Using procedural methods to generate 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.

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, water streams, rivers, water reserves, soil, rocks, buildings and road networks for large virtual worlds "by hand" 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 getting worse.

A popular technique to overcome the burden of repetitive tasks is to have them automated. This involves generating algorithms which, given a set of input parameters, generate the required content automatically. This is called *procedural content generation* and has already been successfully applied in different areas of computer graphics including: the generation of non-repetitive textures [**?** **?** **?**], modelling plants [**?** FZS^{+}08**?** **?**], generating terrains [**?** **?** **?**], generating river networks [**?** **?**] and generating city landscapes [**?** **?** **?**] (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 comes down to iterative trial-and-errors until a "close enough" 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 [**?**] to sketch-based recognition algorithms [**?**].

The intent of this thesis is to develop procedural algorithms to automate the generation of virtual rural worlds. The input parameters for the procedural algorithms must be interactive and/or self-explanatory.

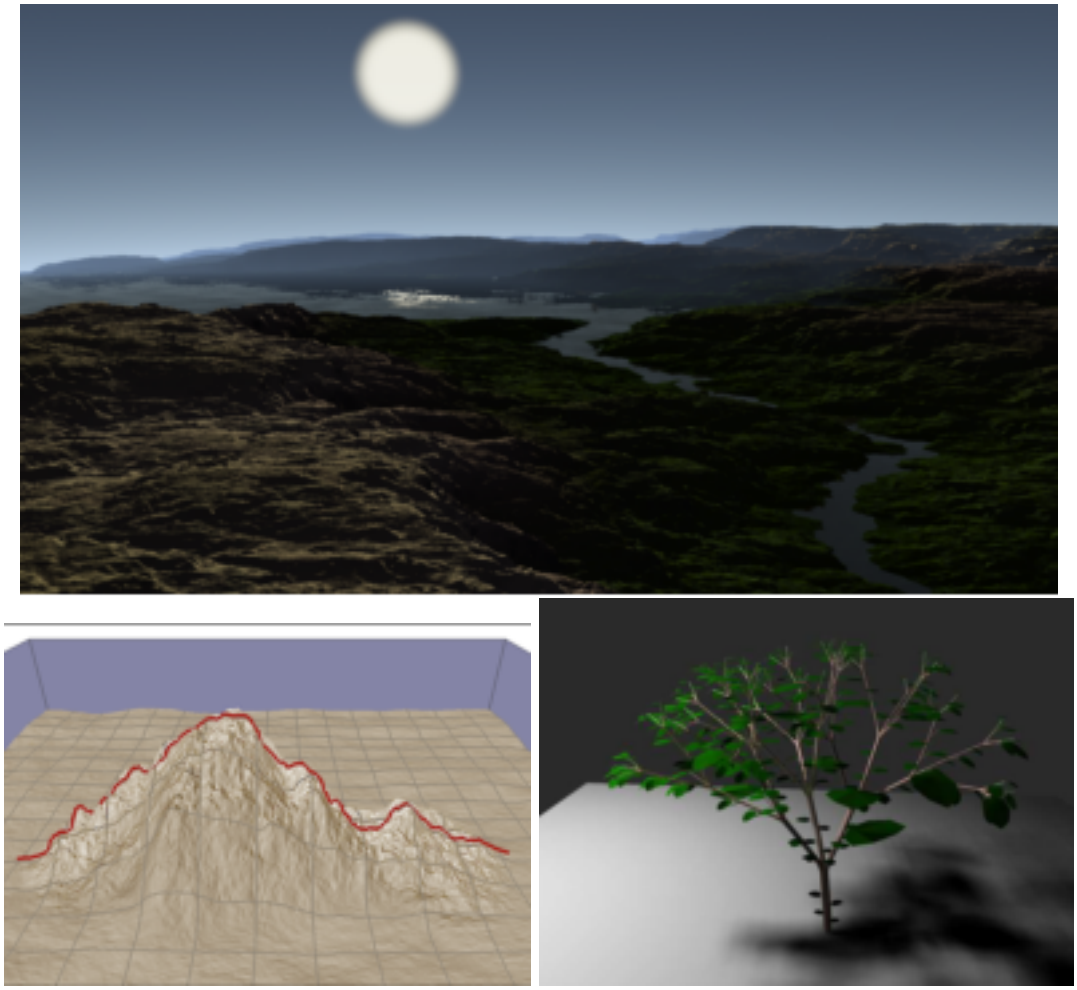

Figure 1.1: Example of procedurally generated content. From top to bottom, left to right: Procedurally generated river stream [**?**], procedurally generated terrain through sketching [**?**], procedurally generated plant [**?**]

1.1 Research Goals

The research goals for this project are as follows:

- Develop procedural methods to automate the generation of realistic virtual rural worlds.
- Provide intuitive and smart controls.
- When possible, make interactions real-time.

One of the most important aspect of rural landscapes is vegetation. As such, our *first goal* must strongly focus on the insertion 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.

For the *second goal*, lots of thought must be put into making all user oriented controls intuitive. To do so, it will be important to research the pros and cons of other graphical applications in terms of control. If need be, multiple prototype controls should be developed in an attempt to find the best suited.

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 *third goal* therefore, efficient algorithms must be developed in order to keep there time complexity to a minimum. When suited, these algorithms should be developed to run on the GPU.

1.2 Contributions

State contributions of this thesis

1.3 Structure

Outline structure of the thesis

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 an exhaustive 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 & 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 a dedicated section: *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 [**?**] 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.

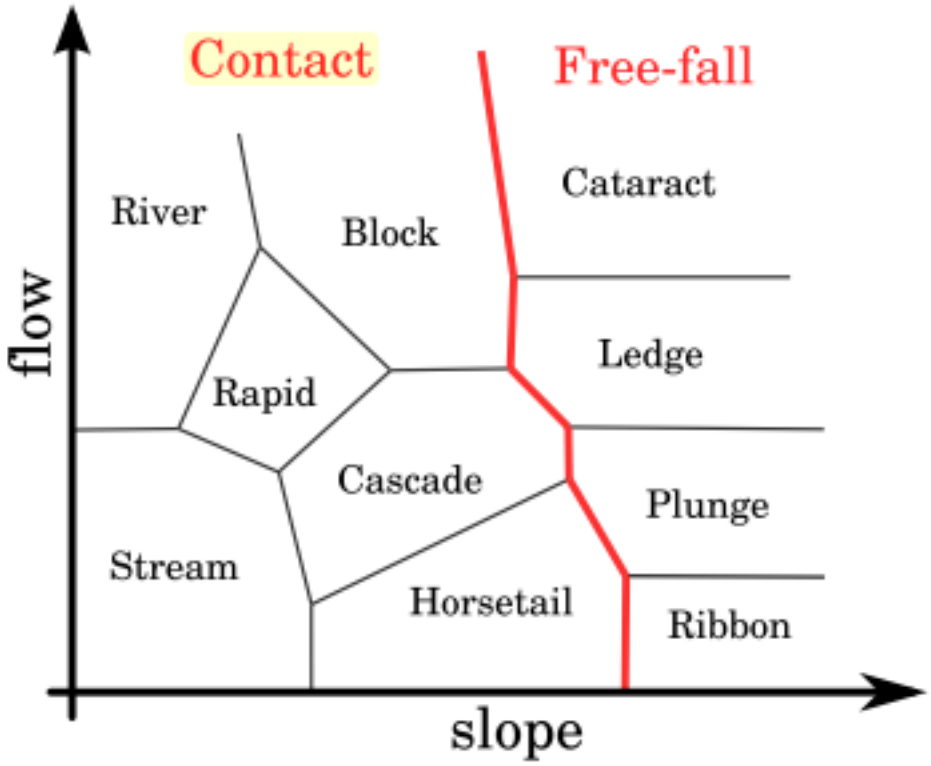

Figure 2.1: Waterfall classifications [**?**]

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 [**?**] is able to produce complex waterfall scenes in near real-time.

2.1.2 Simulation-based

Simulation-based techniques attempt to reproduce real-world phenomena in order to generate realistic river networks on terrains. The phenomena they attempt to reproduce differ and can be split into the following categories which will be discussed further: *Gravity*, *Erosion* and *Rainfall*.

2.1.2.1 Gravity-simulation

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 [**?**] 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 [**?**], 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.

2.1.2.2 Erosion-simulation

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.

[**?**] and t'Ava et al. [ŠBBK08] attempt to produce plausible terrains by modelling these effects.

In the work by Kelley et al. [**?**], 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 (**??**) 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).

Whether modelling erosion indirectly like in the work by Kelley et al. [**?**] 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. [**?**] 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.

2.1.2.3 Rainfall simulations

In order to determine where on the terrain rivers will appear, work by Soon Tee [**?**] 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 simulates 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

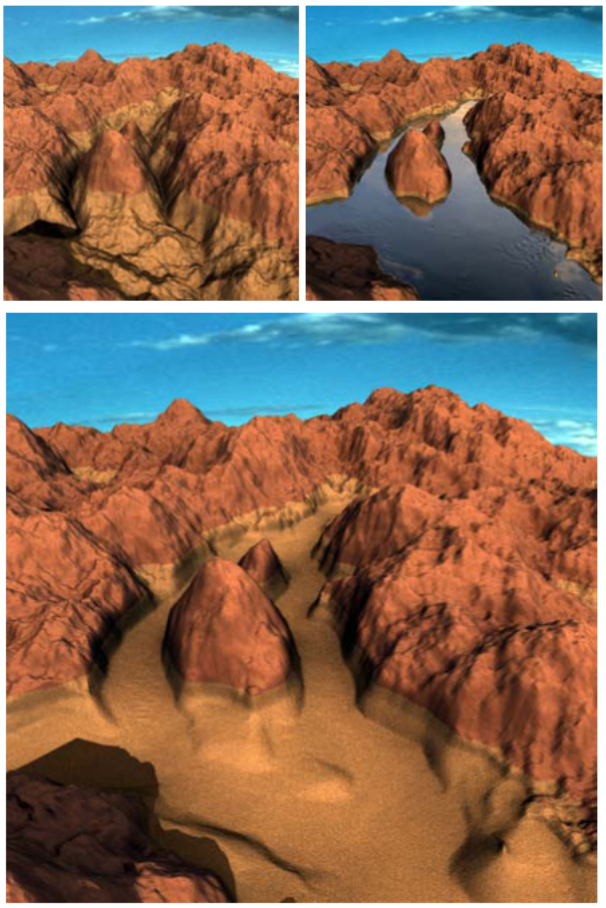

Figure 2.2: Simulation of dissolution-based erosion erosion caused by water movement[ŠBBK08]

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.

2.1.3 Heuristic-based

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

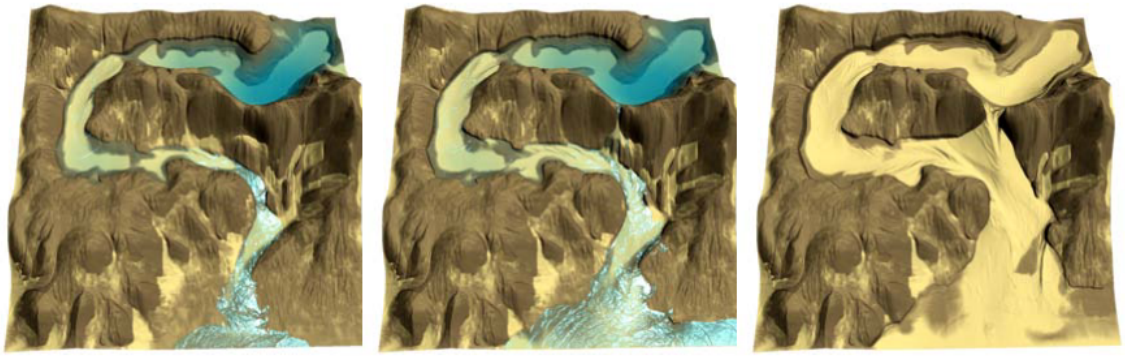

Figure 2.3: Simulation of the effect of force-based erosion caused by running water [ŠBBK08]

Derzapf et al. [**?**] 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 [**?**] 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.).

2.1.4 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.

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 **??**). An *entry edge* defines the point of entry for the river into the triangle, an *exit edge* the

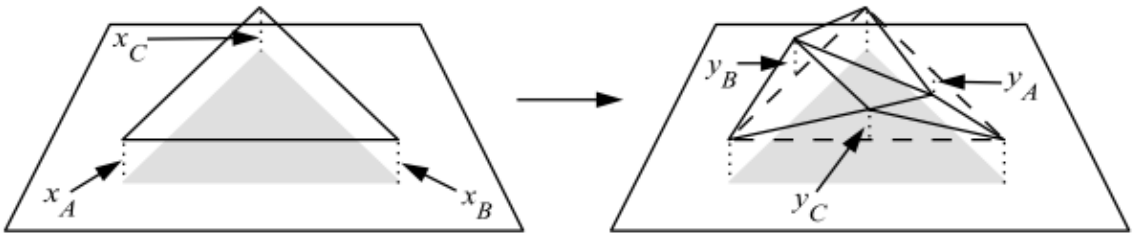

Figure 2.4: A single iteration of midpoint displacement for the creation of mountains [**?**]. New vertices y_A, y_B and y_C are created and shifted vertically by a random offset

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".

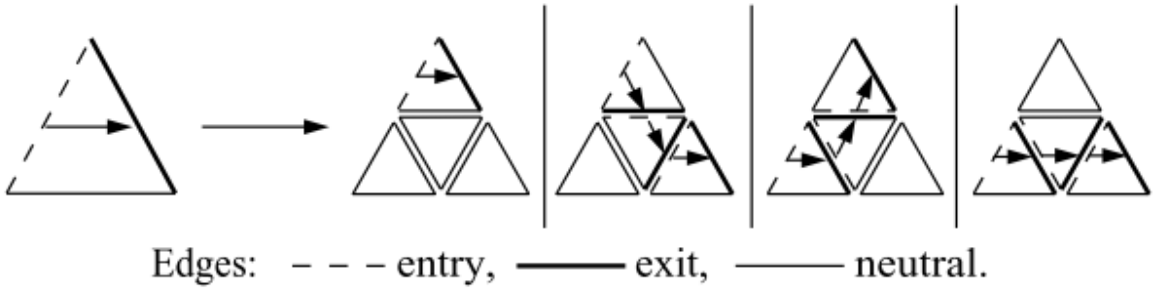

Figure 2.5: Single production of midpoint displacement adapted to river generation [**?**]. Given the initial triangle, four valid split scenarios.

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. [**?**] 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. [**?**], for example, this is limited to specifying the preferred river branching behaviours. In terms of performance, Gnevaux et al. [**?**] are able to produce terrains of several hundred square kilometres in a matter of seconds.

2.1.5 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 [**?**] 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. [**?**], 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 [**?**] 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.

2.1.6 Conclusion

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.

Classification-based, *Simulation-based*, *Heuristic-based*, *Fractal-based* and *Explicit*

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. This water drainage simulation should 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).

	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

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 emphasize 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 vegetative 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. [**?**] 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. [**?**] use orthophotos as input to determine the location and properties of individual plants. Unlike ordinary aerial photographs, aerial orthophotos use normalisation techniques to take into

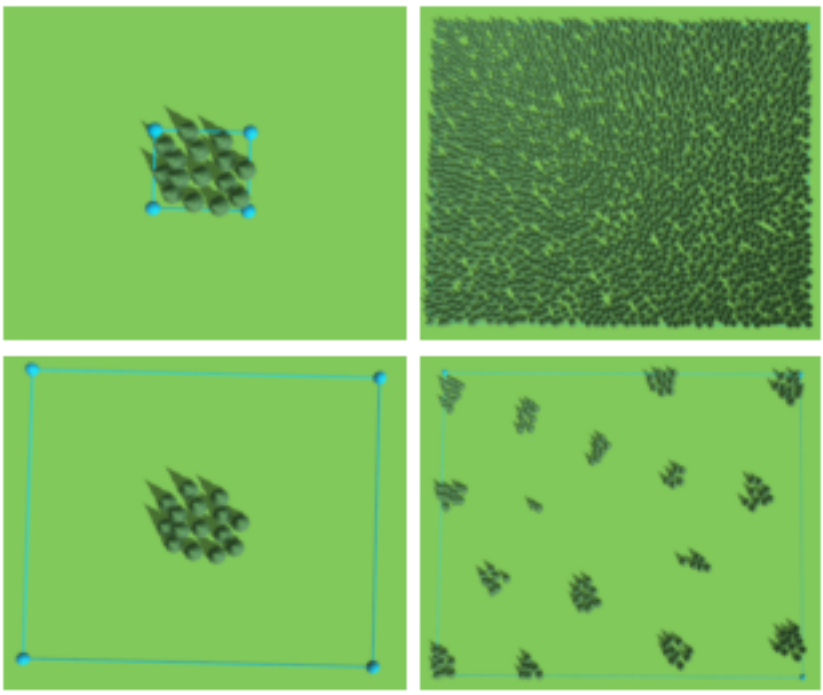

Figure 2.6: Using explicit placement as input examplars for reproduction [**?**]

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.

2.2.1.1 CONCLUSIONS

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.

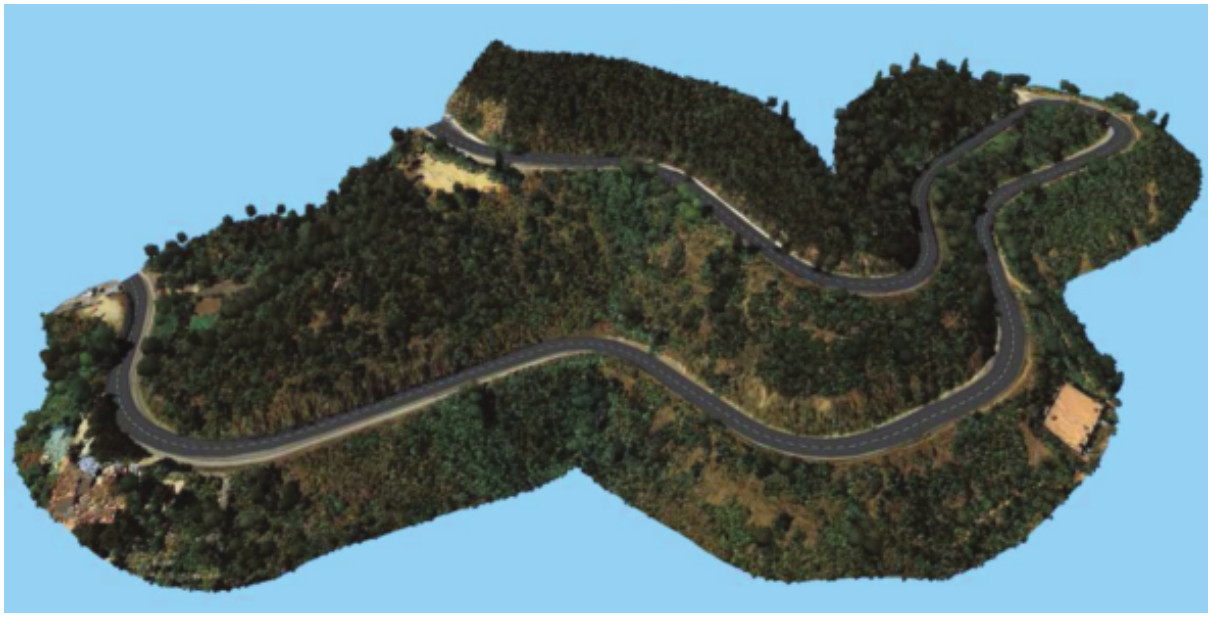

Figure 2.7: Reconstructed roadside vegetation using orthophotos [**?**]

2.2.2 Probabilistic Placement

Probabilistic placement methods use statistical models in an attempt to produce adequate vegetation. These methods can be further split into two sub-categories which are discussed in further detail below: *Radial Distribution Analysis* and *Predefined Ecosystems*. *Radial Distribution Analysis* approaches analyse the underlying distribution of the vegetation for later reproduction. *Predefined Ecosystems* calculate, based on the varying resources of the terrain and a set of predefined ecosystems, the best suited.

2.2.2.1 RADIAL DISTRIBUTION ANALYSIS

Work by Emilien et al. [**?**], Boudon et al. [**?**] 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, 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.

A core part of radial distribution analysis is generating pair correlation histograms for each 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)

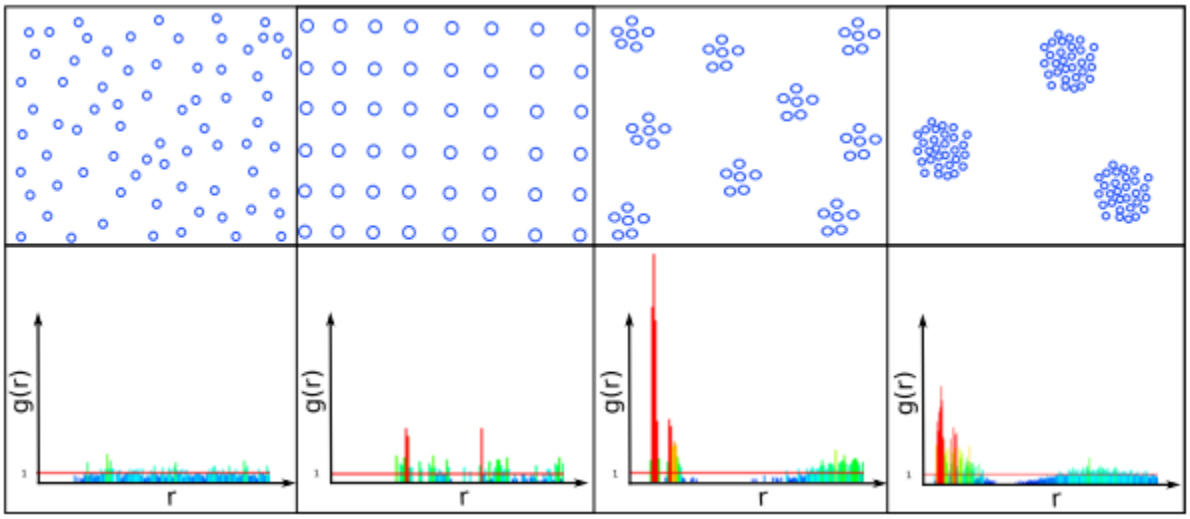

Figure 2.8: Point distributions with associated pair correlation histogram [**?**]

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).

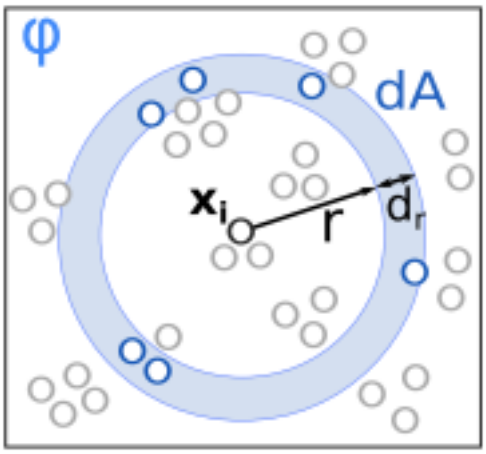

Figure 2.9: Radial distribution analysis

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:

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

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.
- 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 calculates the variance from the average density of the target category at incremental distances from points of the reference category.

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 [**?**] 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.

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 [**?**] are able to perform analysis and reproduction in near real-time.

When using this technique to reproduce a plausible plant distribution, Boudon et al. [**?**] 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.

2.2.2.2 PREDEFINED ECOSYSTEMS

In their work, Hammes el al. [**?**] 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.

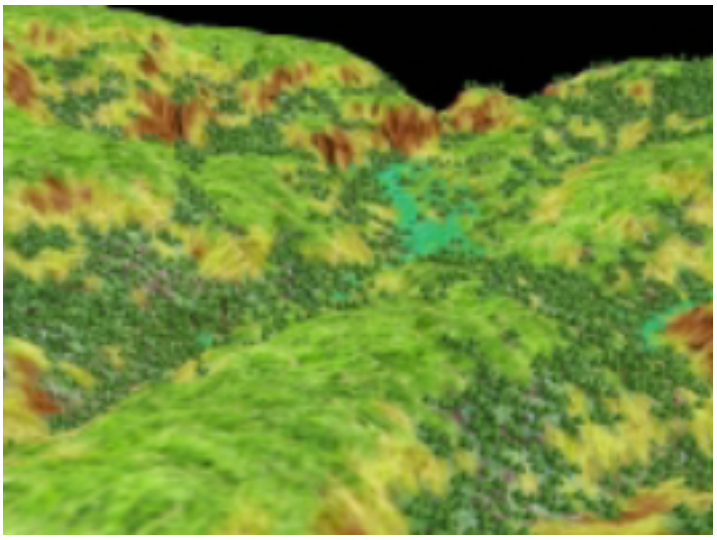

Figure 2.10: Vegetation generated using predefined ecosystems [**?**]

2.2.2.3 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.

2.2.3 Simulators

Another approach used to determine vegetation in a given environment is to simulate plants battling for available resources. This approach can be further classified into two subcategories: *Plant Growth Modelling* and *Ecosystem Simulators*.
Plant Growth Modelling techniques go into extensive detail to model the effect of resources on plant growth. The realism is such that it can often be used to model plant growth on earth.
Ecosystem Simulators try to simulate plants competing for available resources when growing. Unlike plant growth modelling which targets botanical realism, these techniques target visual realism. As a consequence, they are more lenient in terms of realism.

2.2.3.1 PLANT GROWTH MODELLING

These types of simulators 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. [**?** **?**] 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. [**?**], 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. [**?**] 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 specie 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.

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).

2.2.3.2 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.

