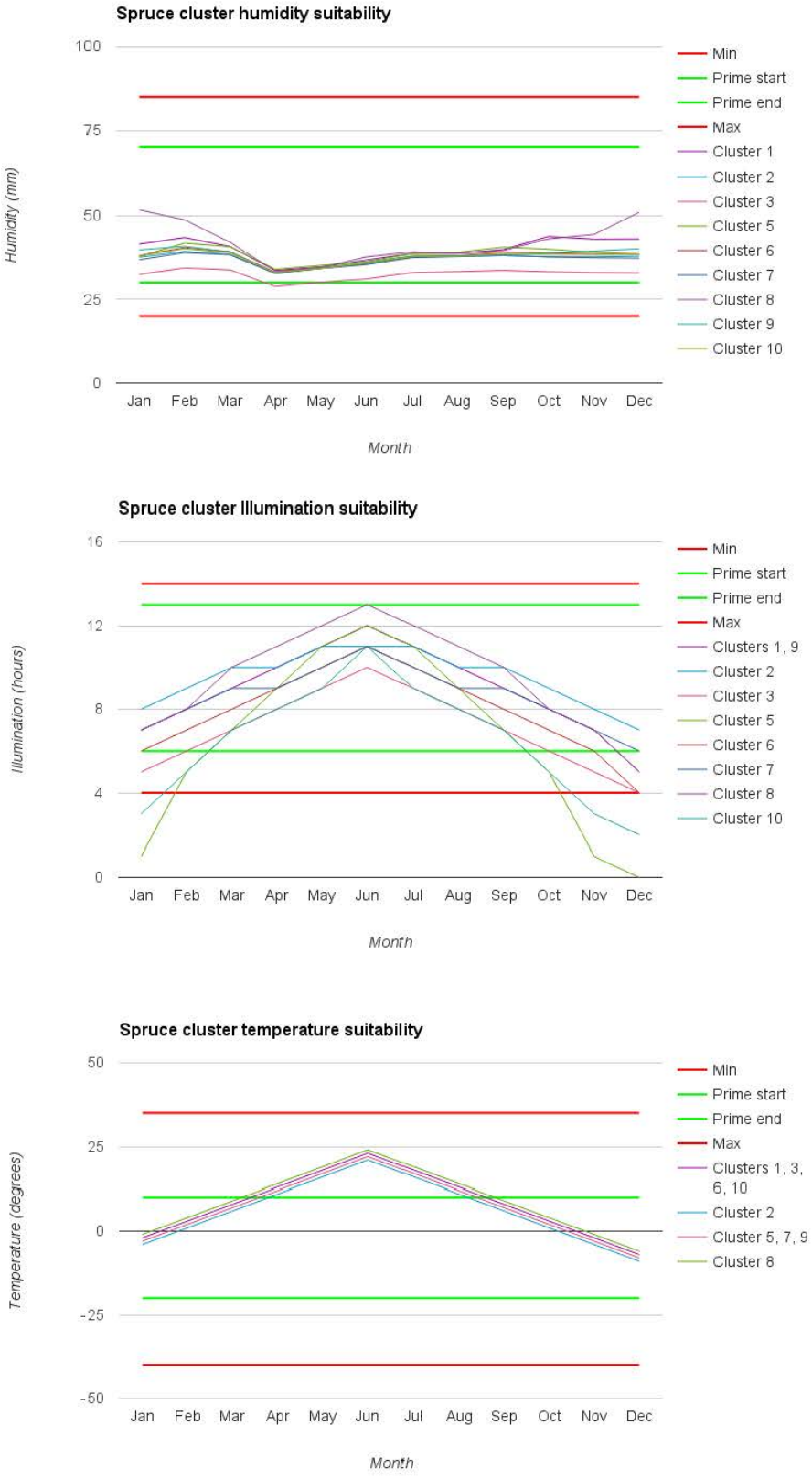

Figure K.2: *Alpine: Spruce suitability to clusters 1, 2, 3, 5, 6, 7, 8, 9 and 10 in terms of temperature (top-left), illumination (top-right) and soil humidity (bottom). The thick green lines and red lines delimit the species prime range and absolute limits respectively.*

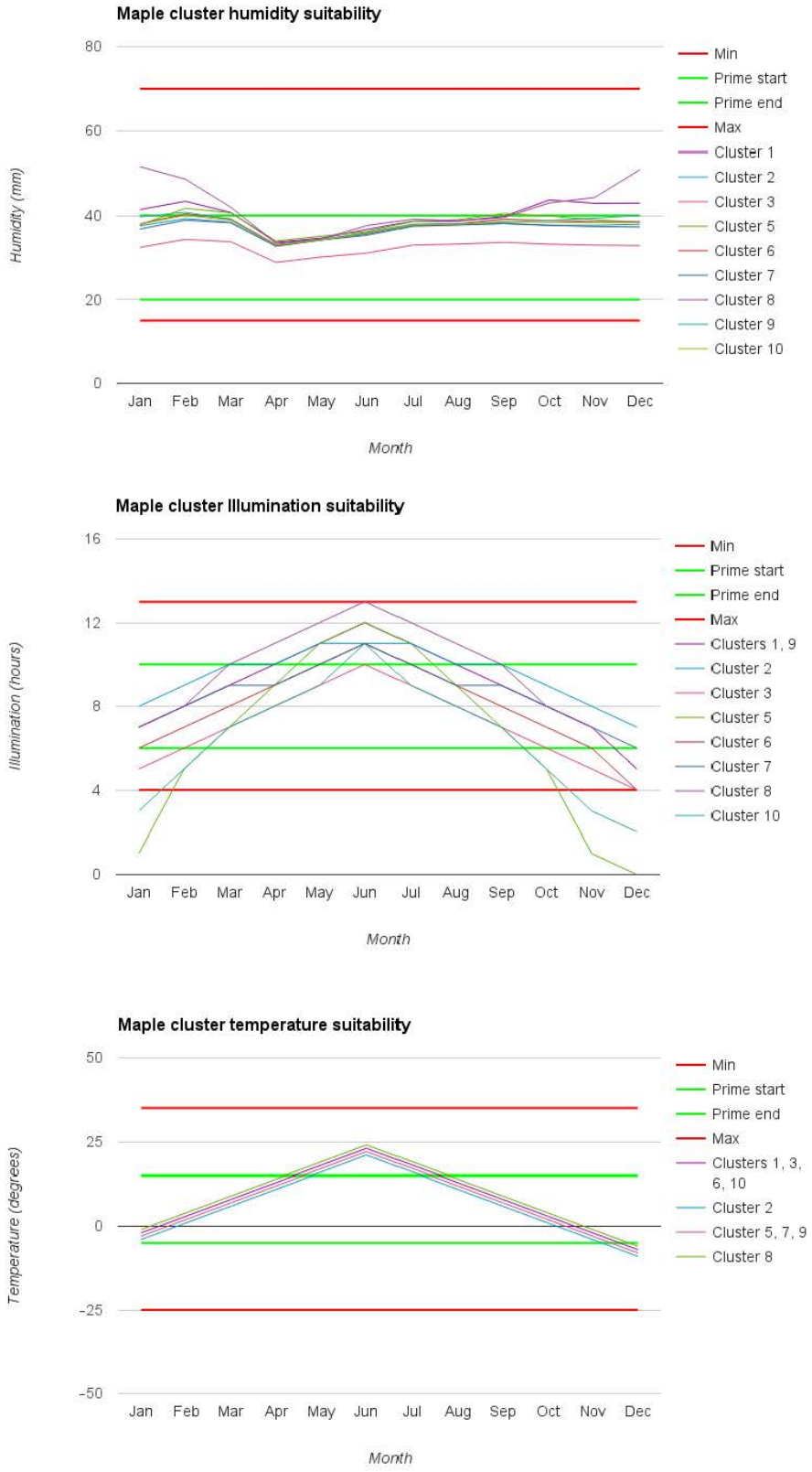

Figure K.3: *Alpine: Maple suitability to clusters 1, 2, 3, 5, 6, 7, 8, 9 and 10 in terms of temperature (top-left), illumination (top-right) and soil humidity (bottom). The thick green lines and red lines delimit the species prime range and absolute limits respectively.*