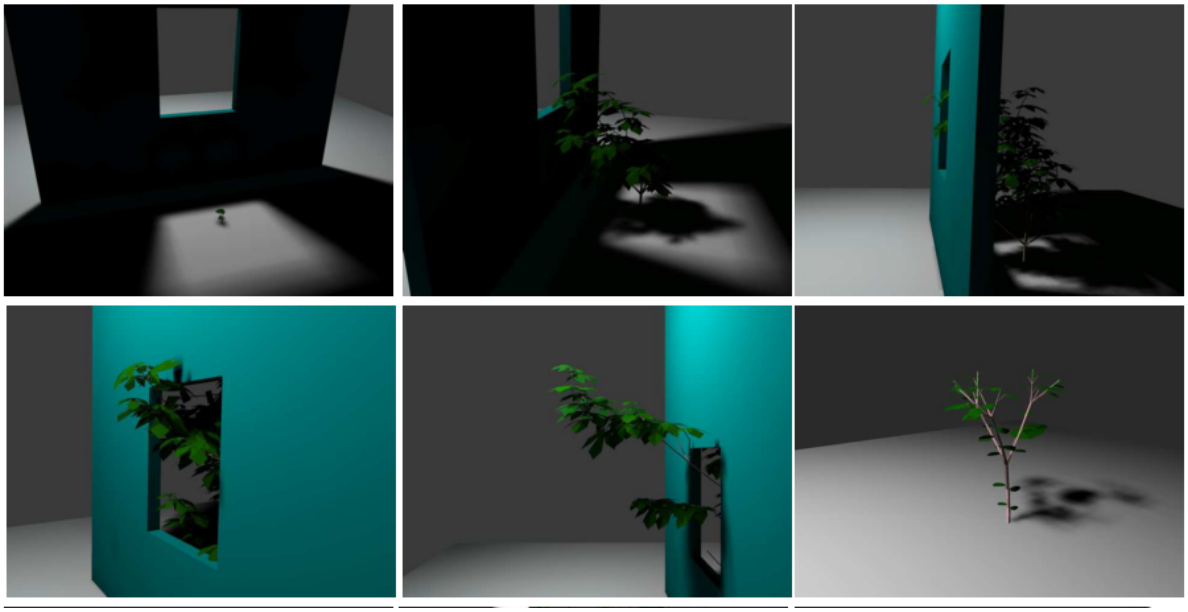

Figure 2.11: Plant growing towards light source [**?**]

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 [**?** **?** **?** **?**].
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 [**?**]. In their work modelling the growth of struce trees, Berezovskava et al. [**?**] 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 specie A with a fast growth rate and specie B with a slower growth rate but higher shade tolerance. At first, the faster growing specie A will dominate and flourish but, with time, the slower growing but more shade tolerant specie 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.

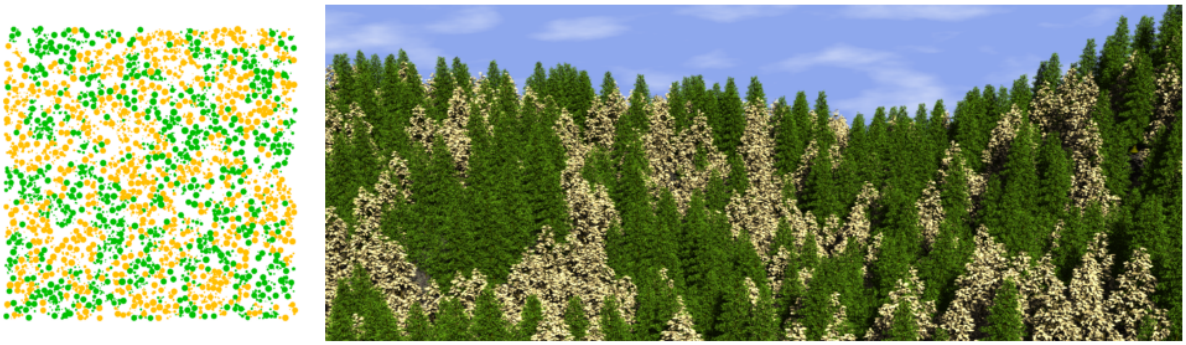

Figure 2.12: Plant placement using an ecosystem simulator modelled by L-Systems [LP02]. *Left:* Result of the simulation where orange circles indicate the positions of poplar trees and green circles the positions of spruce trees. *Right:* Reproduced virtual world where the location of individual plants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

A direct consequence of the automation provided by these ecosystems is that fine control over the final vegetative 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).

2.2.3.3 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. [**?**], simulating 45 cycles for a single plant takes approximately 15 minutes.

2.2.4 Conclusion

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 2.2.2 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. 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

Multiple components serving specific purposes constitute the building blocks of the overall system, notably: *resource gathering*, *resource clustering*, *plant selection*, *ecosystem simulation* and *distribution analysis and reproduction*. The purpose of this chapter is to give an overview of the system, each component and how they fit together (as illustrated in figure 3.1). To do so, an overview of each component is provided separately along with it's purpose, required inputs and outputs.
To conclude, the *limitations* of the system will be discussed and *comparisons and differences* drawn with previous work in the field.

3.1 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.
In order to calculate the *soil humidity* for each month, the soil infiltration rate, monthly rainfall 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

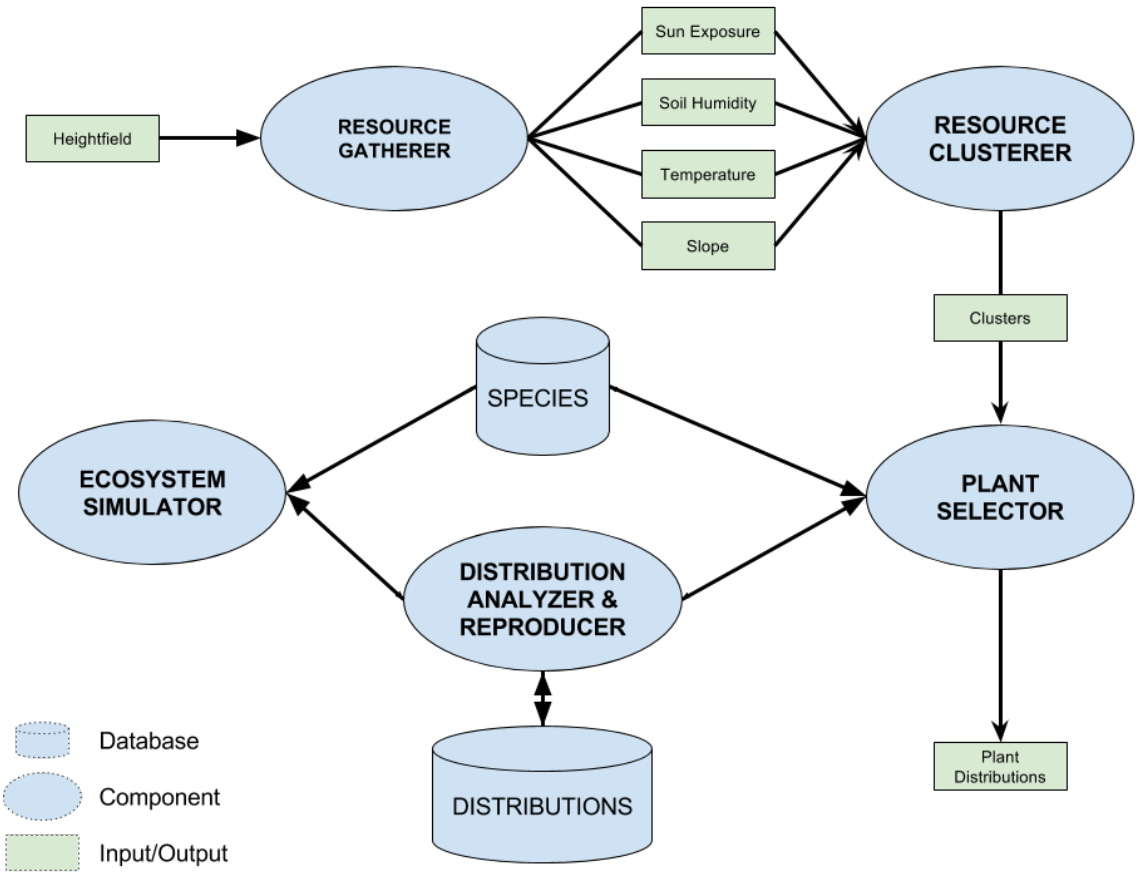

Figure 3.1: System overview

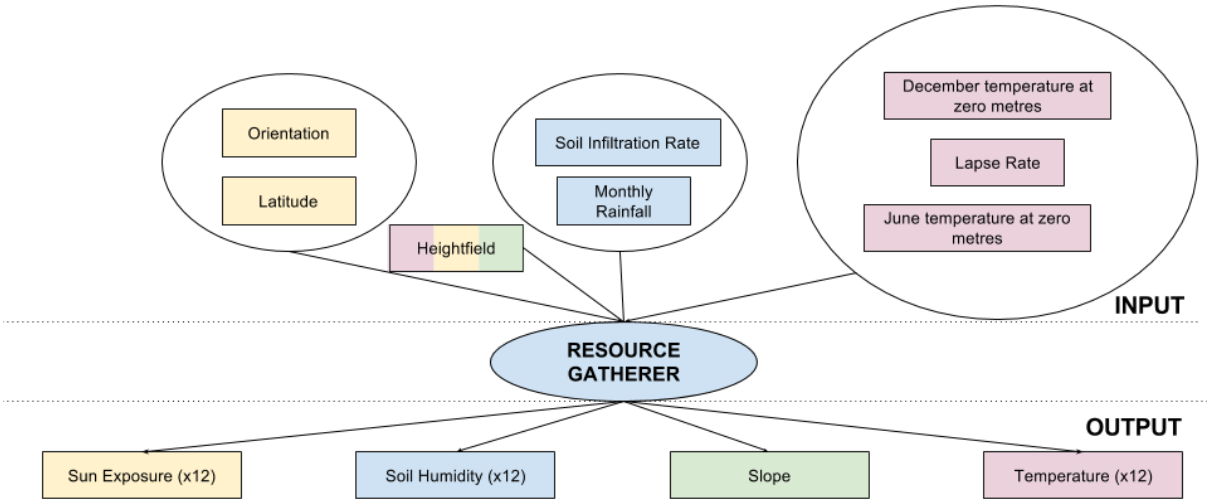

Figure 3.2: Resource gatherer overview with colour coding to correlate input with corresponding output.

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 Resource Clusterer

Determining a suitable plant distribution for each individual terrain vertex is infeasible due to the associated computation cost. 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 3.1. Given all this information, the clustering algorithm gathers points which are most similar in terms of resources into a set of k clusters.

3.3 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.

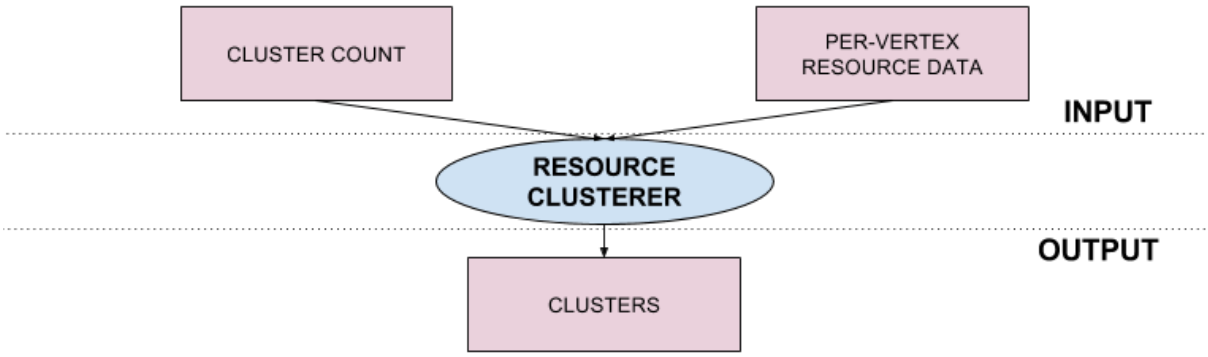

Figure 3.3: Resource clusterer overview with required input and generated output.

This score depicts how suited a species is to each individual cluster and is displayed to the user for informational purposes in order to facilitate the specie selection procedure. Figure 3.4 shows the input requirements and outputs of this component.

3.4 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.

3.5 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 [**? ?** 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.

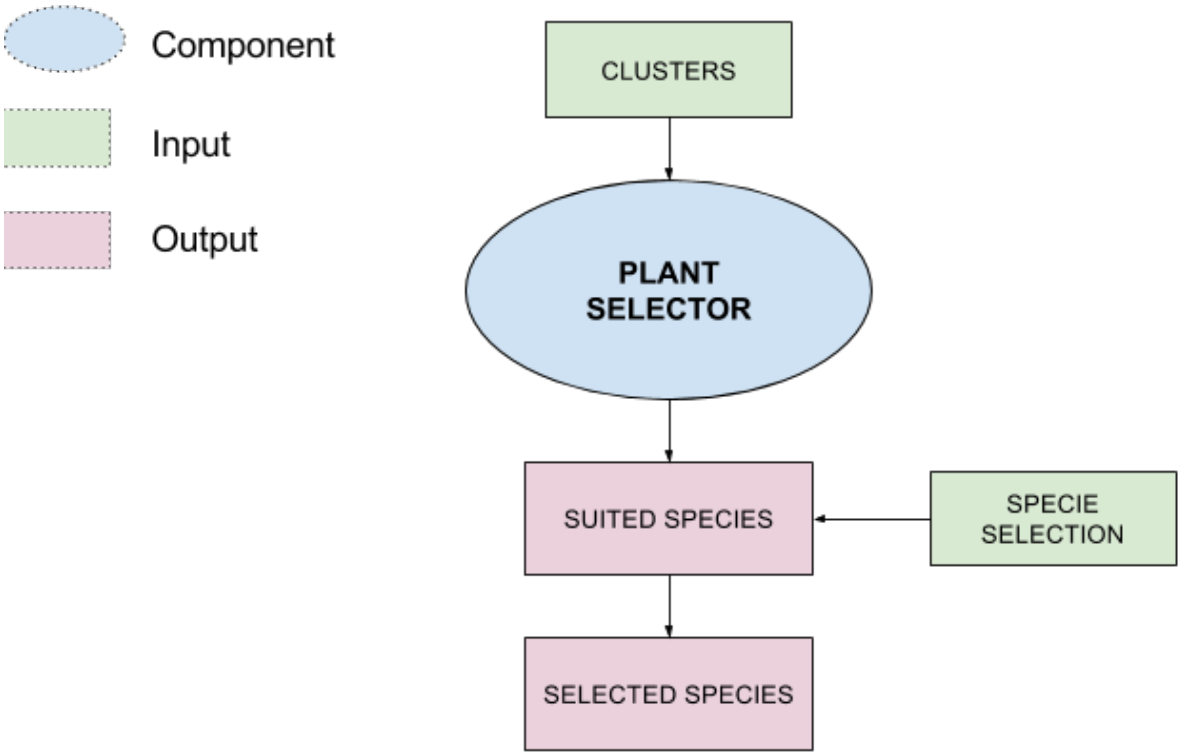

Figure 3.4: Plant selector overview.

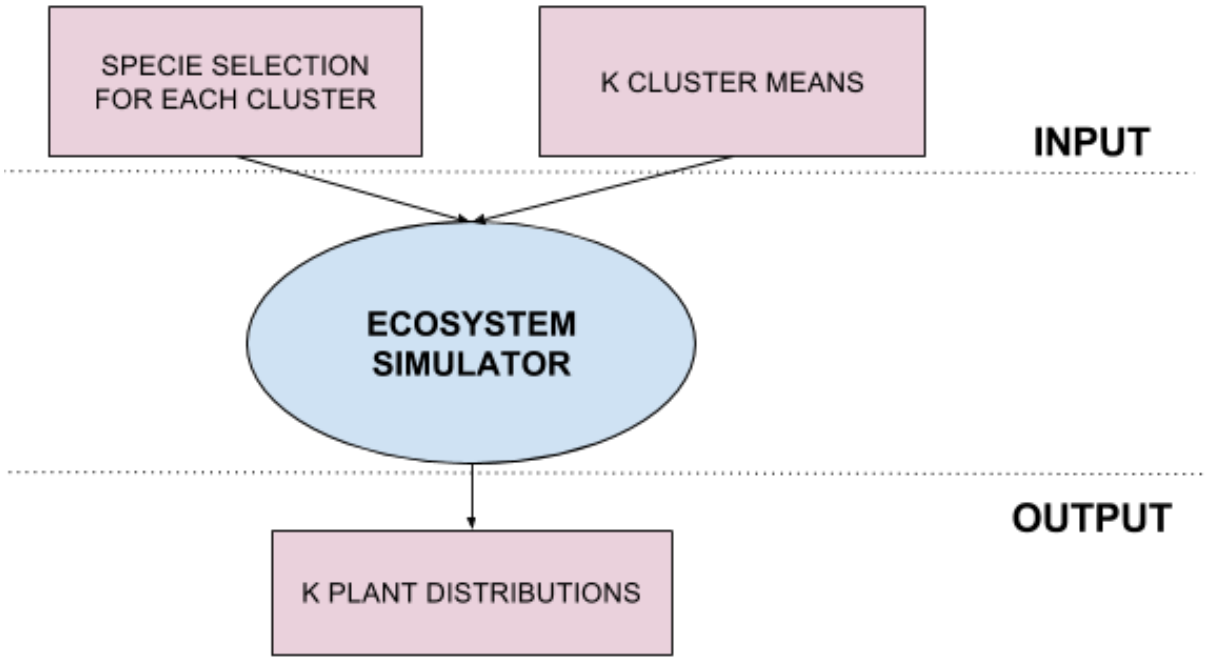

Figure 3.5: Ecosystem simulator overview.

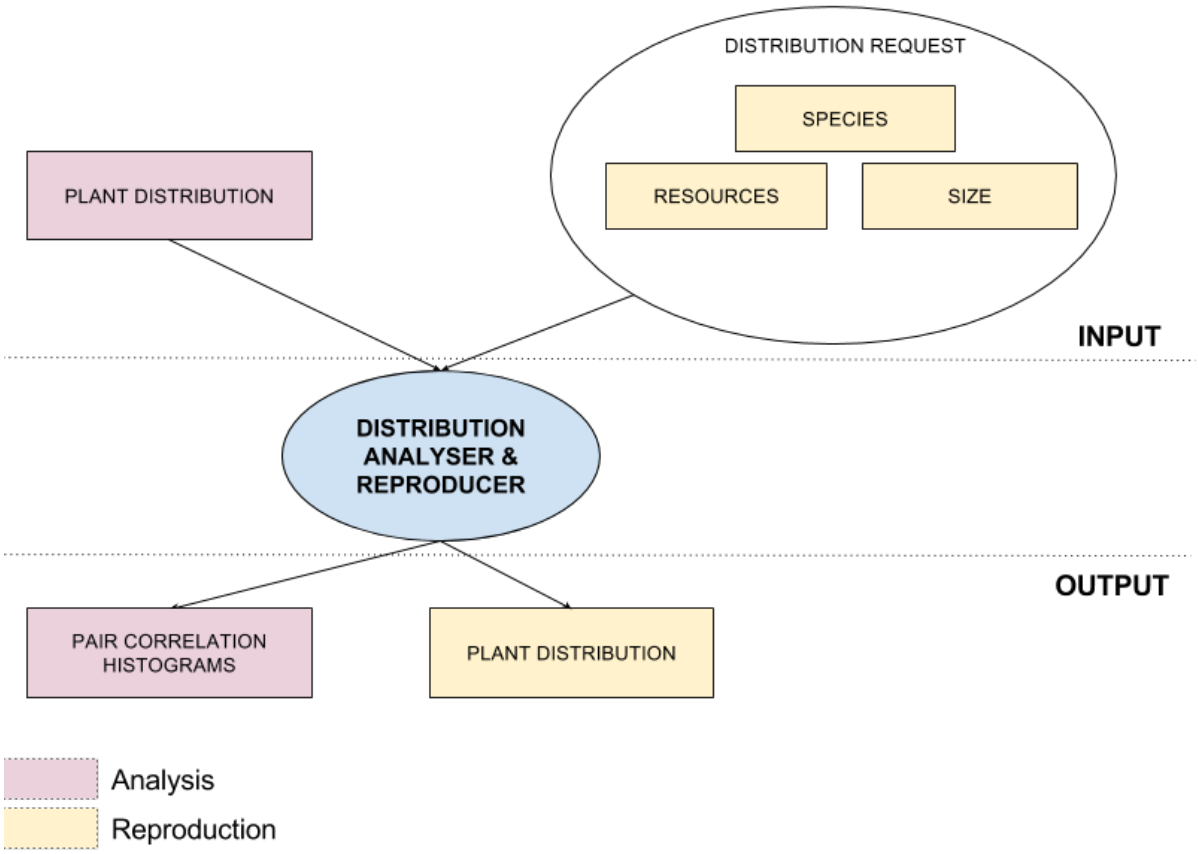

Figure 3.6: Distribution analyser and reproducer overview.

3.6 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.

3.7 Similarities to Existing Work

Explicit instancing techniques [**?** DHL+98**?**] 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. [**?**] 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`

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.

Chapter 4

Terrain & 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 is split 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 [Bli]. 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`

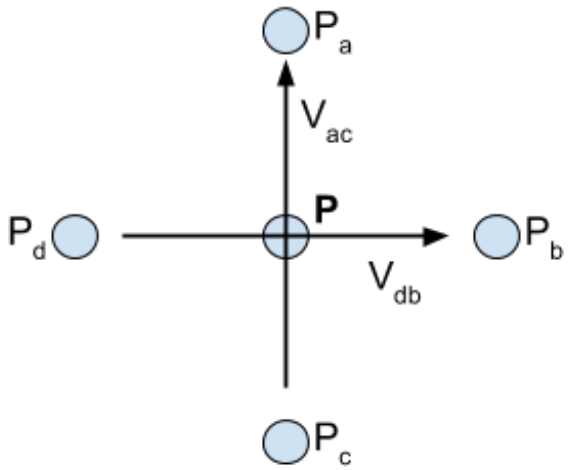

Figure 4.1: Illustration of the vertices and vectors used to calculate terrain normal at position *P*.

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**
First-Person	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

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 vegetative 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

Sunlight and it's annual variation greatly effects the type and density of vegetation. Whereas some plants prefer habitats with limited sun exposure (e.g lilies), some flourish in fully exposed environments (e.g sunflowers).
To determine whether or not a point on the virtual terrain is illuminated at any given time of the year, the system must be able to track the suns trajectory through time.

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). For the northern hemisphere, the day length will be at its maximum during the June equinox and at its minimum during the December equinox. On these days, the earth will be tilted at it's maximum towards and away from the sun respectively.

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.

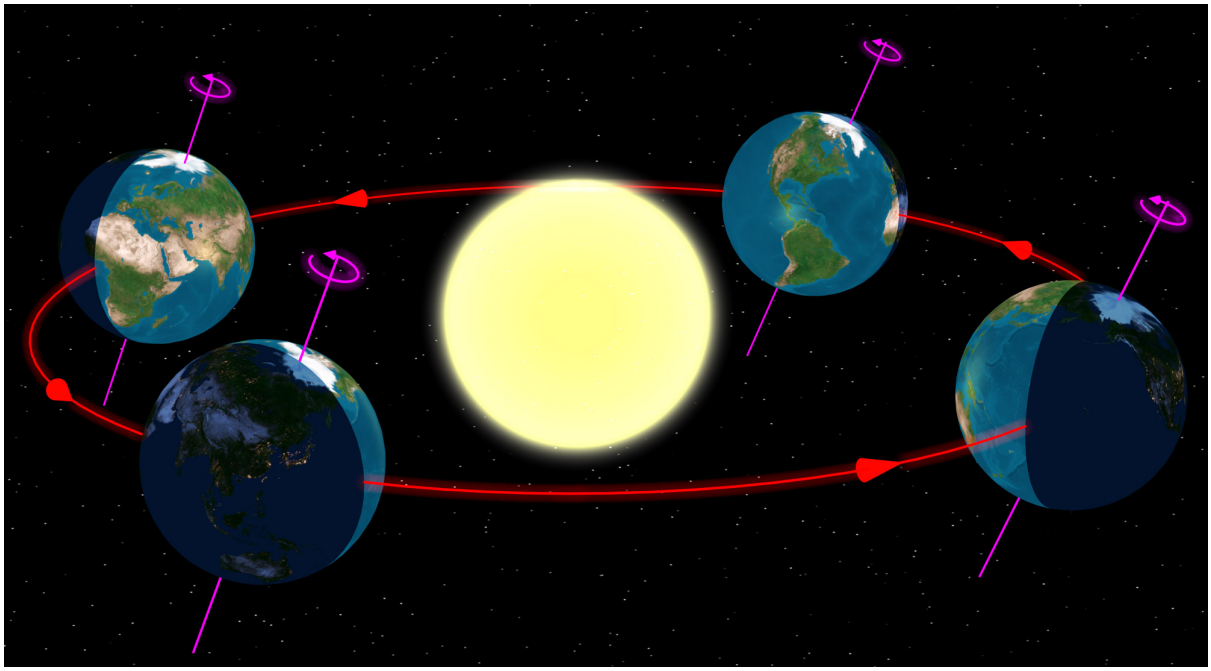

Figure 4.2: Annual orbit of the earth around the sun. Source: `http://en.wikipedia.org/wiki/Summer_solstice`

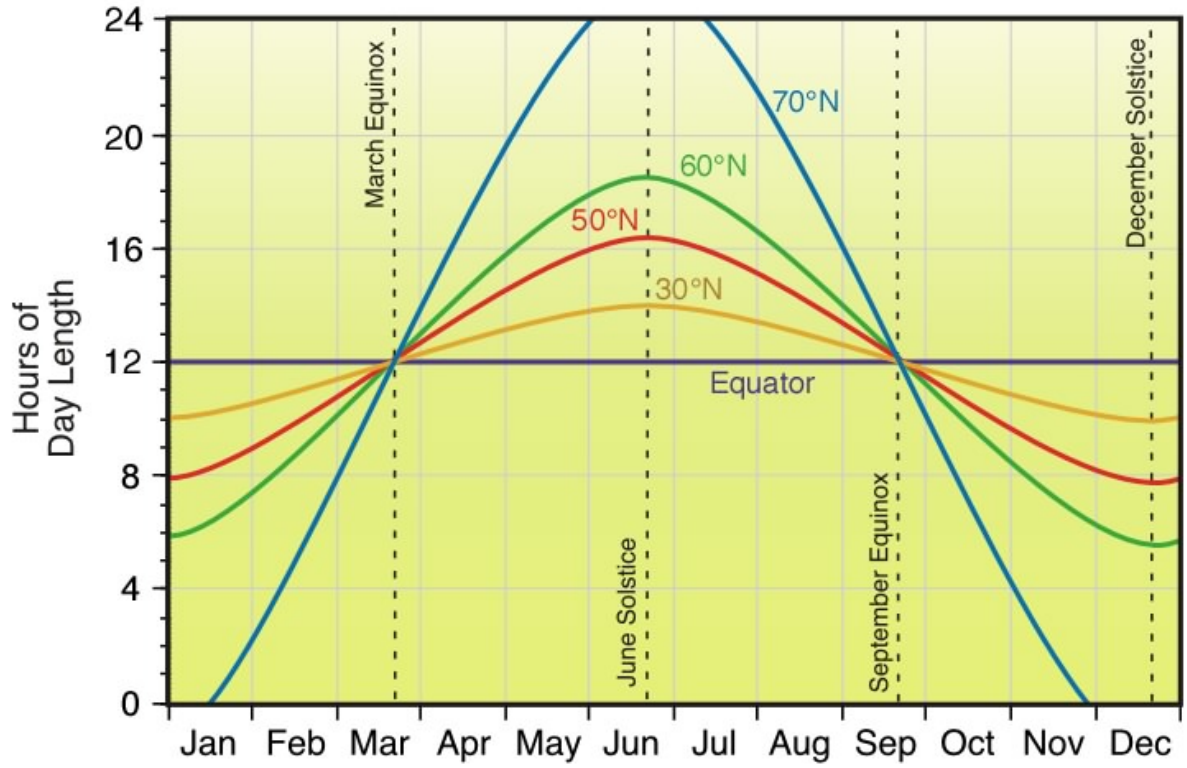

Figure 4.3: Variation in day length for different latitudes. Source: `http://www.physicalgeography.net/fundamentals/6i.html`

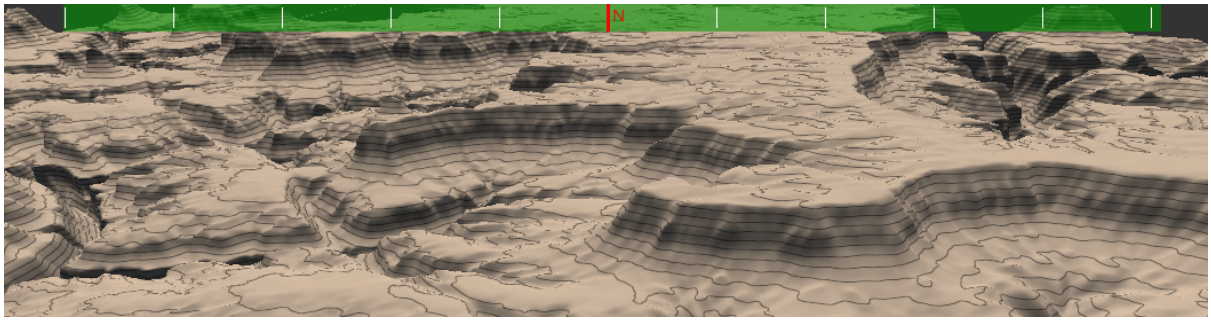

Figure 4.4: Orientation controller compass (top of render window). The compass is displayed green to provide feedback to the user that orientation edit mode is active.