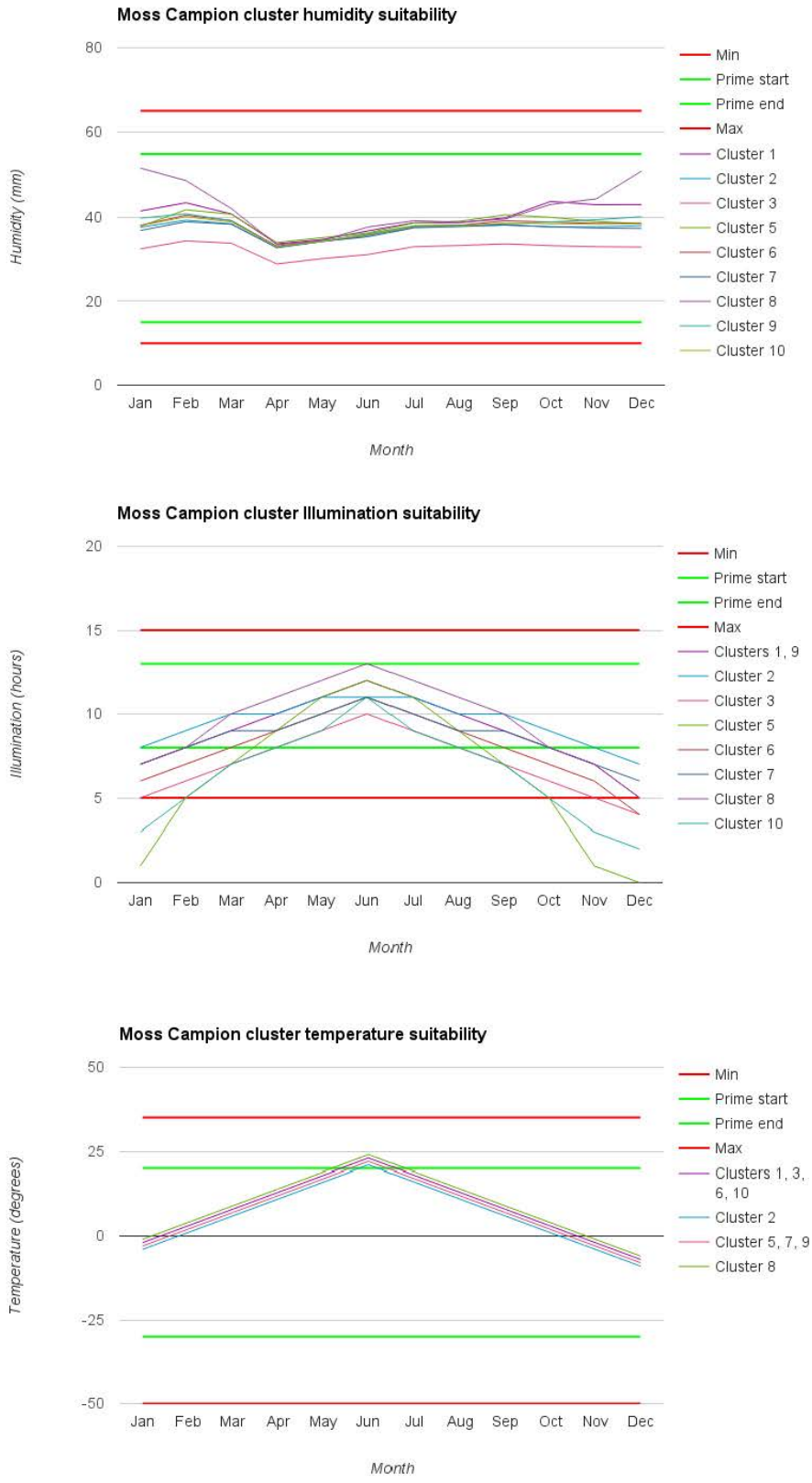

Figure K.4: *Alpine: Moss Campion suitability to clusters 1, 2, 3, 5, 6, 7, 8, 9 and 10 in terms of temperature (top-left), illumination (top-right) and soil humidity (bottom). The thick green lines and red lines delimit the species prime range and absolute limits respectively.*