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 $\mathrm{V}_R(\mathrm{m})$ 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.

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 figures **??**).

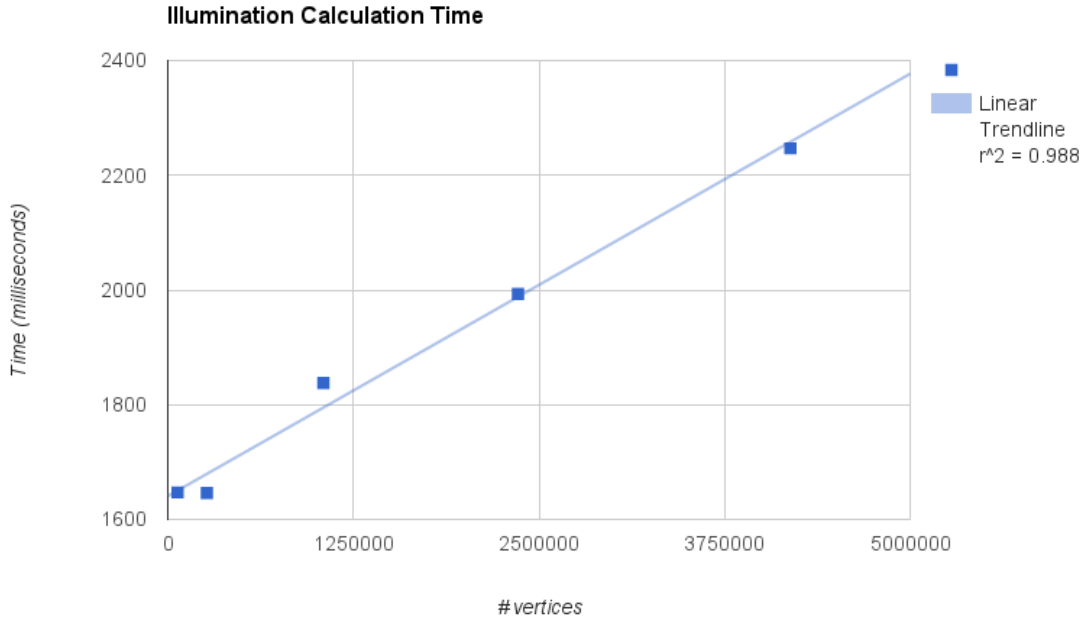

Figure 4.5: Illumination calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.

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. Interpolation 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 atmo-

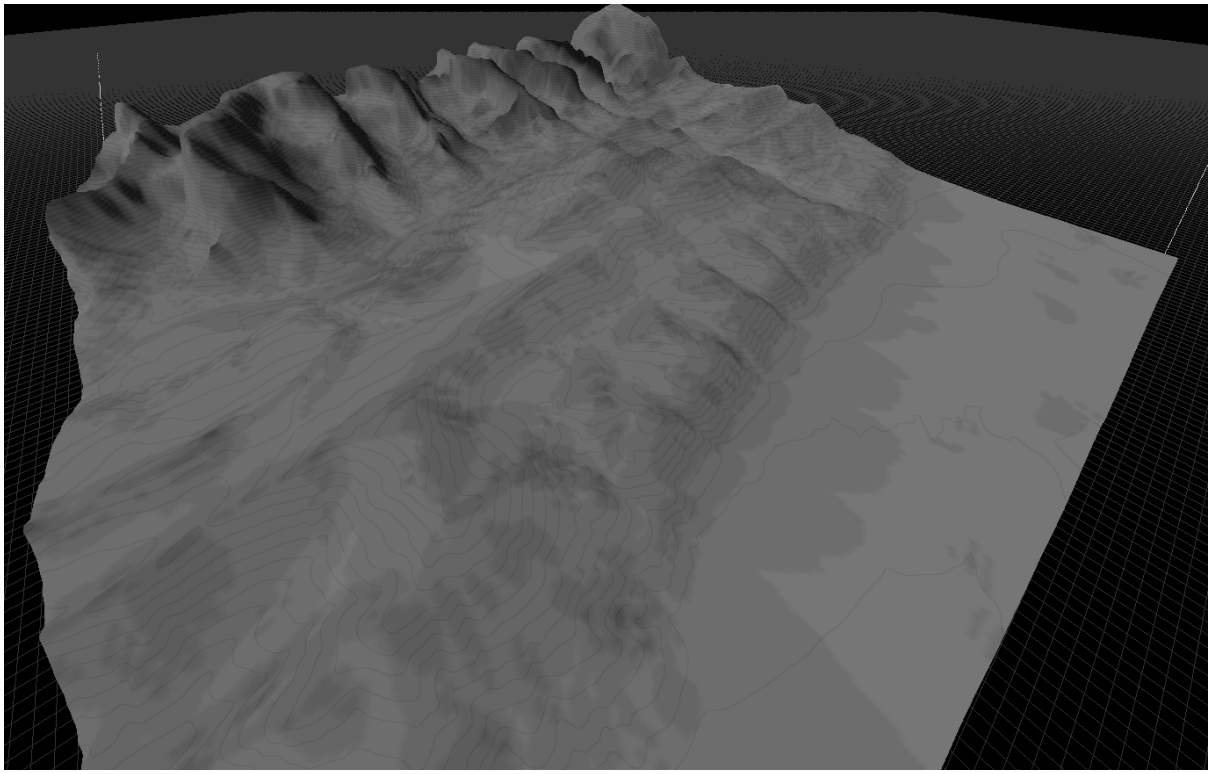

Figure 4.6: Daily illumination overlay. The brighter the area, the more illumination it receives throughout the day.

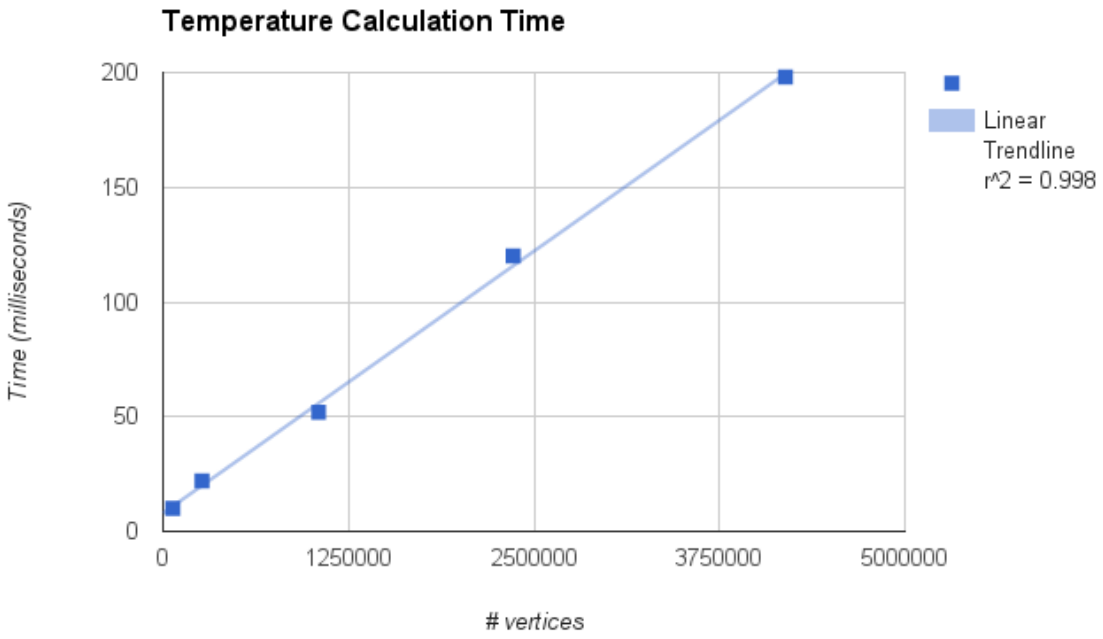

Figure 4.7: Temperature calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.

spheric 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).

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

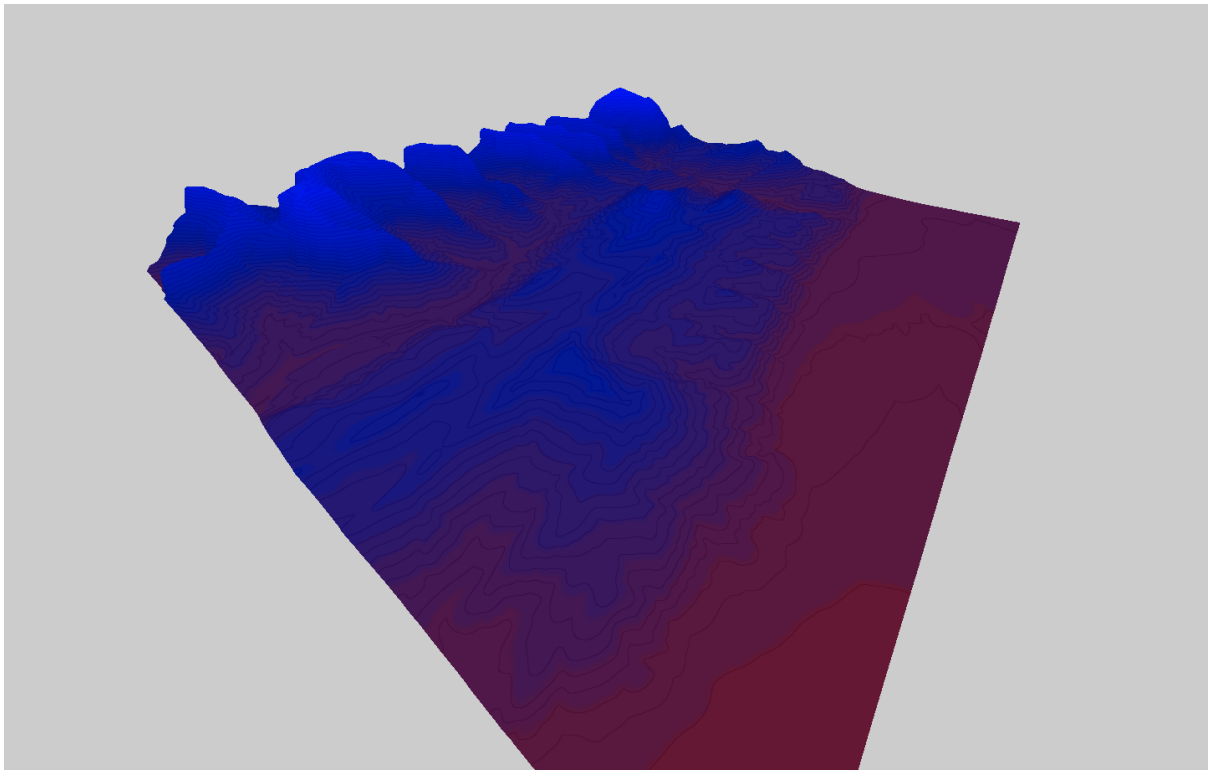

Figure 4.8: Temperature overlay. Colour spectrum ranging from blue (cold) to red (hot). As can be seen, the temperature drops with altitude.

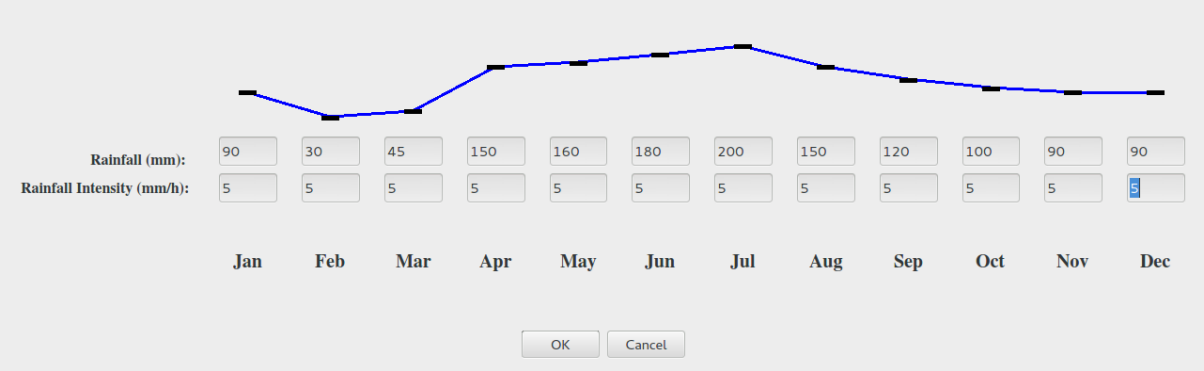

Figure 4.9: Specifying monthly precipitation and precipitation intensity. The user can enter values manually (using the input fields) or interact directly with the graph.

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).

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 [**?**]. 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 (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.).

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

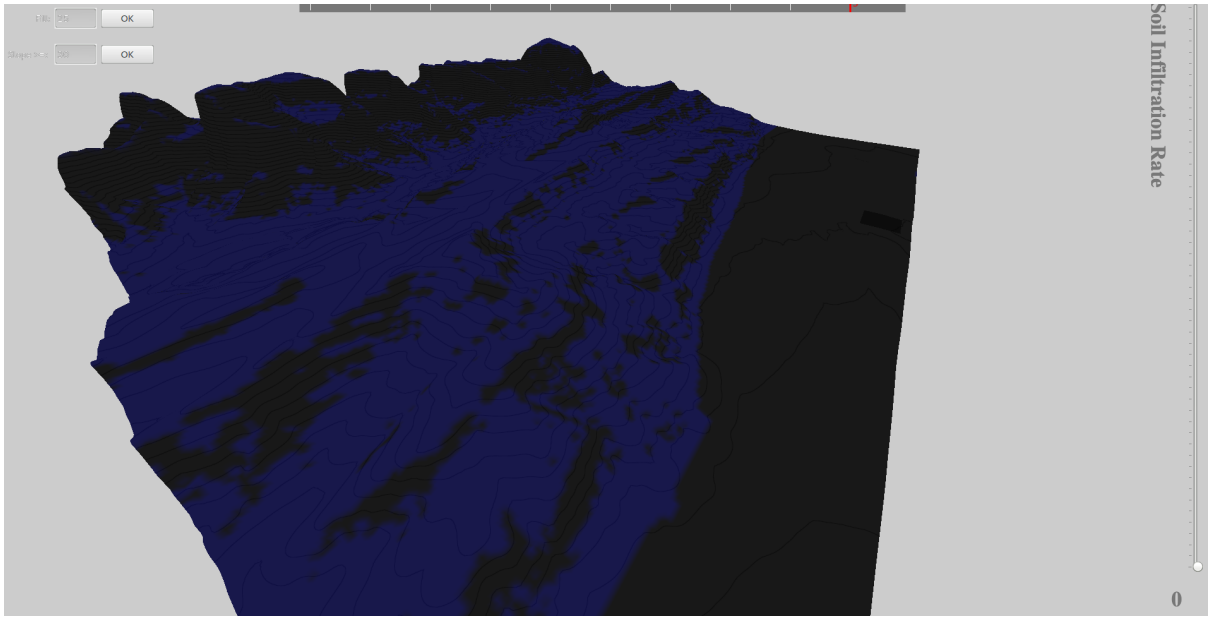

Figure 4.10: Editing the soil infiltration rate on the terrain. Top left are the controllers for the *filling* and *slope-based* tools. On the right is the slider to configure the soil infiltration rate of the *paint brush*.

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.

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

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).

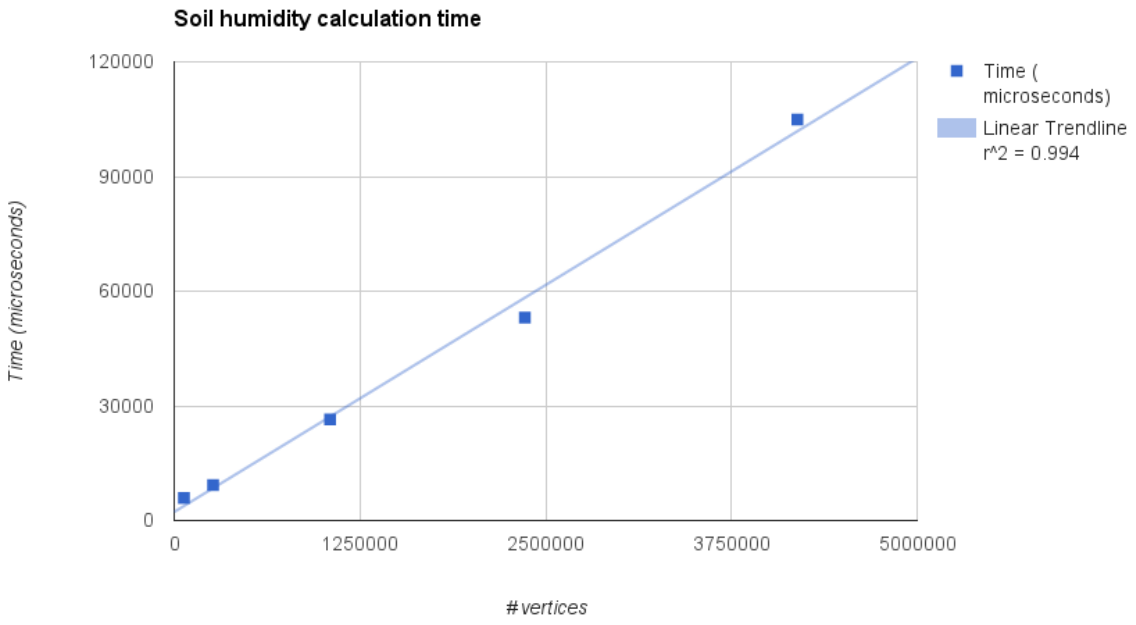

Figure 4.11: Soil humidity calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.

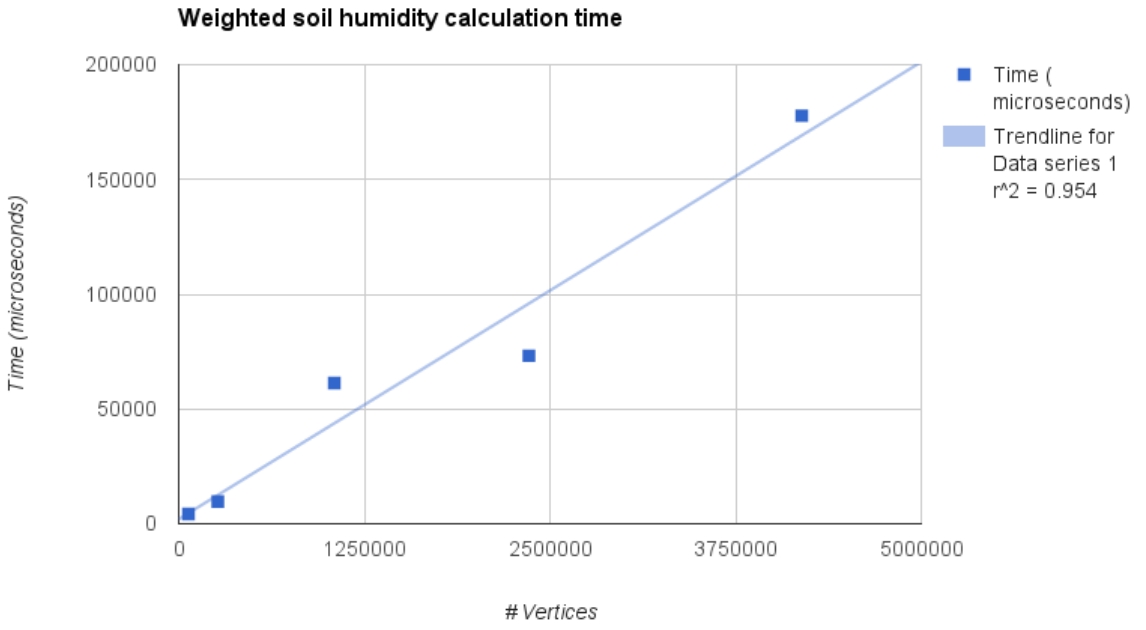

Figure 4.12: Weighted soil humidity calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.

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.

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

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).

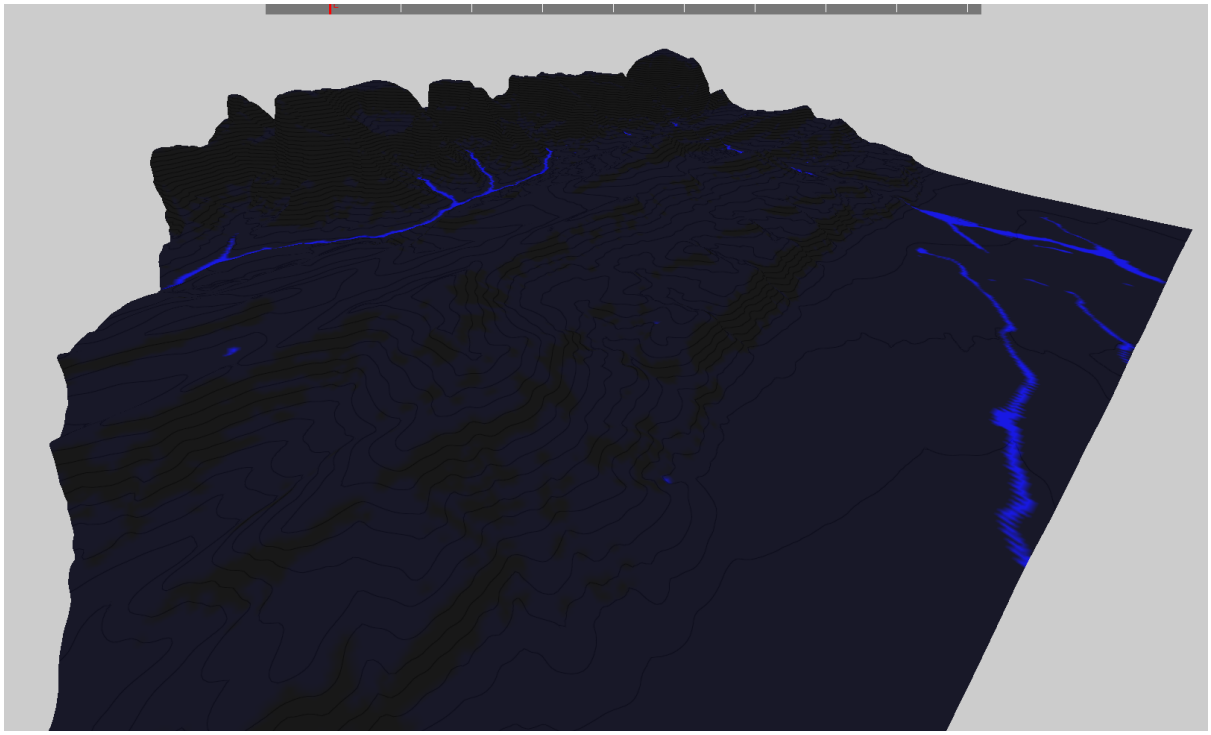

Figure 4.13: Soil humidity terrain overlay. Colour spectrum ranging from black (zero humidity) to blue (standing water).

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 [**?**]. This is clearly visible in nature, where only smaller plants (shrub, grass, etc.) grow on steeper landscapes. To determine suitable 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).

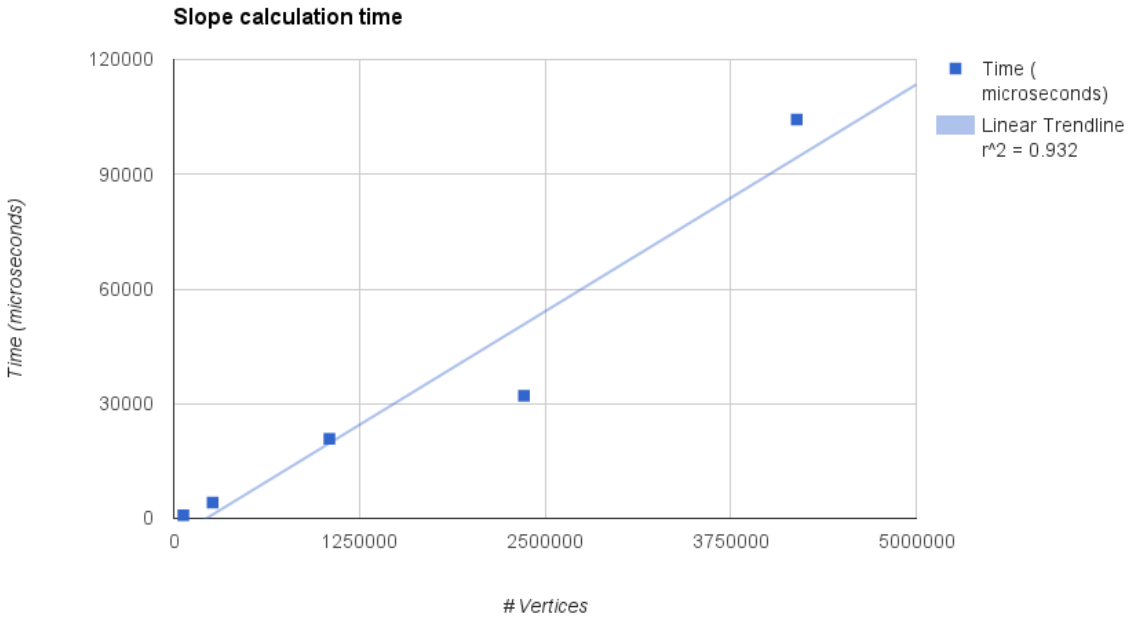

Figure 4.14: Slope calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024, 1536 by 1536 and 2048 by 2048.

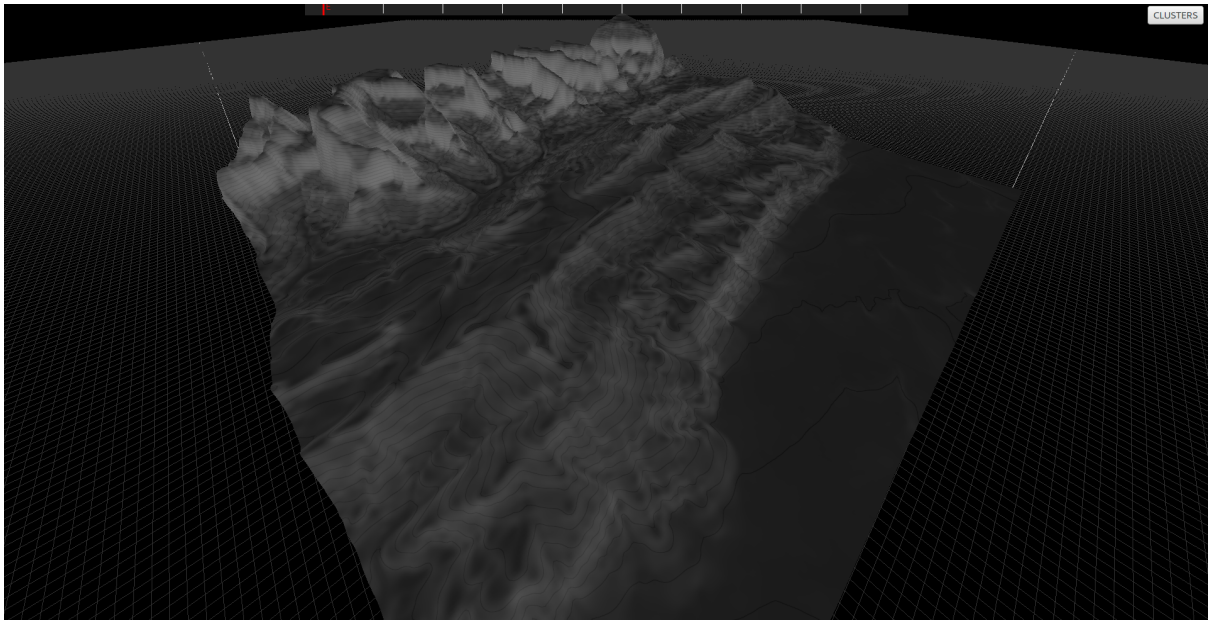

Figure 4.15: Slope visual overlay. Colour spectrum ranging from black (zero degree slope) to white (90 degree slope).

4.3 Rivers & 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 [**?**] 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 .

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

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

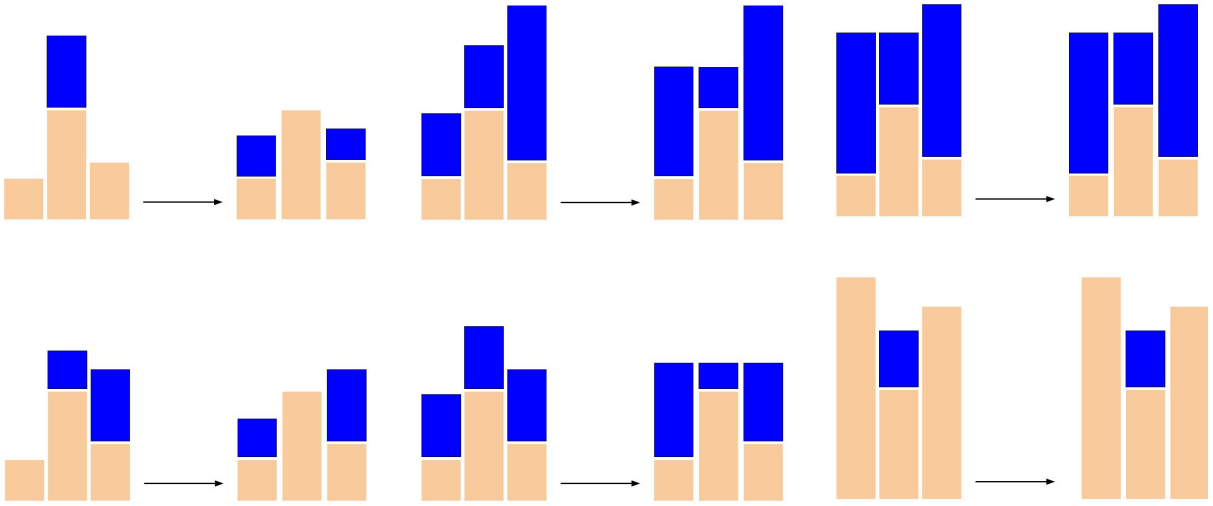

Figure 4.16: Example water-evacuation scenarios. Left: All water can be evacuated from source vertex (middle). Middle: A portion of water can be evacuated from source vertex (middle). Right: No water can be evacuated from source vertex (middle).

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.

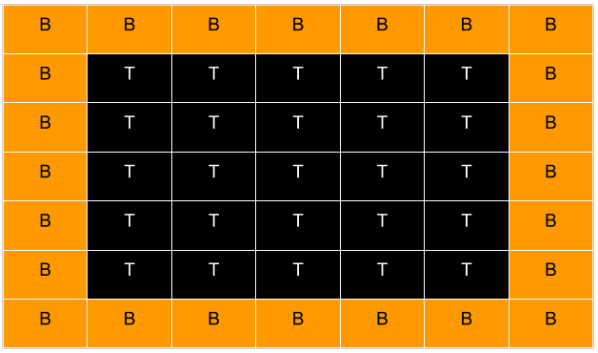

Figure 4.17: Border of vertices generated around the terrain to cater for water evacuation at the extremities. Orange vertices form the border.

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.

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

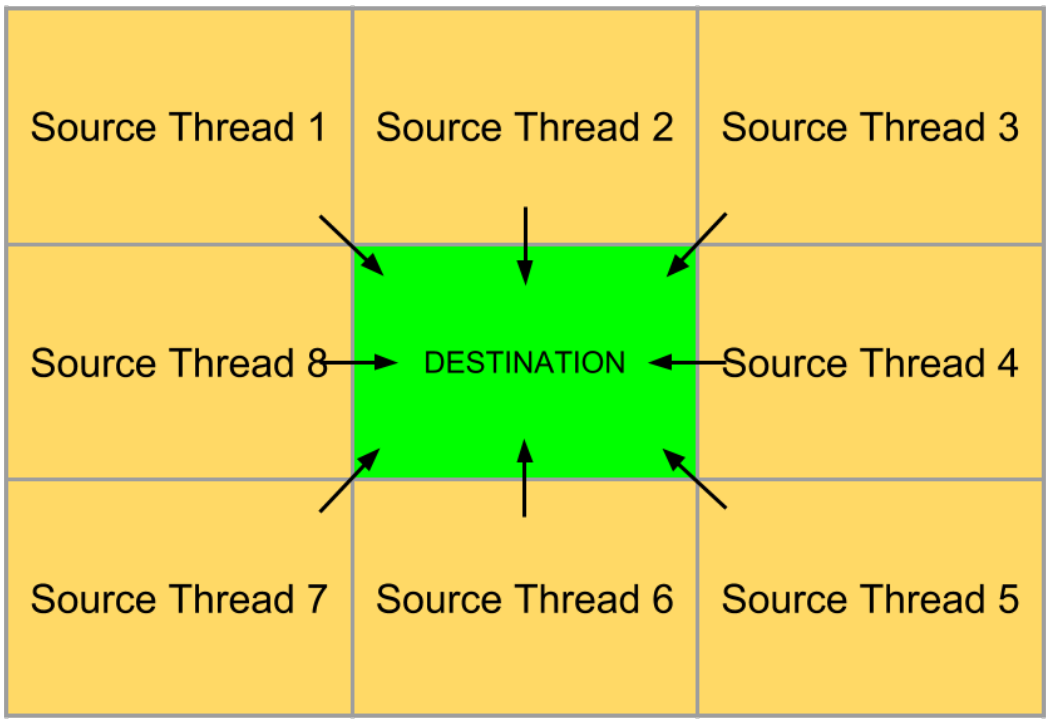

Figure 4.18: Memory conflict cause by 8 surrounding threads attempting to write to a single destination memory location.

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).

To prevent such memory conflicts, 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 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

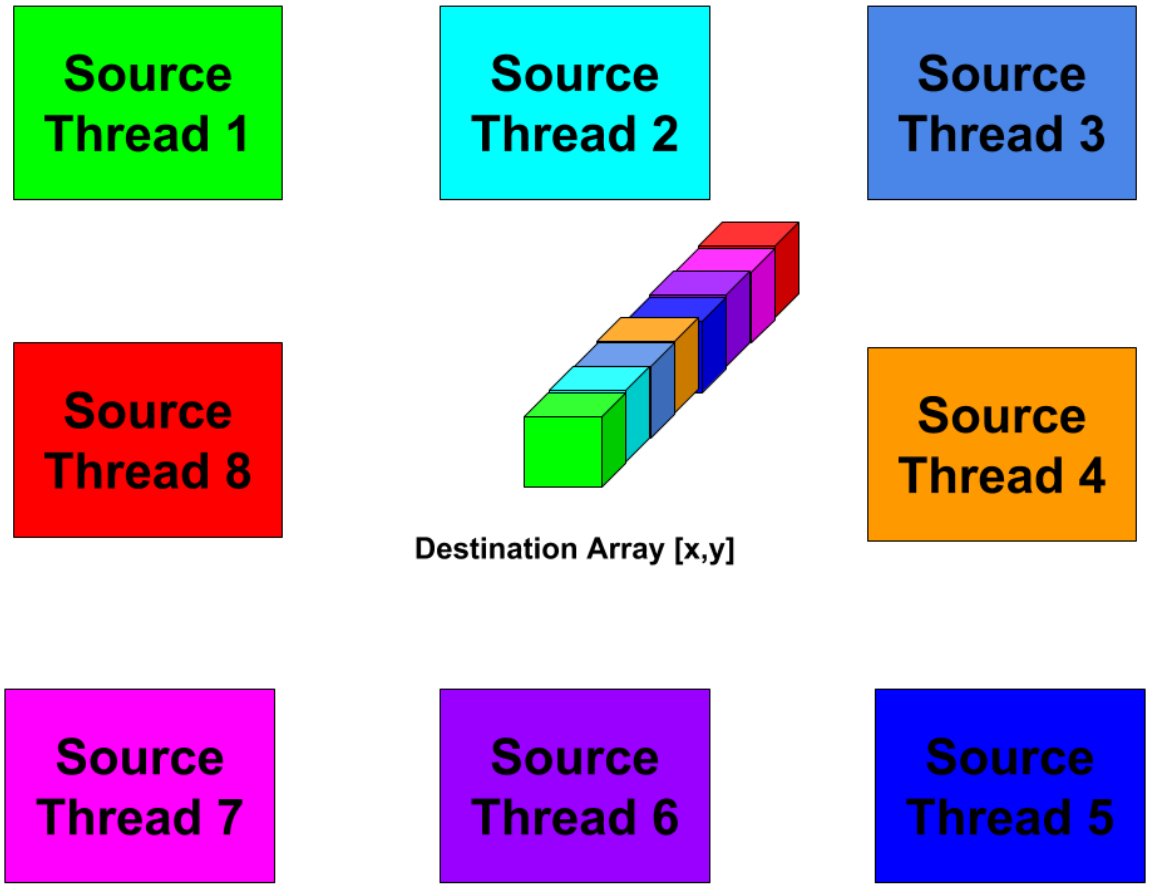

Figure 4.19: Memory conflict prevention by using a three dimensional array to aggregate the water to add.

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

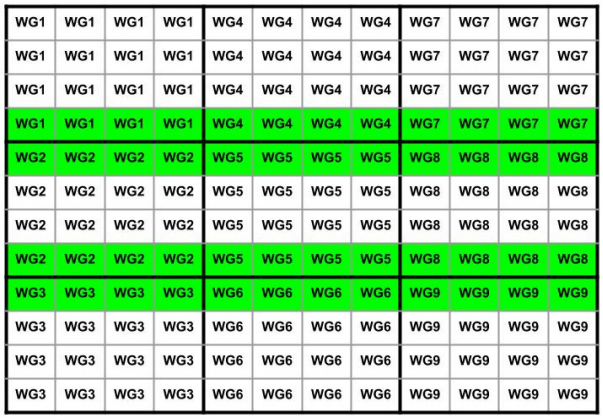

Figure 4.21: Global memory allocation requirement (green) to cater for inter work group memory access in the vertical direction. WG = work group

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.

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

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

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.

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

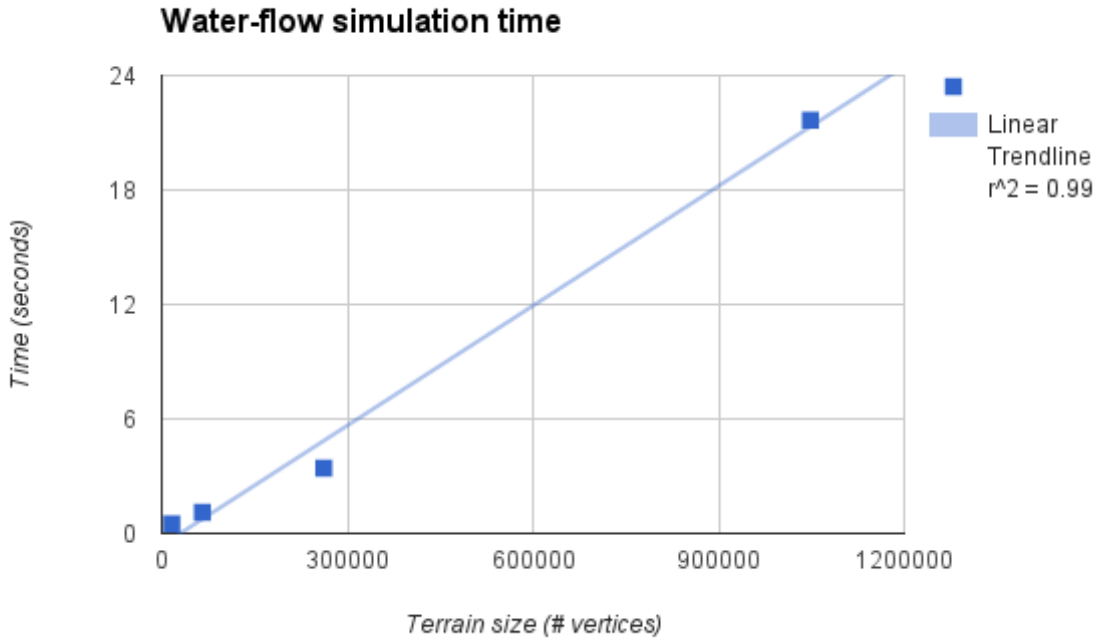

Figure 4.23: Total water-flow simulation time for 100 millimetres per terrain vertex. Analysis performed for terrains with dimensions: 128 by 128, 256 by 256, 512 by 512 and 1024 by 1024.

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 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 [**?**] 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.

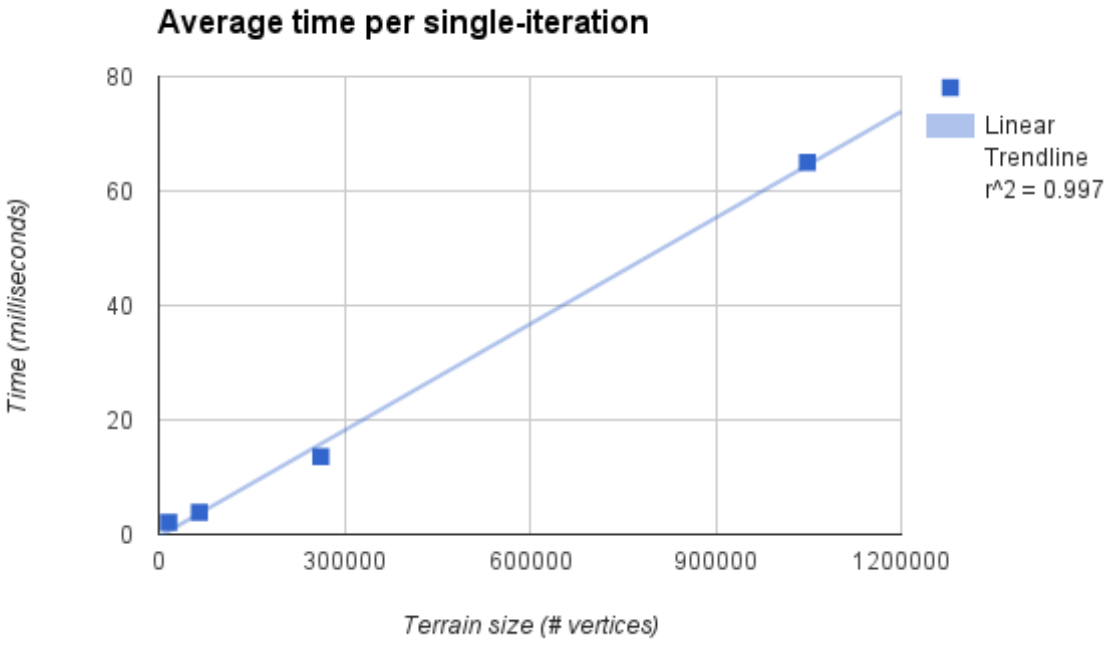

Figure 4.24: Average time for a single iteration of the water-flow for 100 millimetres per terrain vertex. Analysis performed for terrains with dimensions: 128 by 128, 256 by 256, 512 by 512 and 1024 by 1024.

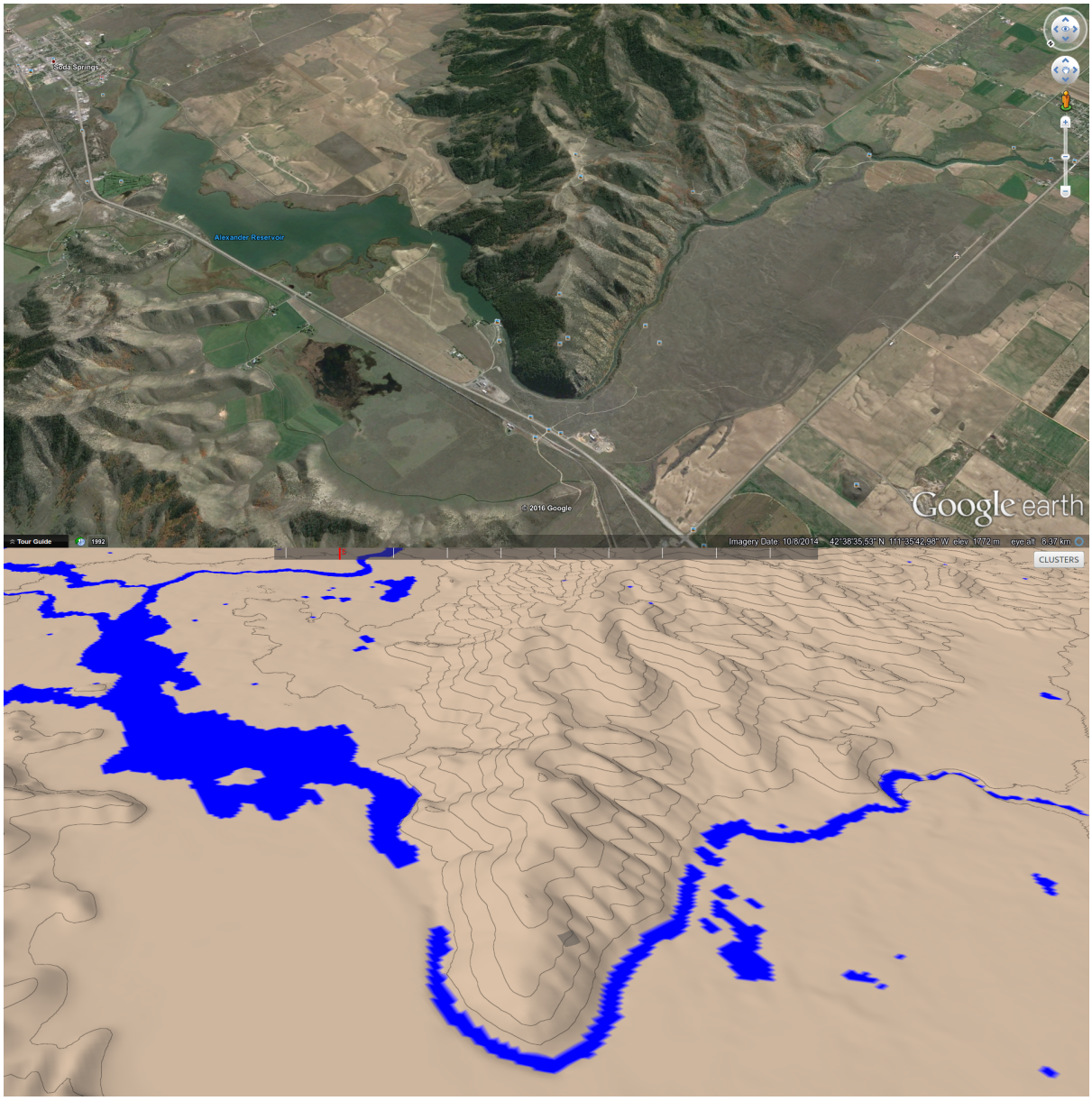

Figure 4.25: Top: Real world water-network at geographic coordinate location: 42°38'N 111°35'W. Bottom: Replica using the water-flow simulation.

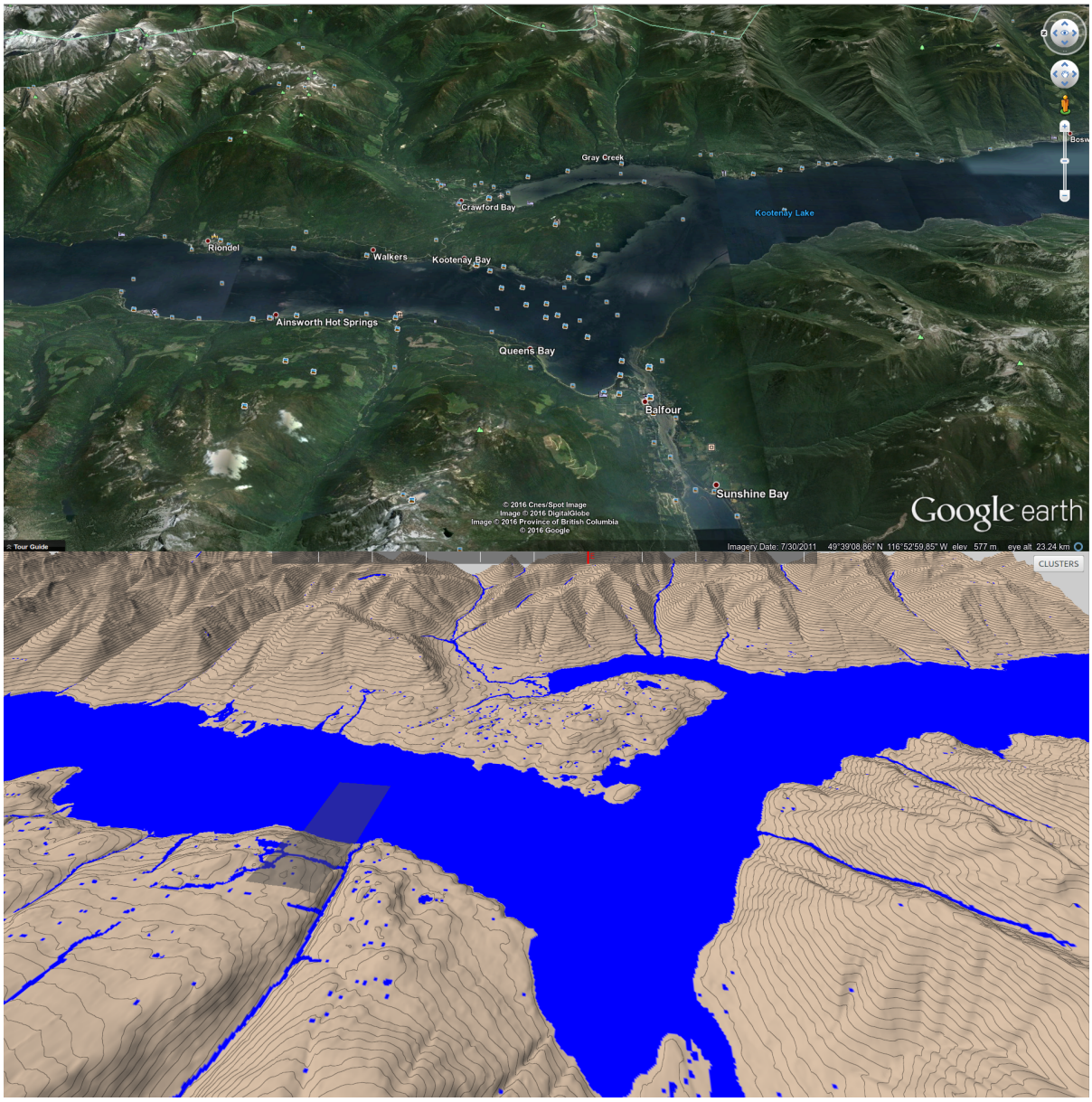

Figure 4.26: Top: Real world water-network at geographic coordinate location: 49°39'N 116°52'W. Bottom: Replica using the water-flow simulation.