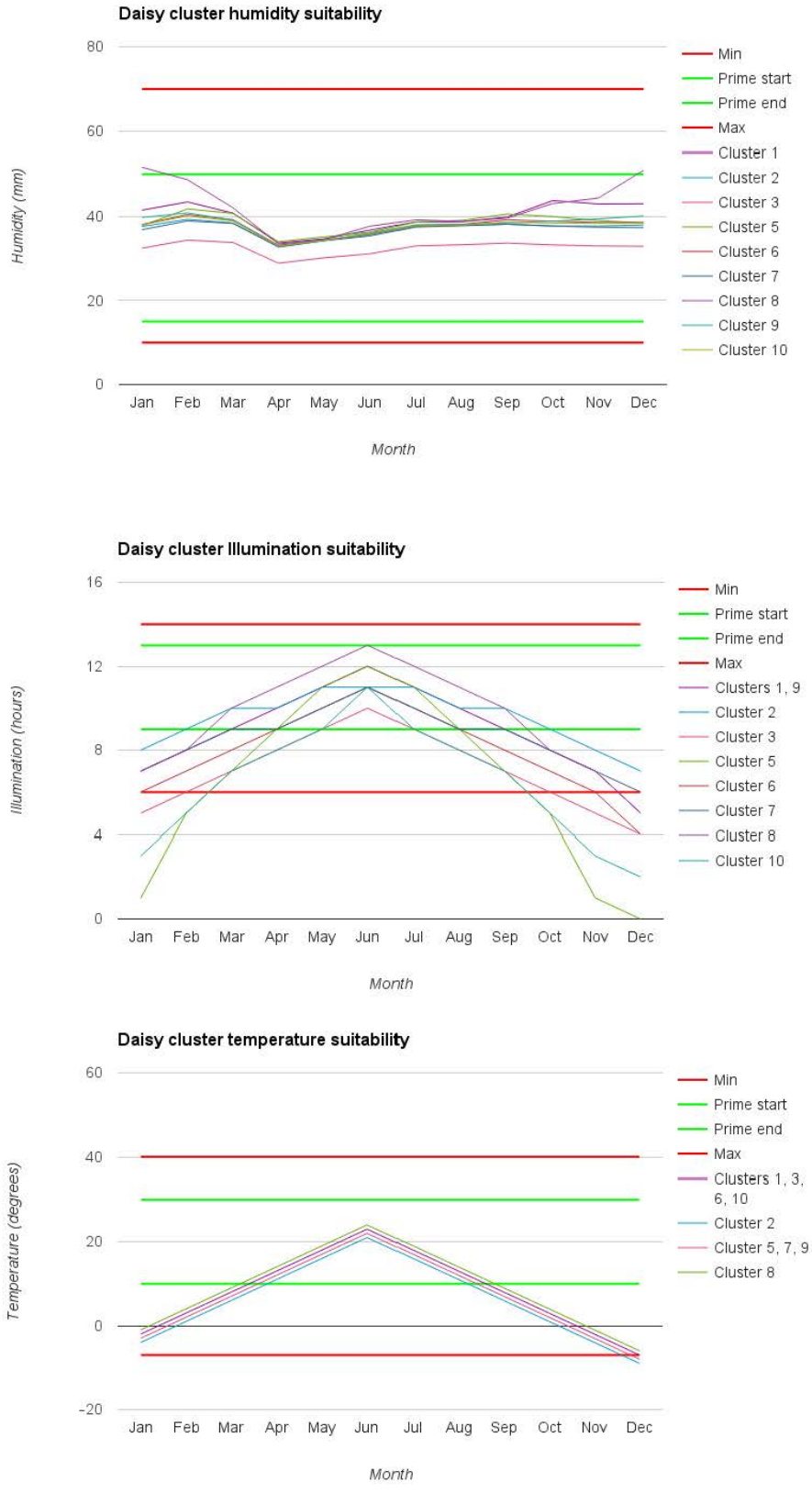

Figure K.5: *Alpine: Daisy suitability to clusters 1, 2, 3, 5, 6, 7, 8, 9 and 10 in terms of temperature (top-left), illumination (top-right) and soil humidity (bottom). The thick green lines and red lines delimit the species prime range and absolute limits respectively.*