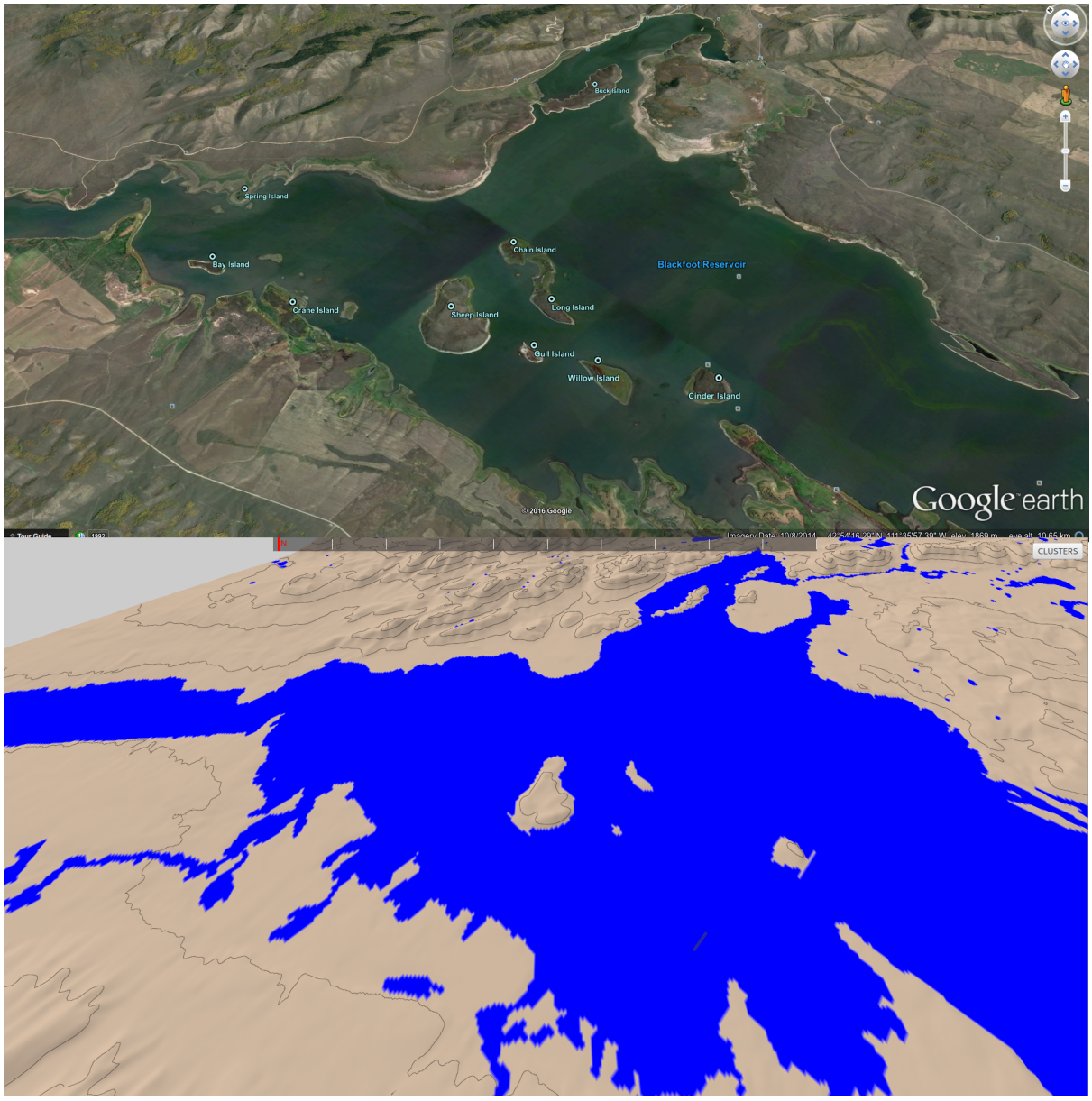

Figure 4.27: Top: Real world water-network at geographic coordinate location: 42°38'N 111°35'W. Bottom: Replica using the water-flow simulation.

4.4 Water Bodies

Water bodies refers here to static water build-ups such as seas, oceans and lakes. The water-flow simulation 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.

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

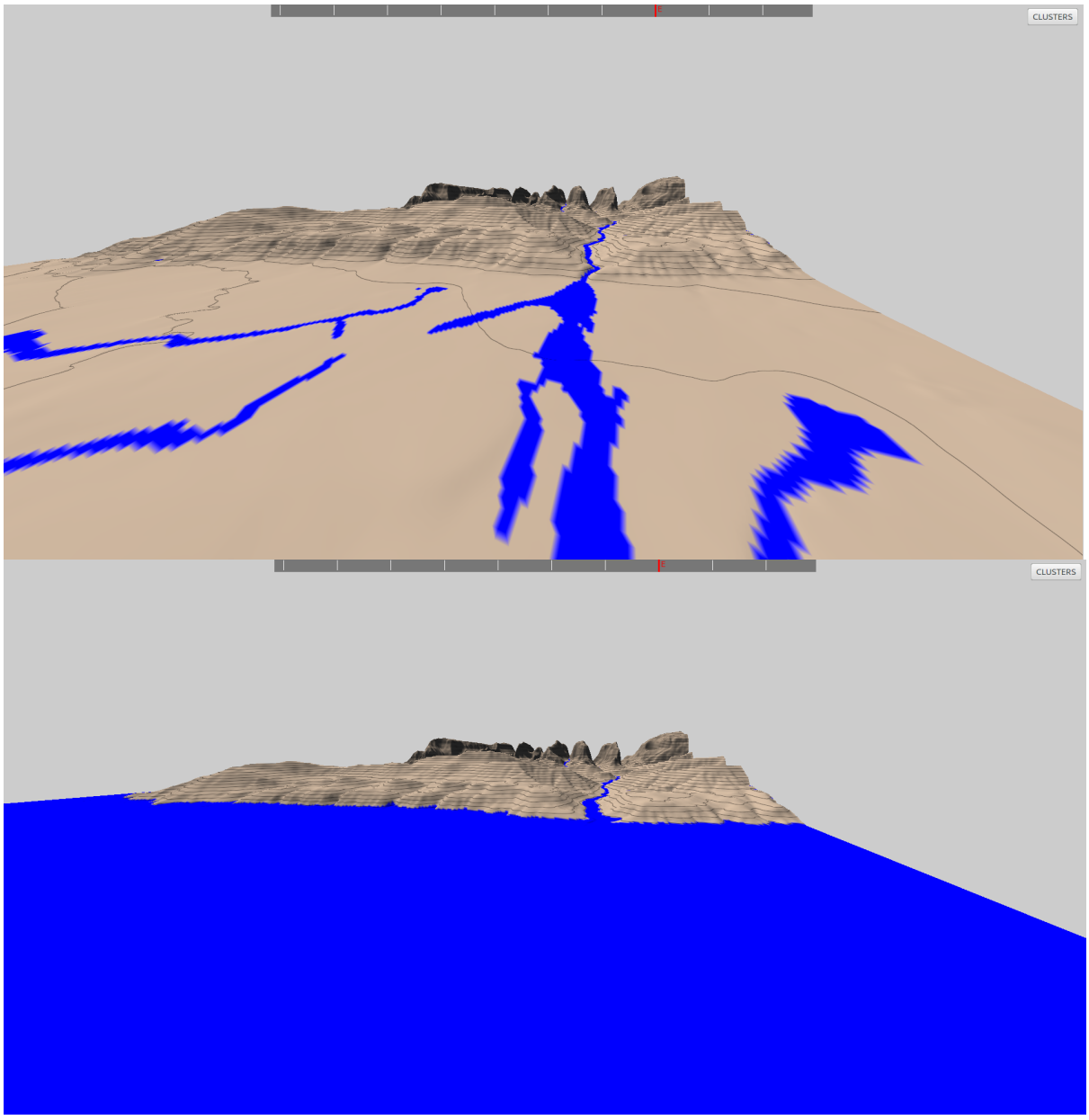

Figure 4.28: Terrain before (top) and after (top) using the flood-fill tool to place a large water body (e.g sea or ocean).

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. However, this simulator is computationally expensive and running it for each terrain vertex 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 . 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 [PC00], image segmentation [BB98, KMN+02] and compression [KMN+02].
The algorithm is of complexity O(N) and can be optimized to make use of parallel architectures [**?**]. 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 [**?**]. 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 5.1.2 and 5.1.3 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 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;

5.1.2 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. Because of this, the value of K is required as input from the user before the clustering is performed.

5.1.3 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 [**?**]. 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.

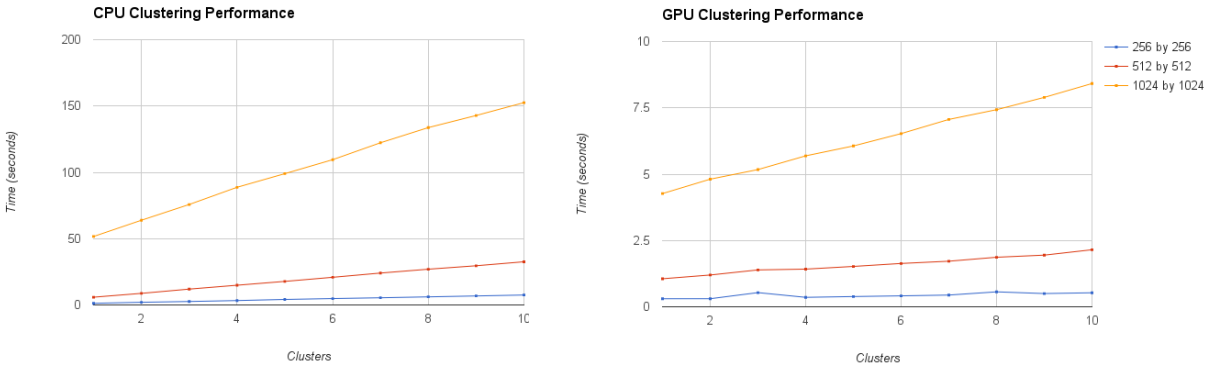

Figure 5.1: Time it takes for the clustering process to complete on the CPU (left) and GPU (right) in relation to the number of clusters. The analysis was performed for terrains of size: 256 by 256 (blue), 512 by 512 (red) and 1024 by 1024 (orange).

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.

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.

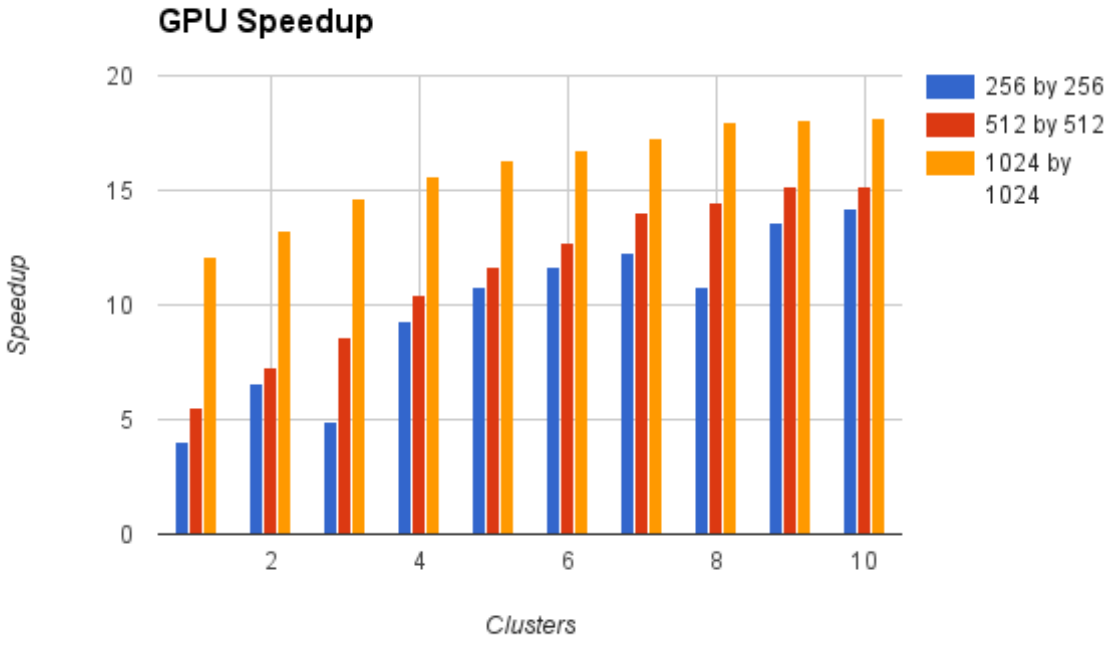

Figure 5.2: Calculated clustering speed-up of the GPU over CPU implementation for square terrains of size 256 by 256 (blue), 512 by 512 (red) and 1024 by 1024(yellow).

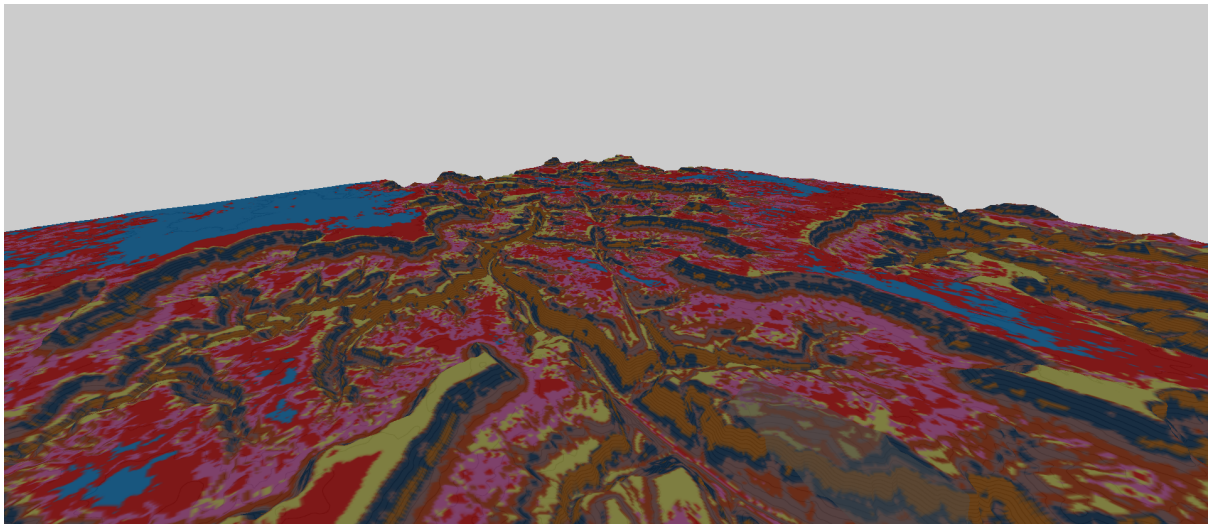

Figure 5.3: Colour coded cluster overlay. Using this it is possible to easily identify the clusters associated to each terrain vertex.

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`

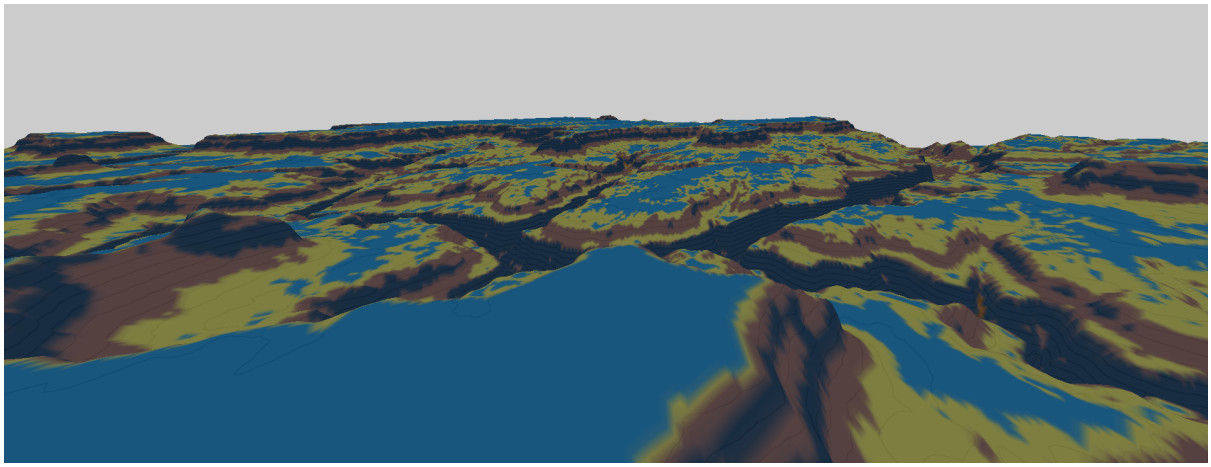

Figure 5.4: Clustering test: Resulting terrain clusters

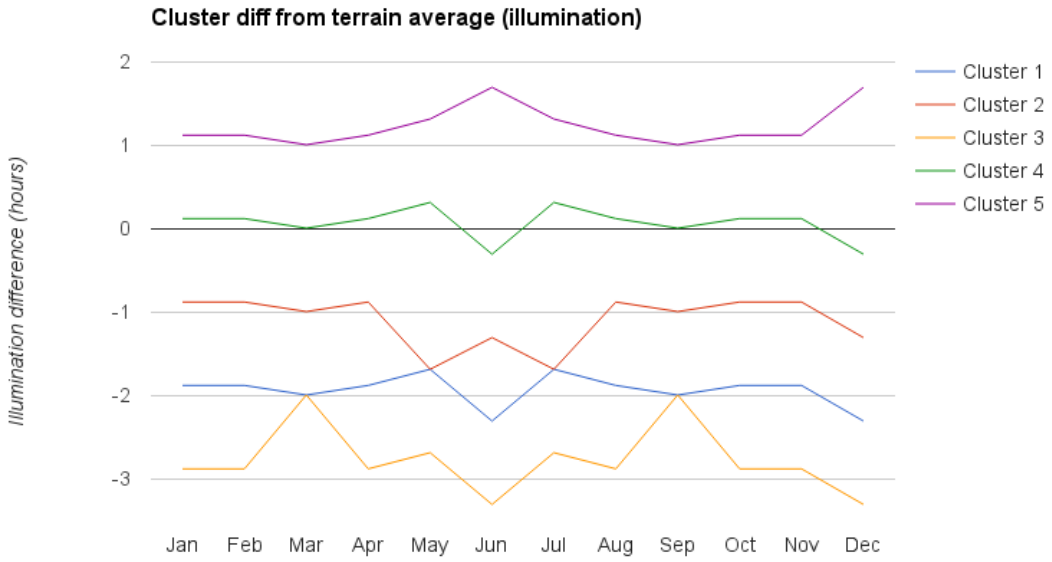

Figure 5.5: Monthly illumination for each cluster and the average over the whole terrain.

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.

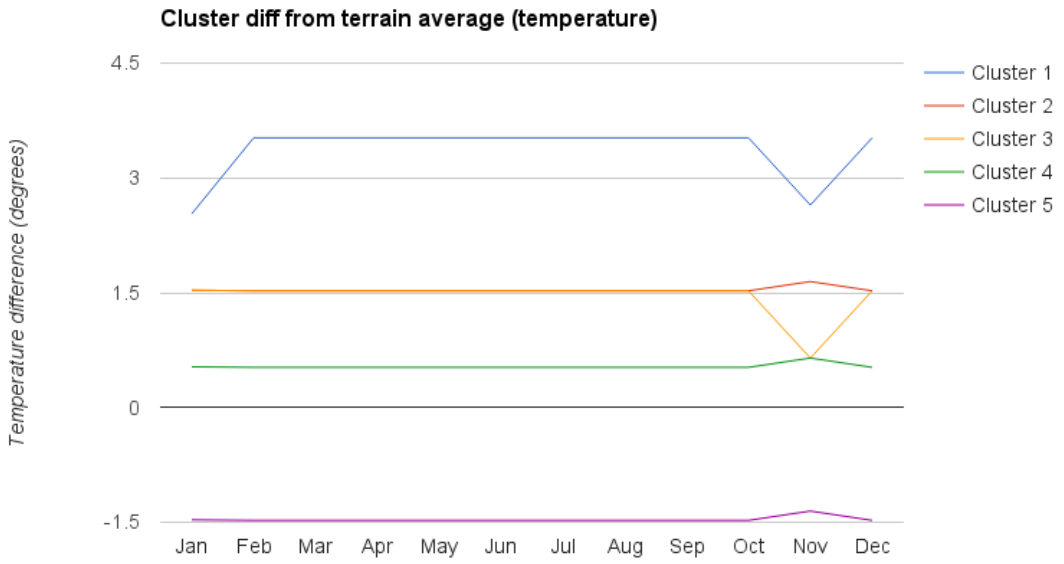

Figure 5.6: Monthly temperature for each cluster and the average over the whole terrain. Cluster 2 has the same values as cluster 4.

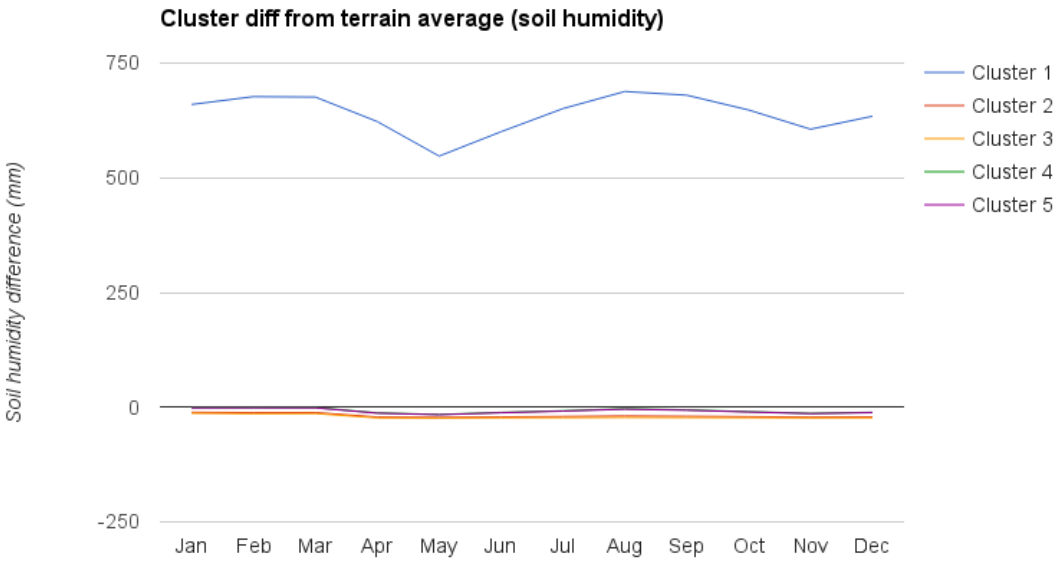

Figure 5.7: Soil humidity for each cluster (same for every month) and the average over the whole terrain.

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

From figure 5.6, 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 altitude causes the temperature to be higher, the canyon walls causes the illumination to be much lower, the river stream causes the soil humidity to be extremely high and the flat bottom causes the slope to be very low.
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 mean, hence its limited variance.
Cluster 5 is constitutes the areas at high altitudes in the canyon where the surface is flat, illumination is high (nothing to shade it) and temperatures are low.

Chapter 6

Vegetation

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 determine suitable vegetation and, subsequently, generate plausible terrains.

To determine environments suited for given species, they are configured with associated resource requirements as outlined in *Plant Species*. Given these properties, it is possible to automatically filter out ill-suited plants from being suggested to the user. Information about this automatic filtering feature is outlined in *Plant Suitability Filtering*.

Although a multitude of plants can grow in a given environment, some will naturally strive more than others. This can be because resources are more adequate or they have a faster, more aggressive growth rate. To model this intra-specie battle for resources and determine a suitable vegetative state, an ecosystem simulator is used. Details of which can be found in *Ecosystem Simulator*.

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 duration. To accelerate the process, the ecosystem simulator is run on 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 are discussed in *Plant Distribution Analysis and Reproduction*.

Property	Value	Unit
Slope	Maximum Slope	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

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 given environments and, subsequently, deduce a plausible distribution using the ecosystem simulator. Details about these, how they can be edited and new species added is discussed in *Specie Properties* and *Storing Species* respectively.

6.1.1 Specie Properties

When configuring a new specie it is necessary to specify with it a set of properties which are used to link it to given suitable environments. These properties are discussed in detail below and summarized in table 6.1.1.

6.1.1.1 Slope

Steep slopes cause essential water and soil nutrients to run-off, making them less rich and, therefore, less suited to plant growth. The slope angle also causes larger species to struggle to

support their own biomass. For this reason, steeper slopes often cater for smaller plant species (grass, shrub, etc.). To model the effect of slope on given plant species, when configuring a new plant specie an associated *maximum slope* must be specified.

6.1.1.2 Growth

To model the growth of a plant specie in the ecosystem simulator, it is necessary to specify: *Maximum height*, *maximum canopy width* and *maximum root size*. Using this along with the specie's ageing properties (see 6.1.1.3), it is possible to simulate the plants vertical growth (height), horizontal growth (canopy) and root coverage. Determining a plant's height and canopy width is also used to determine the shade it project's on other plants during the simulation. Modelling 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.

6.1.1.3 Ageing

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 it's surrounding plants.

6.1.1.4 Seeding

It is necessary to replicate the spawning of offspring in the ecosystem simulator for two core reasons:

1. *Propagation*: Plants propagate on a terrain by producing new offspring which attempt to spawn and invade different areas.
2. *Succession*: New plants spawn to later succeed older and weaker plants of the same specie.

Depending on the specie, plants spawn new offspring by producing seeds or spores. Although biologically different, both can be considered identical for the sole purpose of modelling propagation and succession.

The number of seeds that a given plant creates annually is specie-dependent and therefore must accompany the specie's configuration.

Different species use different techniques to propagate their seeds. For example, some use fruit as a mechanism to propagate using animals digestive systems, some are coated with a

sticky mucous to stick to the fur of animals passing by, some use the wind and some rely solely on gravity to take the seeds to the ground directly below. The seeding mechanism used directly affects the distance which can be achieved between parent and offspring and, to emulate this in the system, a *maximum seeding distance* is configured.

6.1.1.5 Illumination

Illumination has a big impact on plant growth. Whereas some species strive in the shaded undergrowth, others require direct illumination all year round. To model this, the following illumination values must be configured with each plant specie:

- *Minimum daily illumination*: The minimum daily illumination, in hours, at which a plant of this specie can survive.
- *Prime daily illumination range*: The daily illumination range, in hours, which is optimal for the given specie.
- *Maximum daily illumination*: The maximum daily illumination, in hours, at which a plant of this specie can survive.

6.1.1.6 Humidity (rainfall)

Soil water deposited into the soil by either rainfall or existing groundwater is absorbed by plant roots and is vital to its development and survival. Whereas some species have evolved to survive in arid climates with very little water, others require frequent downpours of rain.
To simplify water requirement specifications of different plant species, it is necessary to configure the amount of rainfall necessary as if this was the plant's only source of water (i.e no groundwater). The following values must be configured to grasp a species water requirements:

- *Minimum monthly rainfall*: The minimum monthly rainfall, in millimetres, at which a plant of this specie can survive.
- *Prime monthly rainfall range*: The monthly rainfall range, in millimetres, which is optimal for the given specie.
- *Maximum monthly rainfall*: The maximum monthly rainfall, in millimetres, at which a plant of this specie can survive.

6.1.1.7 Temperature

Another aspect of climates which greatly affect plant growth is temperature and, in order to model a specie's temperature requirements. the following values need to be configured:

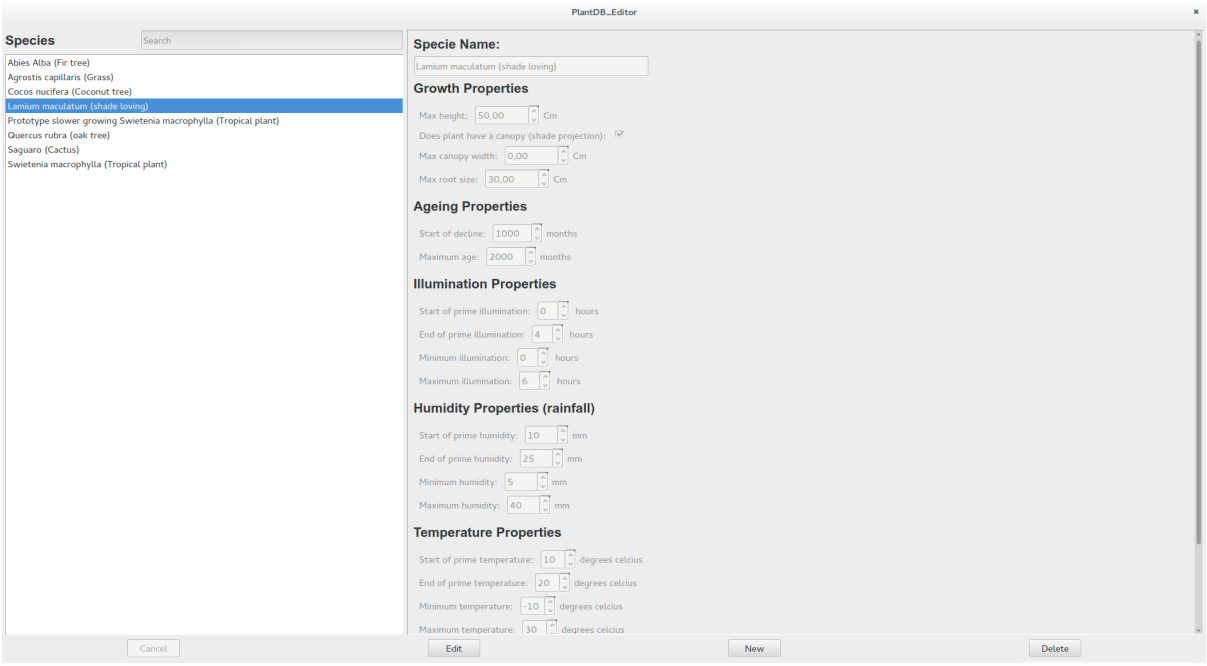

Figure 6.1: Plant database editor tool.

- *Minimum temperature*: The minimum temperature, in degrees, at which a plant of this specie can survive.
- *Prime temperature range*: The temperature range, in degrees, which is optimal for the given specie.
- *Maximum temperature*: The maximum temperature, in degrees, at which a plant of this specie can survive.

6.1.2 Storing Species

All species and associated properties are stored in a database for easy retrieval and filtering. A dedicated tool can be used to interact with the database to add or remove species and edit existing ones (see figure 6.1).

6.2 Plant Suitability Filtering

When the terrain clusters have been generated, the user must specify the plant species to plot. 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 with which to populate the terrain, a filtering pass is performed in order to suggest only the plants which are able to survive. To determine whether a given specie is suited, a *specie suitability score* is calculated for each specie based on the resources of the given cluster. How this score is calculated is described in *Calculating the Specie Suitability Score* below.

As well as being used to filter out ill-suited species, this suitability score also highlights the species which strive in the given environment and could, as a consequence, prove to be useful information for the user when selecting the plants species to select. Various methods are used to effectively communicate the suitability score of each specie to the user, details of which are discussed in *Communicating the Suitability Score* below.

6.2.1 Calculating the Specie Suitability Score

The specie suitability score associated to a given specie S for cluster C, $AggregateScore_S(C)$, illustrates how suited specie S is to the environment of cluster C on a range of 0 (ill-suited) to 100 (perfect conditions). To calculate it, the resource requirements of specie S are matched with the resource availability of cluster C.

To determine this aggregate 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 and is discussed separately below.

6.2.1.1 Slope Suitability Score

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

$$SlopeScore_S(x) = \begin{cases} 100, & \text{if } x \leq max \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

Where: $SlopeScore_S(x)$ is the slope suitability score for specie S given slope x; max is the maximum slope configured for specie S.

6.2.1.2 Illumination Suitability Score

Because the illumination varies on a monthly basis, to calculate the aggregate illumination score for a given specie is is necessary to calculate the *illumination score* for each month as illustrated in equation 6.2.

$$IllumScore_S(x) = \begin{cases} 0, & \text{if } x \leq min \text{ or } x \geq max \\ \frac{x-min}{prime_{start}-min} \times 100, & \text{if } x \in]min, prime_{start}[\\ 100, & \text{if } x \in [prime_{start}, prime_{end}] \\ (1 - \frac{x-prime_{end}}{max-prime_{end}}) \times 100, & \text{if } x \in]prime_{end}, max[\end{cases} \tag{6.2}$$

Where: $IllumScore_S(x)$ is the illumination suitability score for specie S given illumination x; min is the minimum illumination configured for specie S;max is the maximum illumination configured for specie S; $prime_{start}$ is the start of the prime illumination range configured for specie S;$prime_{end}$ is the end of the prime illumination range configured for specie S.

Given the illumination scores for each month, it is possible to calculate the *average illumination score* using equation 6.3.

$$AvgIllumScore_S(x) = \begin{cases} \frac{\sum_{m=1}^{m=12} IllumScore_S(m)}{12}, & \text{if } IllumScore_S(m) > 0 \text{ for } m \in [1, 12] \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

6.2.1.3 Humidity Suitability Score

The humidity also varies on a monthly basis and, as such, it is also necessary to calculate the humidity score for each month as illustrated in equation 6.4.

$$HumScore_S(x) = \begin{cases} 0, & \text{if } x \leq min \text{ or } x \geq max \\ \frac{x-min}{prime_{start}-min} \times 100, & \text{if } x \in]min, prime_{start}[\\ 100, & \text{if } x \in [prime_{start}, prime_{end}] \\ (1 - \frac{x-prime_{end}}{max-prime_{end}}) \times 100, & \text{if } x \in]prime_{end}, max[\end{cases} \tag{6.4}$$

Where: $HumScore_S(x)$ is the humidity suitability score for specie S given humidity x; min is the minimum humidity configured for specie S; max is the maximum humidity configured for specie S; $prime_{start}$ is the start of the prime humidity range configured for specie S; $prime_{end}$ is the end of the prime humidity range configured for specie S.

Given the humidity scores for each month, it is possible to calculate the *average humidity score* using equation 6.5.

$$AvgHumScore_S(x) = \begin{cases} \frac{\sum_{m=1}^{m=12} HumScore_S(m)}{12}, & \text{if } HumScore_S(m) > 0 \text{ for } m \in [1, 12] \\ 0, & \text{otherwise} \end{cases} \tag{6.5}$$

6.2.1.4 Temperature Suitability Score

Temperature also varies on a monthly basis and, as such, it is also necessary to calculate the temperature score for each month as illustrated in equation 6.6.

$$TempScore_S(x) = \begin{cases} 0, & \text{if } x \leq min \text{ or } x \geq max \\ \frac{x-min}{prime_{start}-min} \times 100, & \text{if } x \in]min, prime_{start}[\\ 100, & \text{if } x \in [prime_{start}, prime_{end}] \\ (1 - \frac{x-prime_{end}}{max-prime_{end}}) \times 100, & \text{if } x \in]prime_{end}, max[\end{cases} \tag{6.6}$$

Where: *TempScore$_S$(x)* is the temperature suitability score for specie S given temperature x;*min* is the minimum temperature configured for specie S; *max* is the maximum temperature configured for specie S; *prime$_{start}$* is the start of the prime temperature range configured for specie S; *prime$_{end}$* is the end of the prime temperature range configured for specie S.

Given the temperature scores for each month, it is possible to calculate the *average temperature score* using equation 6.7.

$$AvgTempScore_S(x) = \begin{cases} \frac{\sum_{m=1}^{m=12} TempScore_S(m)}{12}, & \text{if } TempScore_S(m) > 0 \text{ for } m \in [1, 12] \\ 0, & \text{otherwise} \end{cases} \tag{6.7}$$

6.2.1.5 Specie Suitability Score

The suitability score gives an overview of the specie's suitability and is calculated using equation 6.8

$$Score_S(s, i, h, t) = \begin{cases} \frac{SlopeScore_S(s)+AvgIllumScore_S(i)+AvgHumScore_S(h)+AvgTempScore_S(x)}{4}, & \text{if} \\ TempScore_S(m) > 0 \text{ and } AvgIllumScore_S(m) > 0 \text{ and} \\ AvgHumScore_S(m) > 0 \text{ and } AvgTempScore_S(m) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.8}$$

6.2.2 Communicating the Specie Suitability Score

When all the terrain clusters have been created, the terrain suitability score for each specie 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 specie is automatically filtered out to prevent the user from selecting it.

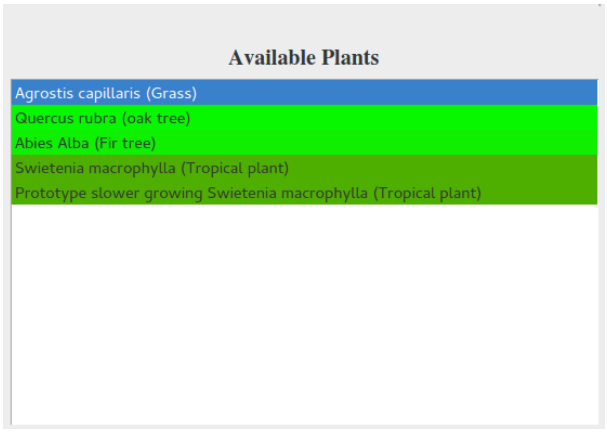

Figure 6.2: Color coding for specie suitability. The greener the highlight the more suited the specie is to the environment.

Color coding is used to make further use of the specie suitability score and intuitively communicate to the user the species most suited to the environment (see figure 6.2).

When the user selects a given specie, all intermediate scores which were used to calculate the *specie suitability score* are communicated to the user in histogram form (see figure 6.3).

Figure 6.3: Average (top) and temperature (bottom) intermediate specie suitability histograms. Not displayed but also generated are the illumination and humidity intermediate specie suitability histograms.

6.3 Ecosystem Simulator

Once the user selects the union of all species that must appear in all clusters of the terrain, it is necessary to determine a valid vegetation distribution for each cluster. To do so, an ecosystem simulator is used similarly to that in the work by Deussen et al [DHL+98] and Lane and Przemyslaw [LP02]. Unlike these other ecosystem simulators, however, it isn't based on L-Systems and models both resource requirements and resource availability in greater detail. The purpose of ecosystem simulator is to determine, given a vegetative state, S_t at time t, the vegetative state S_{t+n} at time $t+n$, for any value of n.

To do so, the simulation advances through time in monthly intervals and the strength of all plant instances are re-calculated at each iteration. Their strength depend not only on resource properties of the given month but also surrounding plants as they battle for these resources. Determining the set S = {P_1, P_2, P_3, ...} of plants which compete for resources with plant P_n depends on the spatial reach of P_n. Spatial awareness is therefore a key requirement of the simulation which is achieved by splitting the simulation area into a grid of cells as described in *Gridded Simulation Area*.

Within each cell of the gridded simulation area, resources must be distributed to the different plant instances present. How this is done is described in *Resource Distribution*.

Given the resources allocated to each plant instance, it is possible to calculate their strength. This is used as a representation of the plant's health and, consequentially, it's ability to survive and grow. Details about the plant's strength calculation and it's usage are discussed in *Plant Strength Calculation* and *Plant Strength Usage* respectively.

On an annual basis, new plant instances are spawned based on the specie's seeding properties. How this is done is discussed in *Spawning Plants*.

To conclude the discussion on the ecosystem simulator, performance and results will be analysed and discussed.

6.3.1 Gridded Simulation Area

The simulation window 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. To model plant's interacting and battling for resources, the window is split into cells to form a grid

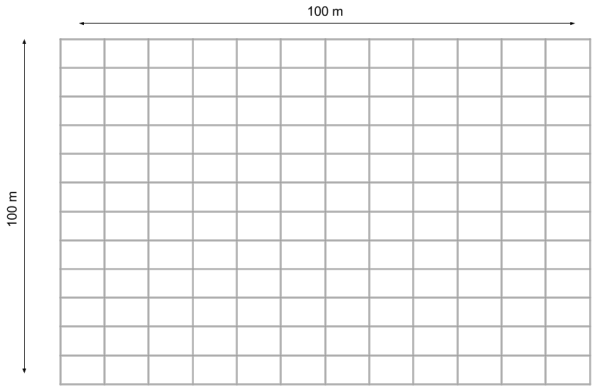

Figure 6.4: Gridded simulation area.

as illustrated in figure 6.4.

The size of individual cells can be configured to increase/decrease the resolution and, therefore, the accuracy of the simulation. As the simulation progresses, plant's grow, their spatial coverage increases, and they enter new grid cells. When a plant enters a new grid cell, it becomes a member of it and cell resources must be distributed to it as well as to all other plants in the cell. The information associated to 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 is comprised of: the *soil humidity* and the *illumination*. The simulation-dependent information changes throughout the simulation as plants spawn, die and grow and is comprised of: the list of plants whose roots intersect the cell and the list of plants whose canopy intersects the cell.

6.3.2 Resource Distribution

The strength of each plant in the simulation must be recalculated on a monthly basis. To determine the strength of a given plant, it is necessary to know the illumination and humidity allocated to it along with the temperature. The temperature is not a distributable resource and is identical for all plant instances given the month. The allocated humidity and illumination is determined by averaging the resources distributed to it in each cell it overlaps in the grid. So, before the strengths of individual plant instances can be calculated, first each cell of the grid must be iterated over and illumination and humidity distributed to the plants contained within. How these are allocated is discussed below.

6.3.2.1 Humidity Distribution

Plants grow there roots in order to access the nutrients and moisture available in the surrounding soil. As roots of different plants overlap, they start to compete for these resources.

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

The humidity is deemed abundant if the available humidity, $H_{available}$, surpasses 300 millimetres. In this situation, the humidity is deemed to be enough for there to be standing water and therefore all plants of S are allocated $H_{available}$.

If $H_{available}$ is less than 300 millimetres, it is necessary to determine whether the humidity is sufficient or insufficient by calculating the requested humidity $H_{requested}$ as outlined in equation 6.9.

$$H_{requested} = \sum MinHumidity(P_n) \text{ for } n \in S \tag{6.9}$$

Where:*MinHumidity(P_n)* is the minimum humidity requirement of the specie to which plant P_n belongs;S is the set of plants whose roots intersect with the given grid cell.

If $H_{requested}$ is less than $H_{available}$, the humidity is deemed sufficient and the amount allocated to each plant is calculated as described in equation 6.10. If $H_{requested}$ is more than $H_{available}$, however, the humidity is deemed insufficient and the allocation is done following equation 6.11

$$\begin{aligned} H_{allocated}(P_n) &= MinHumidity(P_n) + OverFlow \\ OverFlow &= H_{available} - \sum MinHumidity(P_n) \text{ for } n \in S \end{aligned} \tag{6.10}$$

Where:*$H_{allocated}(P_n)$* is the humidity allocated to plant P_n; *MinHumidity(P_n)* is the minimum humidity requirement of the specie to which plant P_n belongs;S is the set of plants whose roots intersect with the given grid cell.

Intuitively, it allocates each plant with the minimum amount of humidity it requires to survive plus the resulting overflow.

$$\begin{aligned} H_{allocated}(P_n) &= min(MinHumidity(P_n), Vigor(P_n) \times H_{remaining}) \\ Vigour(P_n) &= \frac{RootSize(P_n)}{\sum RootSize(P_x) \text{ for } x \in S} \\ H_{remaining} &= H_{available} - (\sum H_{allocated}(P_x) \text{ for } x \in S_{processed}) \end{aligned} \tag{6.11}$$

Where: The plants of S **must** be iterated over in decrementing order of their vigor as this will affect the water they are allocated; $H_{allocated}(P_n)$ is the humidity allocated to plant P_n; $MinHumidity(P_n)$ is the minimum humidity requirement of the specie to which plant P_n belongs; $Vigour(P_n)$ is the vigor of plant P_n in comparison to other plants present in the cell. It is estimated based on root size; $RootSize(P_n)$ is the root size of plant P_n; $S_{processed}$) is the set of plants from S whose water allocation has already been calculated; S is the set of plants whose roots intersect with the given grid cell.

Intuitively, this algorithm prioritises water distribution to more vigorous plant's.

6.3.2.2 Plant Humidity Allocation

To calculate the humidity allocated to plant P_n, it is first necessary to determine the set of grid cells $S = \{C_1, C_2, C_3, ...\}$ which it's roots intersect. Given this, the plants humidity allocation is calculated using equation 6.12.

$$H_n = \frac{\sum H_{allocated}(C_x) \text{ for } x \in S}{|S|} \tag{6.12}$$

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

Intuitively, the humidity allocated to a given plant is simply the average of the humidity allocated to it in all grid cells it's roots intersect.

6.3.2.3 Illumination Distribution

Plants which are heavily dependent on illumination will often grow a large canopy in order to maximize the leaf area receiving direct sunlight. Doing so also restricts the illumination received by smaller plants in the undergrowth. To model the shade projection of larger plants, illumination is distributed in each cell depending on the plants height and canopy width.

Equation 6.13 is used to allocate illumination amongst the set $S = \{P_1, P_2, P_3, ...\}$ of plants which canopy intersects cell C_{xy}.

$$Illumination(P_n) = \begin{cases} C_{illumination}, & \text{if } CanopyWidth(P_n) = 0 for x \in S \\ C_{illumination}, & \text{if } Height(P_n) > height(x) for x \in S : x \neq P_n \\ 0, & \text{otherwise} \end{cases} \tag{6.13}$$

Where: $Illumination(P_n)$ is the illumination allocated to plant P_n whose canopy overlaps with current grid cell;$C_{illumination}$ is the monthly illumination of the cell;$CanopyWidth(P_n)$ is the canopy width of plant P_n; $Height(P_n)$ is the height of plant P_n; S is the set of plants whose canopy intersects with the given grid cell.

Intuitively, if all plants present in the given cell are canopy-free (i.e no shade projection), the equation allocates them all the available illumination. If not all plants are canopy-free, the equation allocates illumination only to the tallest canopy plant.

6.3.2.4 Plant Illumination Allocation

Calculating the illumination allocated to a plant P_n is very similar to calculating the humidity allocation only the set of grid cells $S = \{C_1, C_2, C_3, ...\}$ are those which the plants canopy intersects. Given S, the plant illumination is calculated using equation 6.14.

$$I_n = \frac{\sum I_{allocated}(C_x) \text{ for } x \in S}{|S|} \tag{6.14}$$

Where:I_n is the illumination allocated to plant P_n; $I_{allocated}(C_n)$ is the illumination allocated to plant P_n in grid cell C_n; $|\ S\ |$ is the number of cells in the set S.

Intuitively, the illumination allocated to a given plant is simply the average of the humidity allocated to it in all grid cells it's canopy intersect.

6.3.3 Plant Strength Calculation

The strength of plant P_n is the minimum of S_{age}, $S_{temperature}$, $S_{illumination}$ and $S_{humidity}$ which represent the strength of the plant in terms of it's age, temperature, illumination and humidity respectively. To calculate these values, ranging from negative to positive one hundred, a graph is plotted for each specie based on it's properties as outlined in figures 6.5, 6.6, 6.6, 6.7 and 6.8. Using these graphs, it is possible to calculate the strength of any plant given it's allocated resources.

6.3.4 Plant Strength Usage

The strength of each plant is recalculated on a monthly basis as available resources change and other plants spawn and grow. This value subsequently influences its growth potential and probability of death, as discussed below.

6.3.4.1 Growth Potential

In the simulation, each plant P attempts to grow its roots, its canopy and it's height on a monthly basis. Each specie has an maximum monthly root growth $MaxGrowth_{root}$, canopy growth $MaxGrowth_{canopy}$ and height growth $MaxGrowth_{height}$ which are calculated using the specie's growth and ageing properties as outlined in equations 6.15, 6.16 and 6.17 respectively.

$$MaxGrowth_{root}(S) = \frac{MaxRoot(S)}{Age_{StartOfDecline}(S)} \tag{6.15}$$

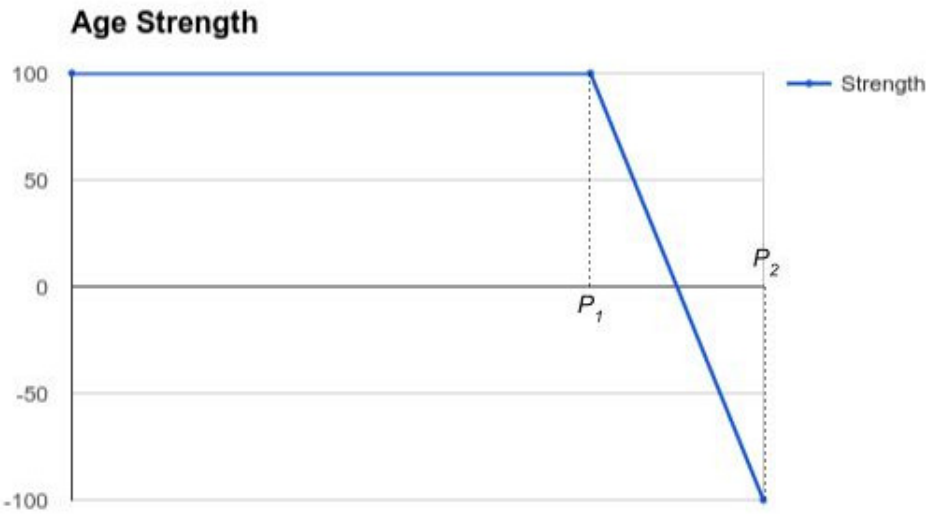

Figure 6.5: Graph used to calculate the age strength of any plant. $\mathbf{P}_1$ is the age of *start of decline* configured for the given specie. $\mathbf{P}_2$ is the *maximum age* configured for the given specie.

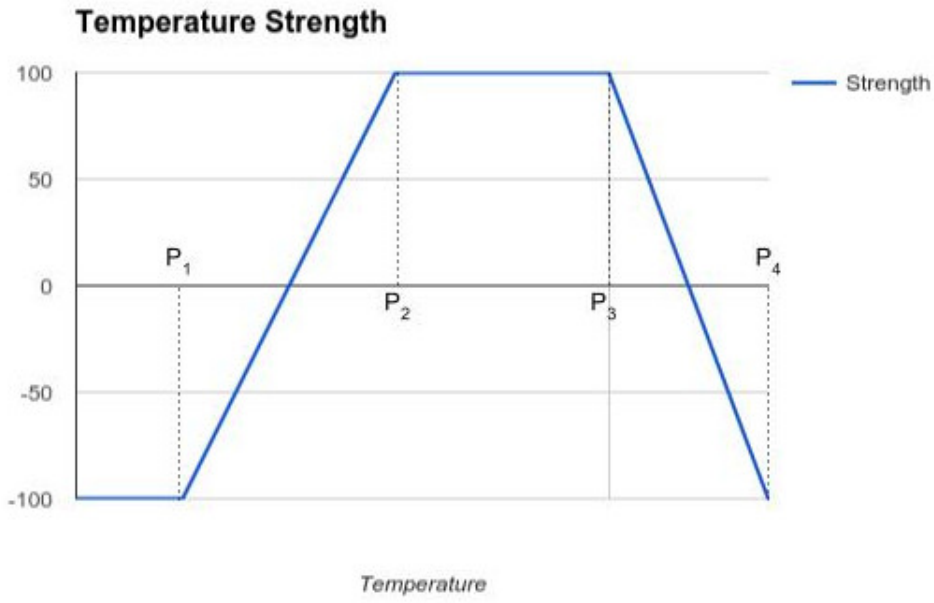

Figure 6.6: Graph used to calculate the temperature strength of any plant. $\mathbf{P}_1$ and $\mathbf{P}_4$ are the *minimum* and *maximum temperature* configured for the given specie. $\mathbf{P}_2$ and $\mathbf{P}_3$ form the *prime temperature range* configured for the given specie.

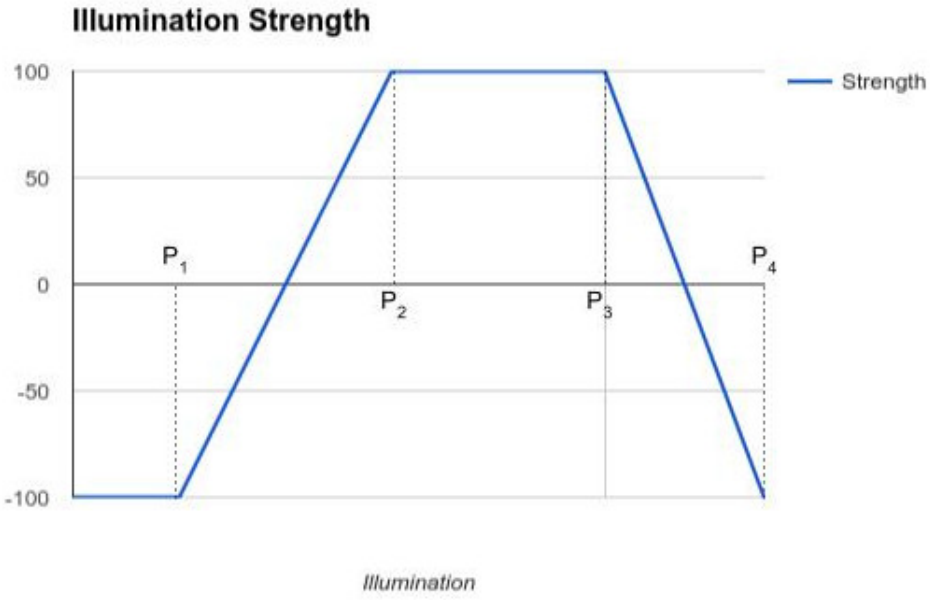

Figure 6.7: Graph used to calculate the illumination strength of any plant. $\mathbf{P}_1$ and $\mathbf{P}_4$ are the *minimum* and *maximum illumination* configured for the given specie. $\mathbf{P}_2$ and $\mathbf{P}_3$ form the *prime illumination range* configured for the given specie.