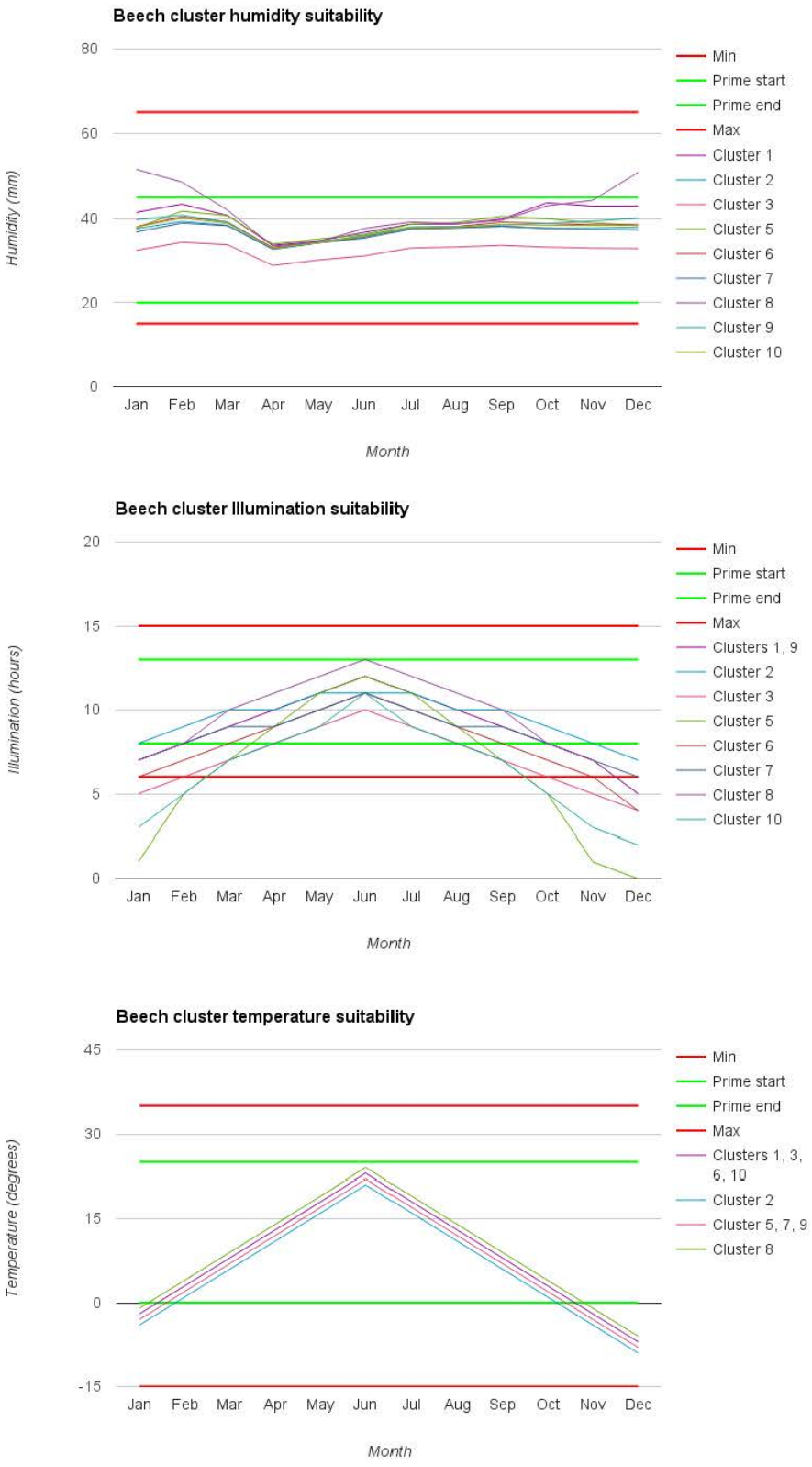

Figure K.6: *Alpine: Beech suitability to clusters 1, 2, 3, 5, 6, 7, 8, 9 and 10 in terms of temperature (top-left), illumination (top-right) and soil humidity (bottom). The thick green lines and red lines delimit the species prime range and absolute limits respectively.*