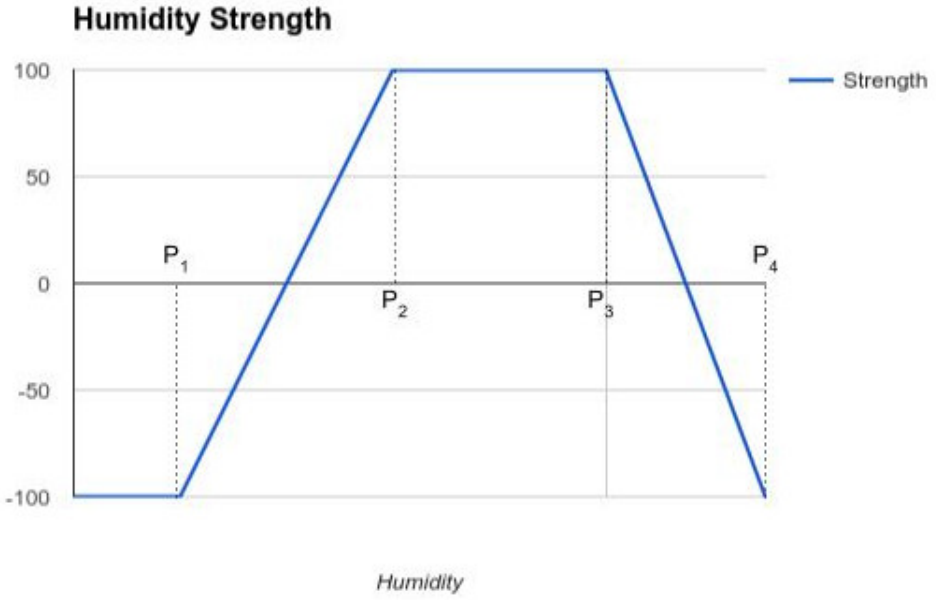

Figure 6.8: Graph used to calculate the humidity strength of any plant. $\mathbf{P}_1$ and $\mathbf{P}_4$ are the *minimum* and *maximum humidity* configured for the given specie. $\mathbf{P}_2$ and $\mathbf{P}_3$ form the *prime humidity range* configured for the given specie.

Where:$MaxGrowth_{root}(S)$ is the maximum monthly root growth of specie S;$MaxRoot(S)$ is the configured maximum root size of the specie S;$Age_{StartOfDecline}(S)$ is the age of start of decline configured for specie S.

$$MaxGrowth_{canopy}(S) = \frac{MaxCanopy(S)}{Age_{StartOfDecline}(S)} \tag{6.16}$$

Where:$MaxGrowth_{canopy}(S)$ is the maximum monthly canopy growth of specie S;$MaxCanopy(S)$ is the configured maximum canopy size of specie S;$Age_{StartOfDecline}(S)$ is the age of start of decline configured for specie S.

$$MaxGrowth_{height}(S) = \frac{MaxHeight(S)}{Age_{StartOfDecline}(S)} \tag{6.17}$$

Where:$MaxGrowth_{height}(S)$ is the maximum monthly height growth of specie S;$MaxHeight(S)$ is the configured maximum height of specie S;$Age_{StartOfDecline}(S)$ is the age of start of decline configured for specie S.

The actual root growth $Growth_{root}$, canopy growth $Growth_{canopy}$ and height growth $Growth_{height}$ is directly dependent on the plant's strength, however, and is calculated using equations 6.18, 6.19 and 6.20 respectively. The maximum growth is achieved only if the plant is at its full strength. Note that no plants grow if there current strength is negative as they are deemed in a *survival state*.

$$Growth_{root}(P,S) = max(0, Strength(P) \times MaxGrowth_{root}(S) \tag{6.18}$$

Where:$Growth_{root}(S)$ is the monthly root growth of plant P of specie S;$Strength(P$ is the current strength of P;$MaxGrowth_{root}(S)$ is the maximum monthly root growth calculated for specie S.

$$Growth_{canopy}(P,S) = max(0, Strength(P) \times MaxGrowth_{canopy}(S) \tag{6.19}$$

Where:$Growth_{canopy}(S)$ is the monthly canopy growth of plant P of specie S;$Strength(P$ is the current strength of P;$MaxGrowth_{canopy}(S)$ is the maximum monthly canopy growth calculated for specie S.

$$Growth_{height}(P,S) = max(0, Strength(P) \times MaxGrowth_{height}(S) \tag{6.20}$$

Where:$Growth_{height}(S)$ is the monthly height growth of plant P of specie S;$Strength(P$ is the current strength of P; $MaxGrowth_{height}(S)$ is the maximum monthly height growth calculated for specie S.

6.3.4.2 Probability of Death

On a monthly basis, the probability of death of each plant is calculated based on it's strength using equation 6.21 and the plant killed with the said probability. Note that a plant P will only be susceptible to be killed off if it's strength is negative.

$$Probability_{death}(P) = max(0, \frac{-1 \times Strength(P) + counter}{100}) \tag{6.21}$$

Where:*Probabilitydeath(P)* is the probability of death of plant P;*Strength(P* is the current strength of P;*counter* is a value which increases by ten each month the plant's strength is negative and resets to zero when it becomes positive. This is to prevent plant's from surviving with a continuous negative strength for too long.

6.3.5 Spawning Plants

In nature, the spawning of new plants ensures specie *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 specie and are positioned either randomly or at predefined positions. The number of seeds that are produced for a given specie is determined by the specie's *annual seed count* configuration.

Different seeding mechanisms are used in the simulator depending on the number of plant's of the given specie present in the simulation, S_{count}, and it's illumination properties.
If S_{count} is greater than zero, existing plant instances are used to determine the location of new plants, irrespective of it's illumination requirements, as described in *Spawning from Existing Plants*.
If S_{count} is zero, the seeding mechanism depends on the specie's illumination requirements. If the specie is shade loving, *canopy seeding* is employed, else *random seeding*. Both are discussed below.

6.3.5.1 Spawning from Existing Plants

To ensure specie propagation, when plant's of the given specie 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 if the number of plants of the given specie is less than the number of seeds to produce, a single plant is used to produce multiple seed locations.

This technique effectively ensures *propagation* until the number of plant instances present is larger than the number of seeds to produce. At which point, the *propagation* potential decreases as the plant count increases. This is because as the selection pool for the random plants increases in size, the probability of selecting a plant at a location which will permit propagation decreases. To overcome this and ensure the initial seeding plants that are selected span a wide area of the simulation window, they are selected at from individual simulation grid cells.

6.3.5.2 Canopy Seeding

Shade-loving plants strive in the shaded undergrowth. If shade-loving plants are spawned at random locations in the simulation, the probability of them landing under the canopy of an existing plant is extremely low. To overcome this, shade-loving plants are not spawned at random but under the canopy of randomly selected plants.

6.3.5.3 Random Seeding

This is the most simple form of seeding and is used when no plants of the given specie are present in the simulation and the specie is not shade-loving. The set of locations for new plants to spawn is selected at random within the simulation window.

6.3.6 Performance

The number of plants present in the simulation will heavily influence it's performance as the strength of each plant needs to be recalculated on a monthly basis. To test the influence of plant count on simulation time, a simulation is run with a single specie of plant and the monthly processing time analysed alongside the number of plants present. The plant used is grass at is has no canopy and very minimal root coverage, permitting a large number of instances to grow simultaneously (see appendix B for properties of 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 increase. The results are summarized in Figure 6.9 and show that the processing time increases linearly with plant count.

Another simulation property which heavily impacts performance is the root and canopy growth of plant species present. The reason for this is that, as plant's roots and canopy grow, they will cover more simulation grid cells and more calculations will be required per individual cell when the contained resources are distributed. To analyse the impact of plant growth, a base specie S_{base} is created with a given root and canopy growth rate. Then, two species are created S_{X2} and S_{X3} with twice and thrice the growth rates of S_{base} respectively (see appendix B for

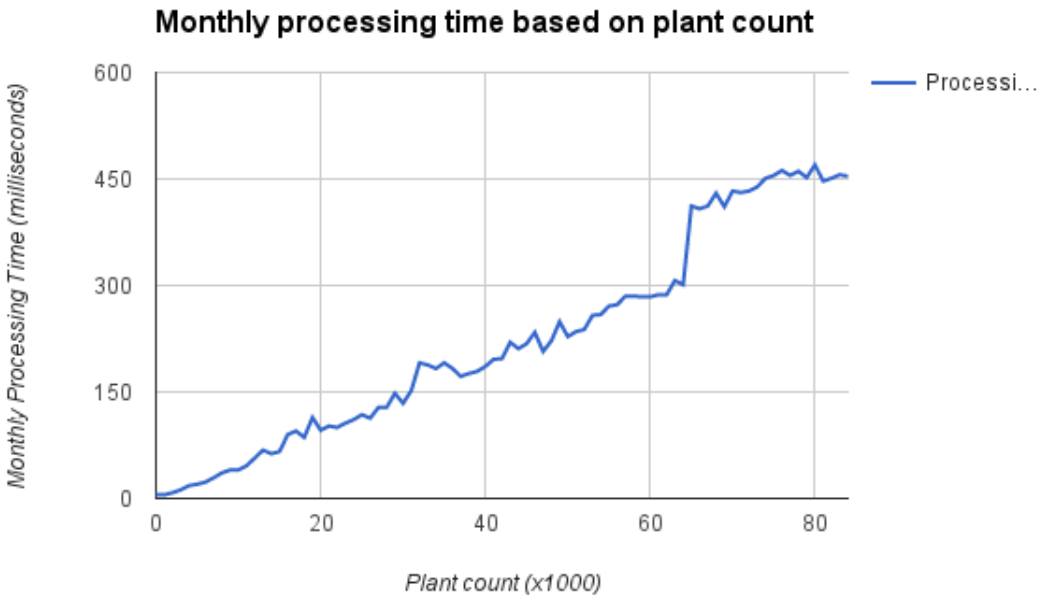

Figure 6.9: Processing time based on plant count. Total simulation time for 100 years: 271 seconds

specie details). An identical simulation is run on each specie in terms of available resources and, on a monthly basis, the number of plants present in the simulation along with the monthly processing time analysed. Given this information, it is possible to track the average monthly processing time per plant throughout the simulation. It is important to normalise based on the number of plants as the faster growing plants will permit less plants to survive for the simple reason that they will require and be able to access resources from a larger amount of grid cells. As can be seen in the results plotted in figure 6.10, the processing times are similar to start and then increase proportionally to the species growth rate.

6.3.7 Results

To test the resulting spatial distribution of plant communities in their work, Lane and and Przemyslaw [LP02] attempt to reproduce three important properties of nature: *Self-thinning*, *succession* and *propagation*.

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.

Given plant specie A with a fast growth rate and specie B with a slower growth rate but higher shade tolerance. At first, the faster growing specie A will dominate and flourish but, with time,

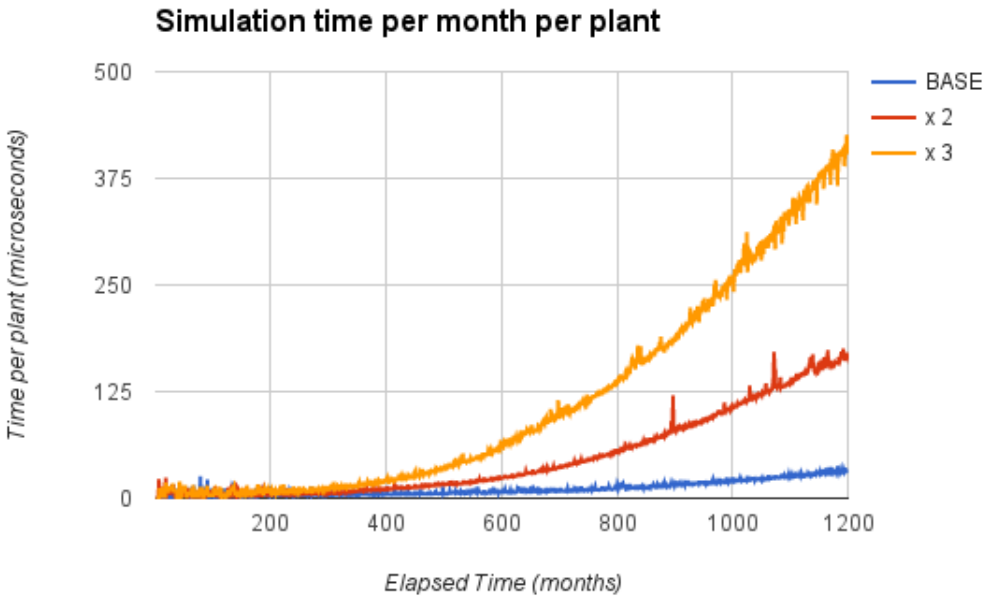

Figure 6.10: Evolution of the monthly processing time normalised based on plant count. The processing time increases as the plant's grow larger as they cover more grid cells. Total simulation time for one hundred years: 49 seconds for S_{base}, 122 seconds for S_{X2} and 166 seconds for S_{X3}

Simulation	Simulation time (years)	Humidity	Illumination	Temperature
1	100	25	10	15
2	100	30	10	15
3	100	35	10	15

Table 6.1: Self-thinning test simulation configurations. For simplicity, monthly resources are kept constant.

the slower growing but more shade tolerant specie B will flourish and dominate. This is the *succession* property.
Plants *propagate* in clusters surrounding the seeding plant.

To test the ecosystem simulator, we will first employ the same methodology as Lane and and Przemyslaw [LP02] and attempt to reproduce *self-thinning*, *succession* and *propagation*.

To ensure a given plant specie strives better in its optimal environment, the same simulation will be run using a single plant specie with varying resource properties and the resulting plant distribution analysed. This test is discussed in *Varying Resources Test* below.

A *shade-simulation test* is performed to ensure plants which depend on direct illumination strive less in the shaded undergrowth of larger plants.

To ensure shade-loving plants are able to propagate and strive under the canopies of larger plants, a *shade-loving plant test* is also performed.

6.3.7.1 Self-thinning Test

Self-thinning occurs when the plant biomass surpasses a given tipping point where available resources become insufficient to permit further plant growth. At this point, larger plants start to kill off smaller, weaker plants by stealing their resources.

To test whether self-thinning is successfully modelled in the ecosystem simulator, three simulations are run as described in table 6.1 and the plant count tracked throughout. As described previously, self-thinning occurs because of insufficient resources. By modifying only available humidity in each simulation, it's effect on self-thinning becomes apparent. As can be seen in the results summarized in figure 6.11, the plant count increases at first, reaches a maximum and decreases thereafter. This is the exact behaviour of self-thinning. Furthermore, it is apparent that the maximum plant count increases with the humidity available, therefore showing that self-thinning is sensitive to available resources.

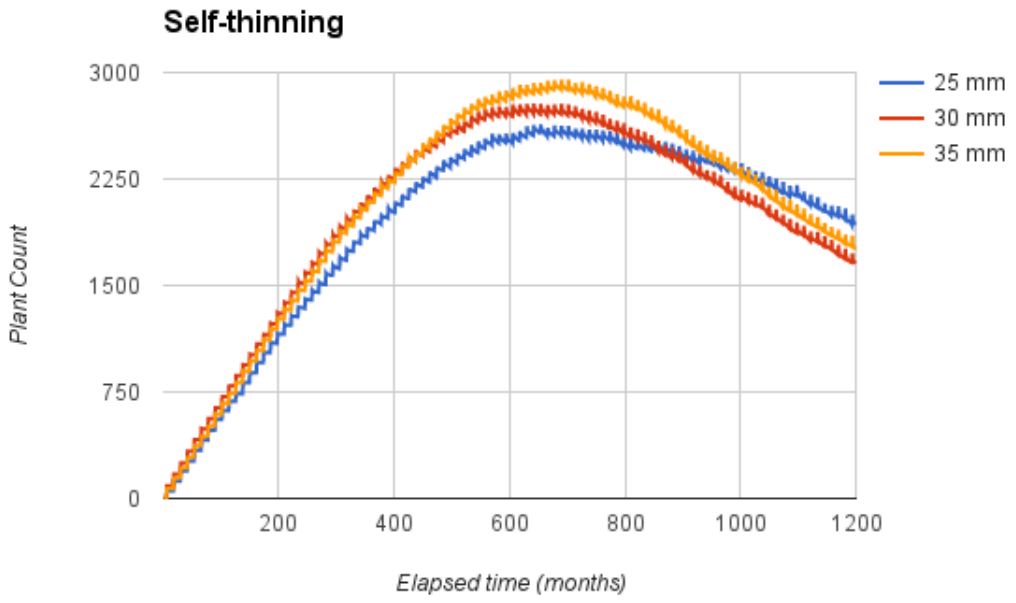

Figure 6.11: Plant count tracked throughout three separate simulations differing only in available humidity.

6.3.7.2 Succession Test

Succession occurs in an ecosystem due to the different growth rates of the species it contains. To test *succession* in the ecosystem simulator, two plant species 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 analytical results illustrated in figure 6.12 along with the appearance at ten year intervals displayed in figure 6.13 shows that S_{fast} dominates at first (300 months in) followed by S_{slow} (500 months in). A balance is found thereafter.

6.3.7.3 Propagation Test

To ensure propagation is modelled in the ecosystem simulator a simulation is run with a single starting grass seed (see appendix B for specie details) and it's evolution tracked throughout. Figure 6.14 shows that iterative propagation through annual seeding enables a single seed plant to colonize the entirety of the terrain.

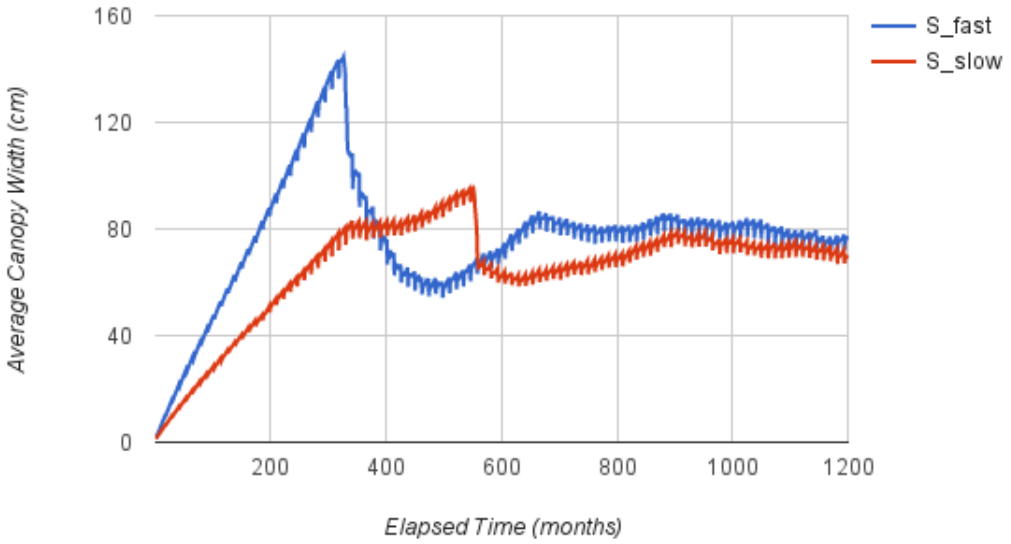

Figure 6.12: Succession Test: Average size of the slow growing S_{slow} (red) and fast growing S_{fast} (blue) throughout a simulation run in optimal conditions.

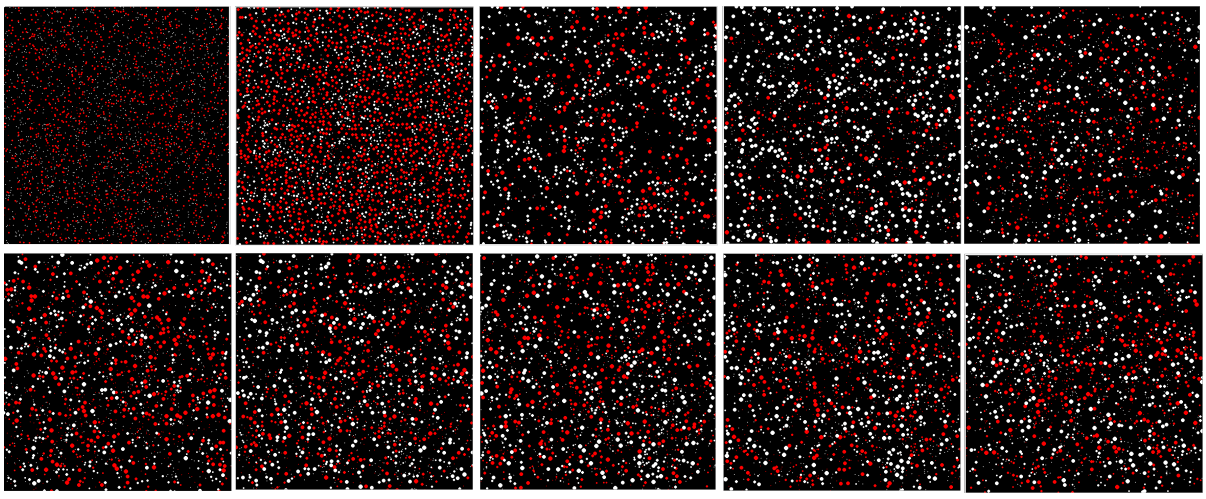

Figure 6.13: Succession Test: Appearance of the slow growing S_{slow} (white) and fast growing S_{red} (blue) at different times during the simulation. From left-to-right, top-top-bottom: 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 years.

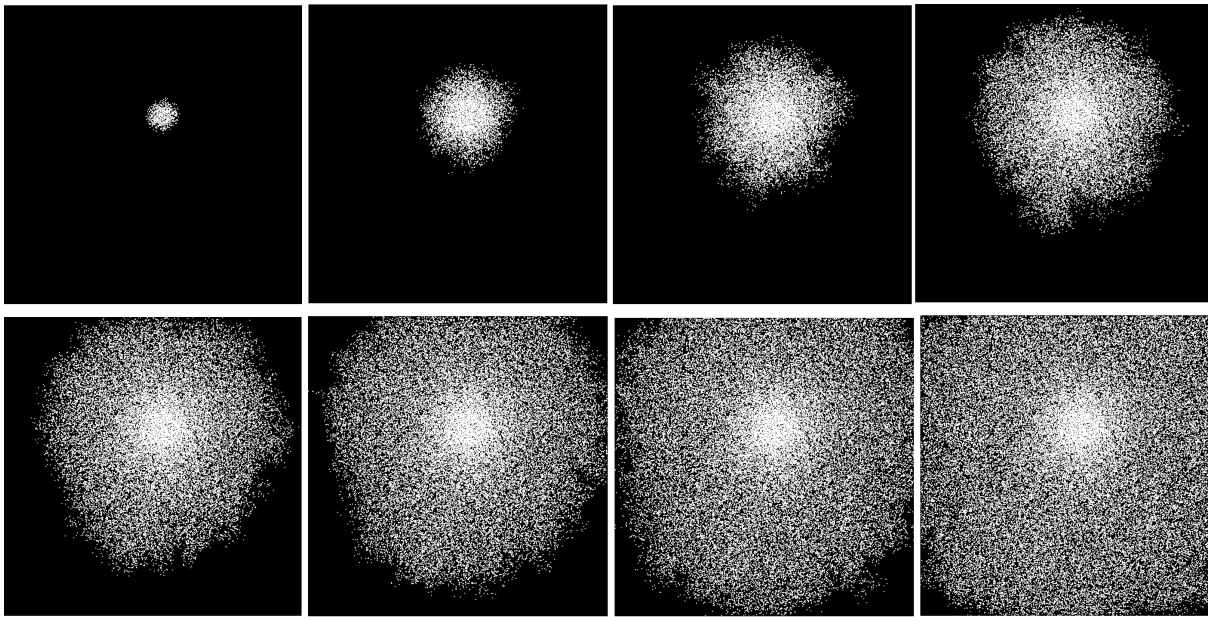

Figure 6.14: Propagation Test: Evolution through time of a simulation starting from a single seed plant of grass. From left-to-right, top-to-bottom: 2, 10, 20, 30, 40, 50, 60, 70 years in.

6.3.7.4 Varying Resource Test

To ensure a given plant specie strives better when it's habitat is more suited, multiple simulations are run with only specie S_{base} (see appendix B for specie properties) present. As outlined in table 6.2, the simulations vary only in their configured humidity. As seen by the results plotted in figure 6.15, illustrating the average canopy width for a single plant instance throughout the simulations, plants strive better in environments better suited to their resource requirements.

6.3.7.5 Shade Test

Plant's that are heavily dependent on illumination struggle to grow in areas shaded by the canopy of larger plants. To test this is modelled in the ecosystem simulator, a simulation is run with two species: $S_{smallroots}$ and grass (see appendix B for specie details). $S_{smallroots}$ is a custom specie created for the purpose of this test which has very small roots growth. This is important so as to focus on the effects of illumination and minimize the influence of drought. Figure 6.16, which illustrates the state of the simulation after fifty years, shows the grass struggling to grow in areas directly below the canopies of $S_{smallroots}$.

6.3.7.6 Shade-loving Test

Species which strive in shaded areas and struggle to survive in open spaces directly illuminated by the sun are deemed to be 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

Simulation	Simulation time (years)	Humidity	Illumination	Temperature
1	100	22	10	20
2	100	24	10	20
3	100	26	10	20
4	100	28	10	20
5	100	30	10	20
6	100	32	10	20
7	100	34	10	20
8	100	36	10	20
9	100	38	10	20

Table 6.2: Varying resource rest simulation configurations. For simplicity, monthly resources are kept constant.

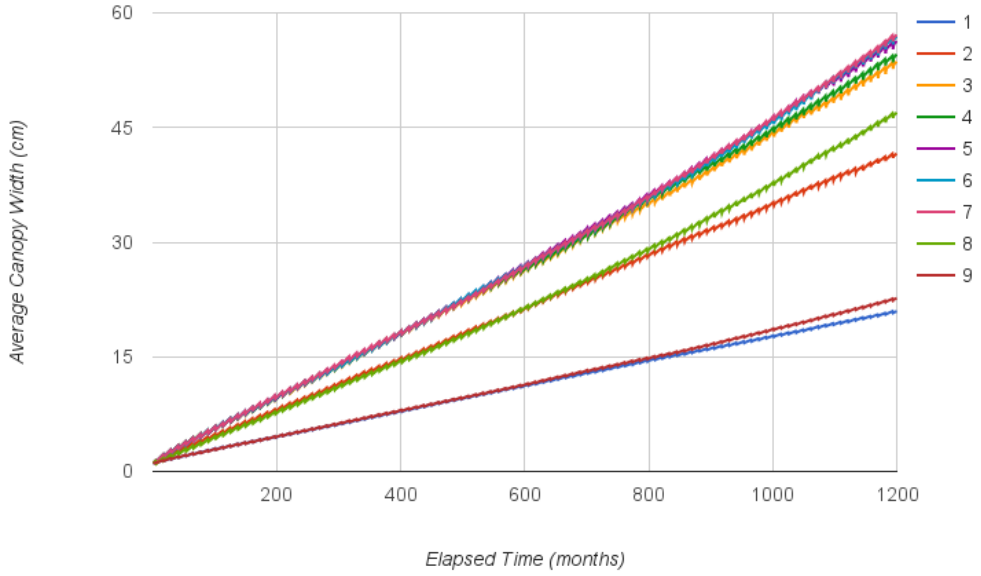

Figure 6.15: Varying Resource Test: Average canopy width of a single plant throughout the simulations outlined in table 6.2.

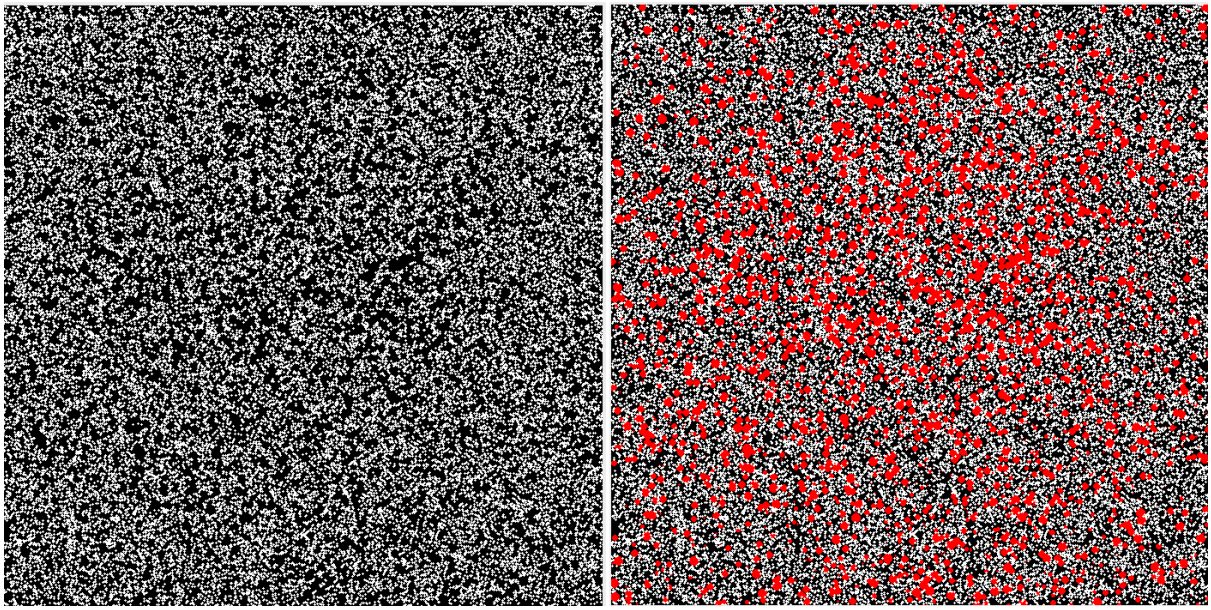

Figure 6.16: Shade Test: Simulation with $S_{smallroots}$ (red), grass (white) after fifty years. Left without rendering instances of $S_{smallroots}$ to visualise the effects of the shade

successfully caters for such plant species, a simulation is run identical to that done in the shade test (section 6.3.7.5) but with shade-loving specie $S_{shadeloving}$ added (see appendix B for specie details). As seen by the snapshot of the simulation after fifty years illustrated in figure .., instances of $S_{shadeloving}$ only appear in areas directly covered by the canopies of $S_{smallroots}$.

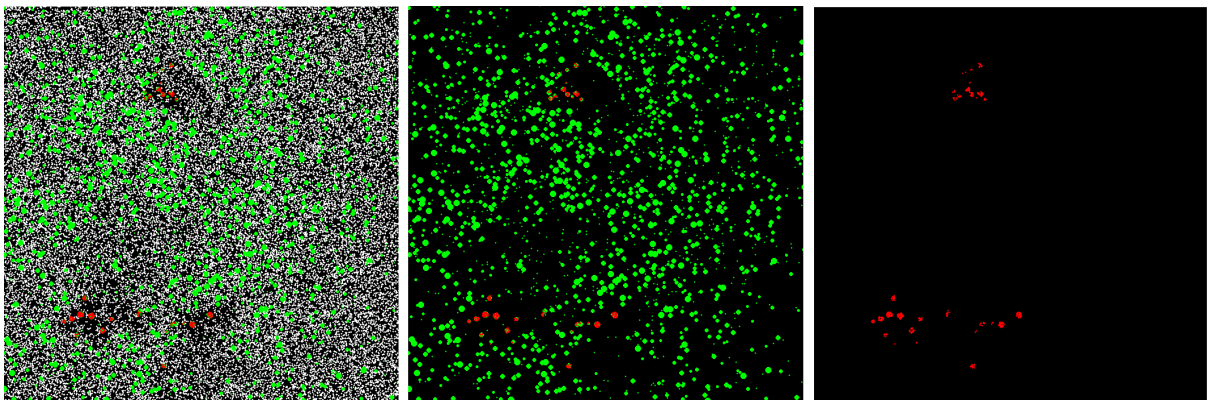

Figure 6.17: Shade Loving Test: Simulation with $S_{smallroots}$ (green), grass (white) and $S_{shadeloving}$ after fifty years. From left-to-right: All species, excluding grass and only $S_{shadeloving}$. As can be seen, $S_{shadeloving}$ strive under the canopies of $S_{smallroots}$.

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 6.3.6). This processing time depends on the plant count and the resolution of the simulation window. The simulation illustrated in figure 6.9, for example, took four and a half minutes to run.

The area to be covered by vegetation on the terrain, and therefore for which valid plant distributions created, is much larger than the hundred by hundred metre simulation window used in the ecosystem simulator. There are three obvious ways the ecosystem simulator could be used to generate vegetation for larger areas: *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, however, as *increasing the simulation* window will further increase the processing time, *decreasing the resolution* would impact the resulting realism and *tiling* would create repetitive vegetation.

To attempt to generate vegetation which is coherent with that produced with the ecosystem but on a larger scale whilst minimizing impacts on performance and realism, the plant distribution output by the ecosystem simulator is analysed for later reproduction. The analysis and reproduction process are discussed in *Analysis* and *Reproduction* respectively.

The analysed distribution data, as well as permitting larger scale reproduction, can also be used as a cache mechanism to bypass future runs of the same ecosystem simulator run. Details on how this is implemented are discussed in *Caching Distribution Data*.

To conclude this section, input exemplars will be analysed and the resulting reproduction(s) evaluated.

6.4.1 Distribution Analysis

Radial distribution analysis, as described in section 2.2.2.1, is performed on the output of the ecosystem simulator to grasp its core characteristics. Each plant instance acts as a single point and the different species represent the individual categories when performing the analysis. Customizations to the generic analysis algorithm are performed, however, to better suit the purpose of analysing plant distributions. Each are discussed separately below in: *Generating the category hierarchy*, *Category Dependency Analysis* and *Point-size Analysis*. In *Configuration Parameters* are discussed the parameters used. To conclude, the performance of the distribution will be discussed in *Performance*.

6.4.1.1 Generating the category hierarchy

During the reproduction phase, distributions for each category are created sequentially. One 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. A side effect of this hierarchical approach is that pair-correlation histograms do not need to be generated for each category pair combinations but only for combinations for which the target category is under or equal to the source category in the hierarchy.

Because taller plants will potentially have a canopy which shades and influences the position of smaller plants, it is important these be generated first during reproduction. For this reason, the hierarchy is generated according to the average height of the represented plant specie in descending order.

6.4.1.2 Category Dependency Analysis

Shade-loving plants will appear under the the shaded canopies of taller plants (see section 6.3.7.6). When analysing the distance of these shade-loving plants to the plant's which shade them during the analysis phase, a new *negative-bin* is created.

During reproduction, the taller plants will be generated first because they are classed higher in the hierarchy (see section 6.4.1.1). However, this does not guarantee all shade-loving plants will be placed in the shaded canopy of other plants as if a shade-loving plant is placed at a distance larger than R_{max} to any other plant instance, it is attributed a strength of one and is therefore deemed valid. It is essential to attribute a strength of one in such condition to permit plant propagation. A solution to this problem would be to make R_{max} large enough to cover the entire simulation window. This is extremely wasteful in terms of computational resources however and, as such, another solution is used here; When the pairwise histograms have been generated for a given category A, they are analysed sequentially to check whether or not all instances appear within the *negative-bin* of the other category (specie). The specie 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 to be deemed valid.

6.4.1.3 Point-size Analysis

As well as the position of individual plants, an important property of the ecosystem simulator output is plant size. 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 also analysed.

6.4.1.4 Configuration Parameters

The *radial distribution analysis* requires the following configuration parameters: R_{min}, R_{min} and *bin-size*. Details on each parameter can be found in section 2.2.2.1.

Increasing the analysis range [R_{min}, R_{max}] and decreasing the *bin-size* will impact performance but potentially increase the accuracy of the analysis and finding optimal values for these parameters depends on the properties of the points being analysed.
It is unnecessary to have too large of an analysis range as the impact a plant has on it's surrounding is finite. This impact radius varies however and is dependent on specie 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** passed the extremity of the plant's canopy.
The bin-size doesn't influence performance as severely as the analysis range, however, as it has no impact on the number of points that need to be processed. Smaller bin sizes will result in less 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 bin size of **twenty centimetres** is used as it strikes a good balance between accuracy and point count per bin.

6.4.1.5 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 which are subsequently analysed and the time to do so, measured. Because the number of histograms that need to be generated depends on the number of categories to be analysed, one would easily assume that the processing time is also correlated to the category count. This is **not** the case however and only point density influences analysis performance. To demonstrate this, the various plant densities are generated containing one, two and three distinct categories.The results plotted in figure 6.18 indicate an exponential correlation between plant count and processing time and a quicker analysis time when points are split into multiple categories. Although the correlation is exponential, the most extreme scenario (over ninety thousand points of a single category) is processed in a manageable time of just under two seconds.

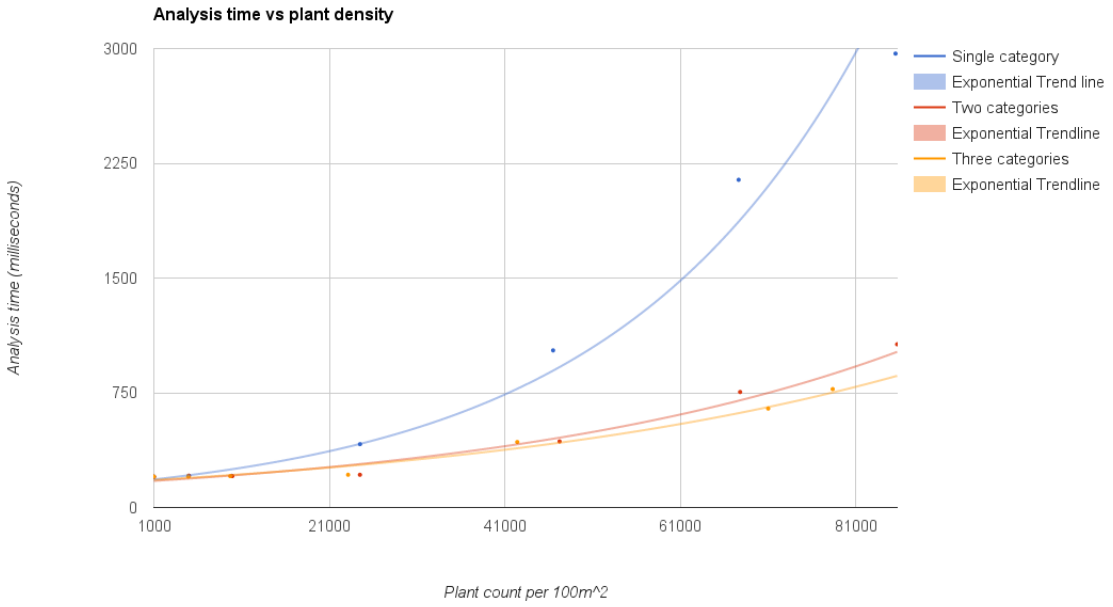

Figure 6.18: Distribution analysis time based on aggregate plant density for single category (blue), two categories (red) and three categories (yellow).

6.4.2 Reproduction

The purpose of the analysed radial distribution data is to later use it to reproduce similar distributions on larger scales. To do so, the same reproduction technique described in section 2.2.2.1 is employed with slight nuances described below, namely: *Matched-density initialization* and *Iterative point moving*.

The radial distribution data generated to test the analysis performance above (section 6.4.2.3) is used to test the reproduction performance.

6.4.2.1 Matched Density Initialization

Rather than employ a birth-and-death technique like that described in section 2.2.2.1, 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 involved generating a fixed number of points for each category so that their density matches that of the analysed density. 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, but valid.

6.4.2.2 Iterative Point Moving

When points of a given category have been initialized and the required density reached, iterative point-moving is performed where each added point is iterated over and moved to two random locations. The new distribution strength is calculated after each move and the best scoring move is accepted with probability $P_{acceptance}$, calculated using equation 6.22.

$$P_{acceptance} = \frac{Strength_{n+1}}{Strength_n} \tag{6.22}$$

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

6.4.2.3 Performance

The reproduction area and point density are two properties which greatly affect performance. Although both effect the number of points to reproduce and, therefore, the reproduction time, because an increase in density will lead to more points within R_{max} of any given source point, given a fixed plant count, performance should increase with area. 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).

As seen in figure 6.19 which plots the reproduction performance based on point density for various reproduction areas, the correlation between plant density and reproduction time is exponential. It also shows the increase to be more accentuated 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.

Using the test data generated for figure 6.19, it is possible to plot the reproduction time based solely on plant count for various densities (see figure 6.20). It shows the correlation between plant count and reproduction time to be linear and dependent on point density. The 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 ot be taken into consideration when calculating the strength of the distribution. In order to keep reproduction times manageable, the 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 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.

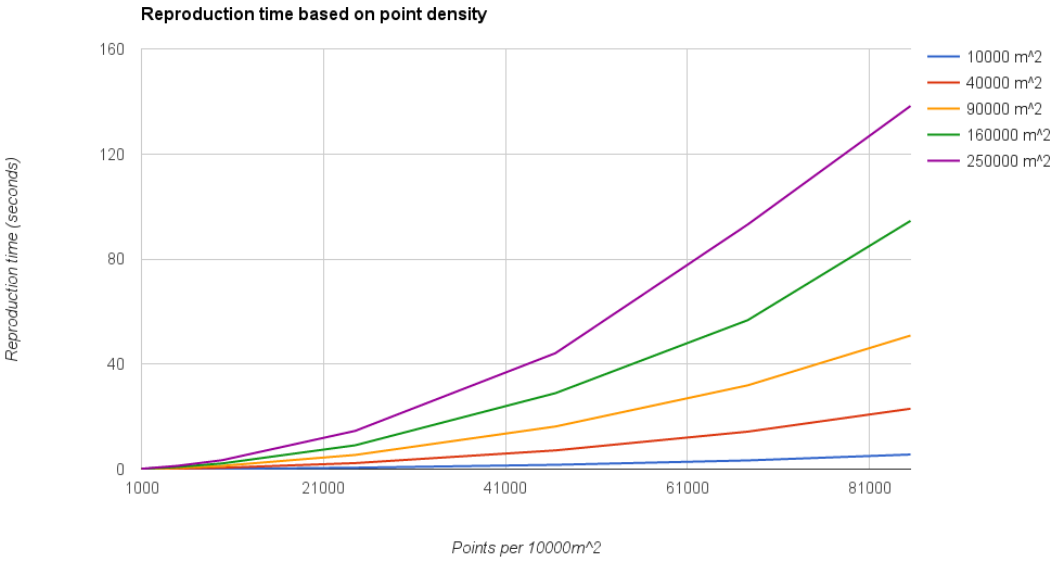

Figure 6.19: Reproduction time based on point density for different reproduction areas.

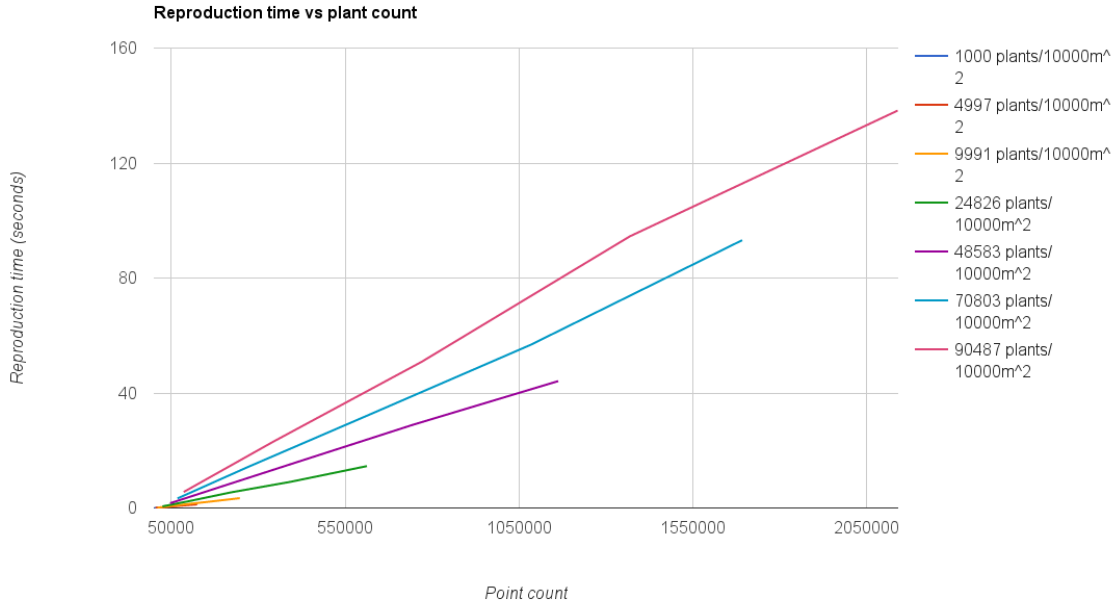

Figure 6.20: Reproduction time based on point count for different densities.

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 analysed distribution data.

6.4.4 Results

The important properties of the input exemplars which must be reproduced are: *inter and intra specie separation*, *plant size* and *specie 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 6.21 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-specie 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 6.22). 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) drastically decreases within the radius of canopy plants (category 9). Note that in both the original and reproduction, all shade-loving plants appear within the radius of a canopy plants as the categories are dependent (see section 6.4.1.2). The variation in histogram values between the two is caused by the positioning of other nearby canopy 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 specie 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. During iterative point-moving (see

section 6.4.2.2, 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.

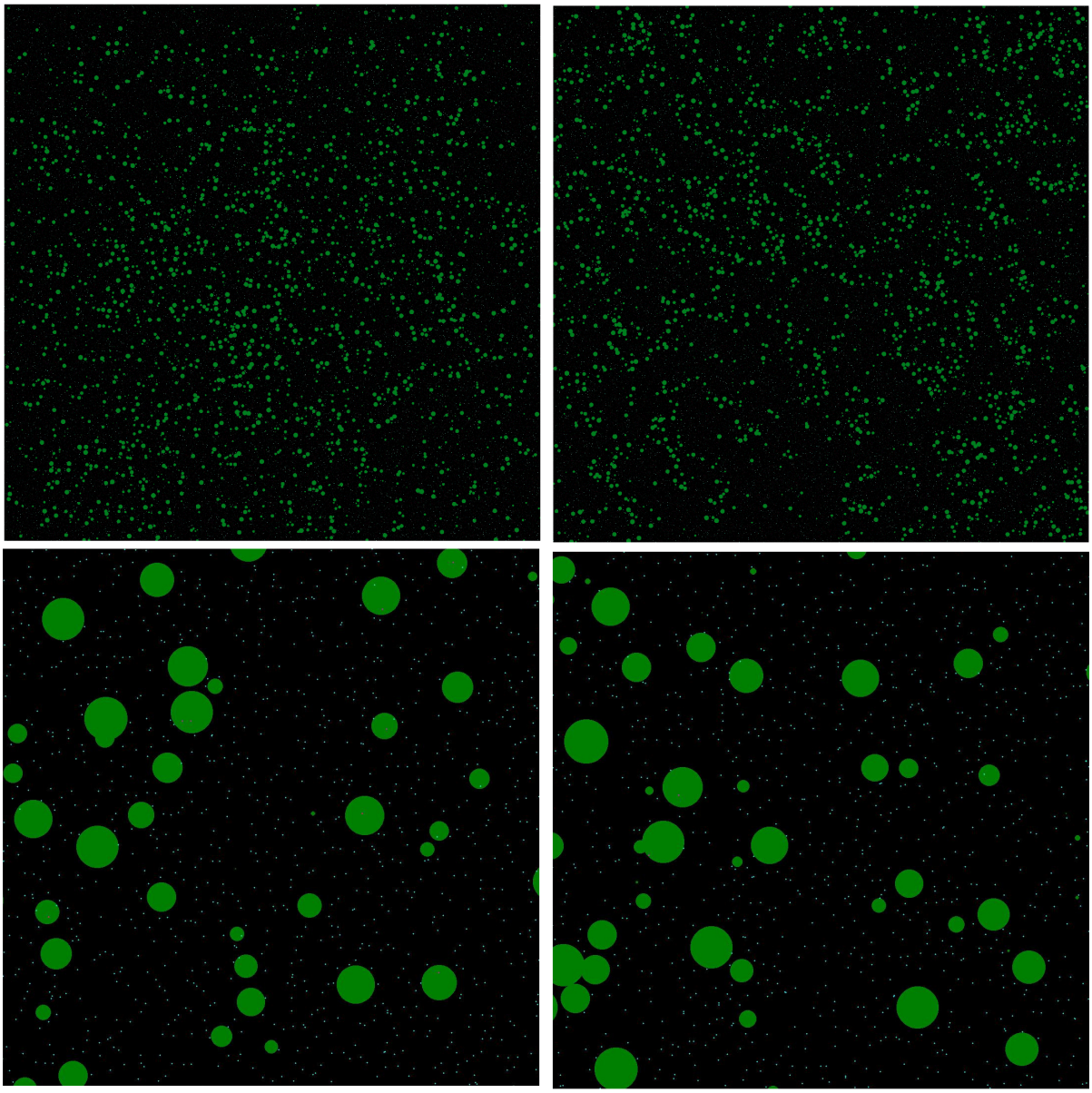

Figure 6.21: Distribution analysis and reproduction test: Input exemplar overview (top-left), reproduction overview (top right), zoomed input exemplar (x 10), zoomed reproduction (x 10).

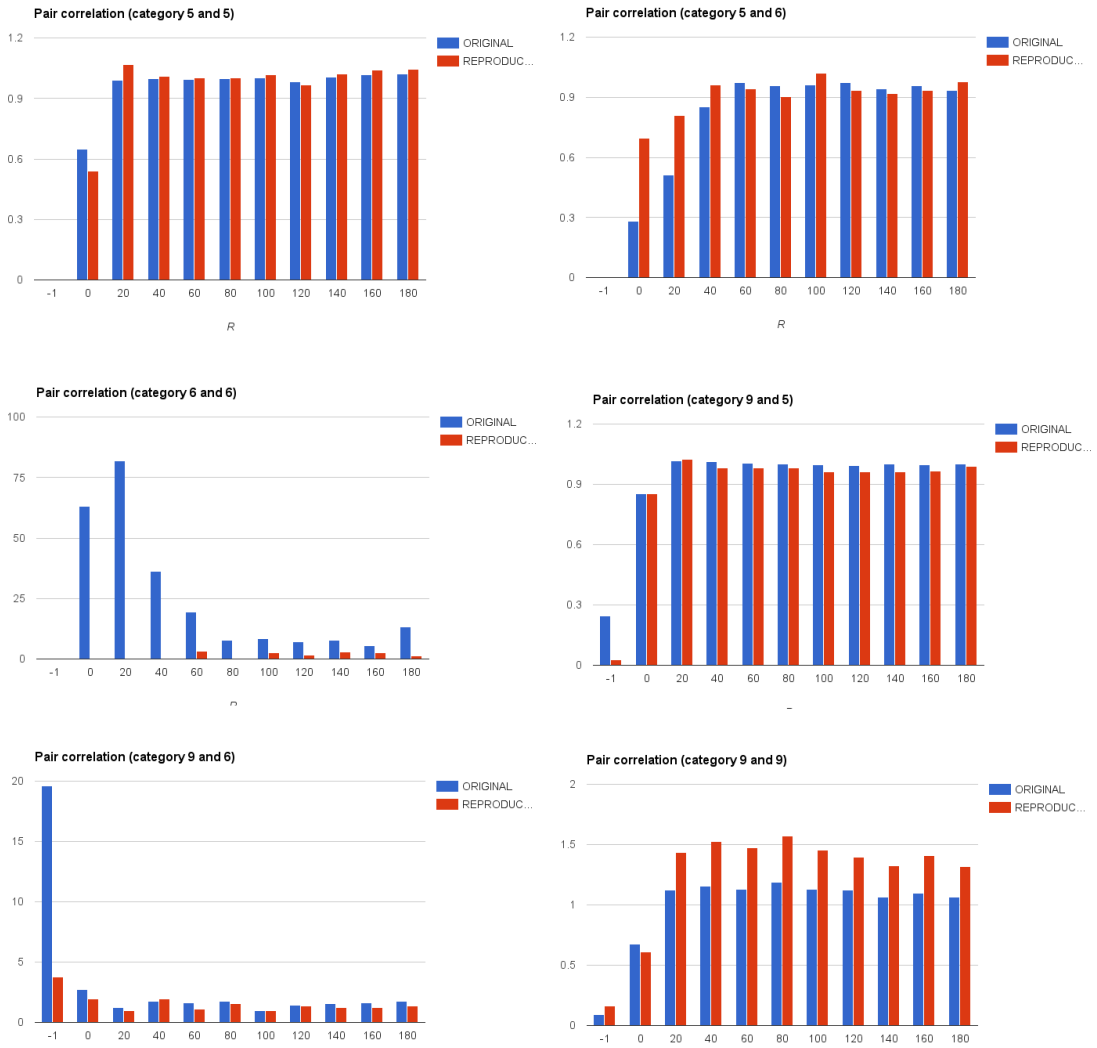

Figure 6.22: Original (blue) and reproduced (red) pair correlation histograms for different bins where category 6 is a shade-loving, category 5 is shade intolerant and category 9 is a canopy specie. Bin sizes of -1 signify the target category is within the radius of the source.

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

Specie Properties

Name	Grass	S_{base}	S_{X2}	S_{X3}	S_{slow}	S_{fast}	$S_{smallroots}$	$S_{shadeloving}$
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 dis-	3	50	50	50	50	50	50	3

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

Table D.1: Specifications of the machine used for all performance testing.

Bibliography

[BB98] Paul S Bradley and P S Bradley. Refining Initial Points for K-Means Clustering. *Microsoft Research*, pages 91–99, 1998.

[Bli] James F. Blinn. Models of light reflection for computer synthesized pictures. pages 192–198.

[DHL+98] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomir Měch, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. *Conference on Computer Graphics and Interactive Techniques*, pages 275—-286, 1998.

[DSS93] Rudy P Darken, John L Sibert, and Computer Science. A Toolset for Navigation in Virtual Environments. pages 157–165, 1993.

[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.

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

[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.

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

[PC00] MP Perrone and SD Connell. K-means clustering for hidden Markov models. *Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition*, (NOVEMBER 2000):229–238, 2000.

[ŠBBK08] Ondej Št'Ava, Bedich Beneš, Matthew Brisbin, and Jaroslav Kivánek. Interactive terrain modeling using hydraulic erosion. *EG CA - EuroGraphics Symposium on Computer Animation*, 2008.