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 [aEL99, LLX+01, WLKT09], modelling plants [BPC+12, FZS+08, GFJ+11, Lew99], generating terrains [SKG+09, GMSe09, DP10], generating river networks [DGGK11, EC15] and generating city landscapes [GMN14, KM07, PM01] (figure 1.1)
A common difficulty with these methods, however, is finding the appropriate input parameters for the procedural algorithms. The correlation between the parameters and the resulting content is often unintuitive and, as a consequence, often 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 [EC15] to sketch-based recognition algorithms [GMSe09].

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.

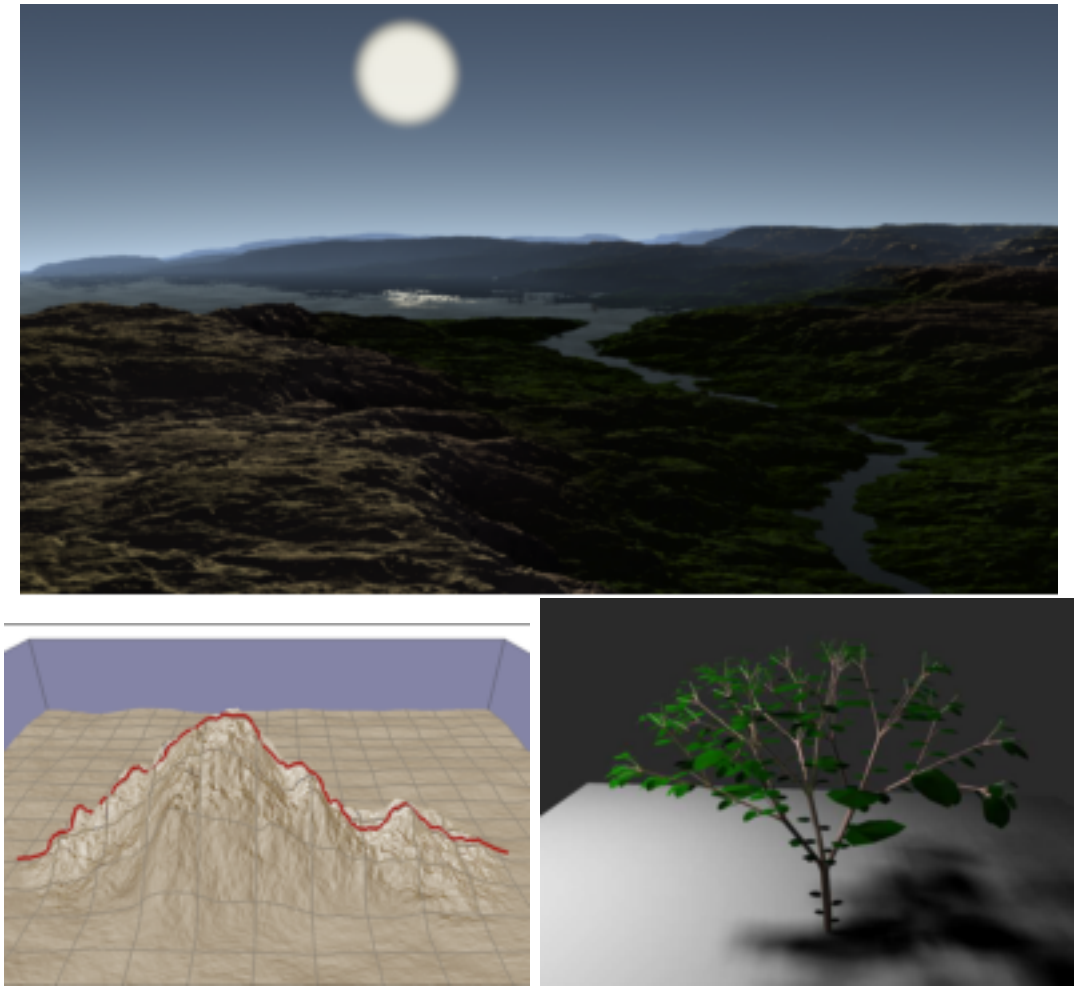

Figure 1.1: Example of procedurally generated content. From top to bottom, left to right: Procedurally generated river stream [DGGK11], procedurally generated terrain through sketching [GMSe09], procedurally generated plant [SSBR01]

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

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

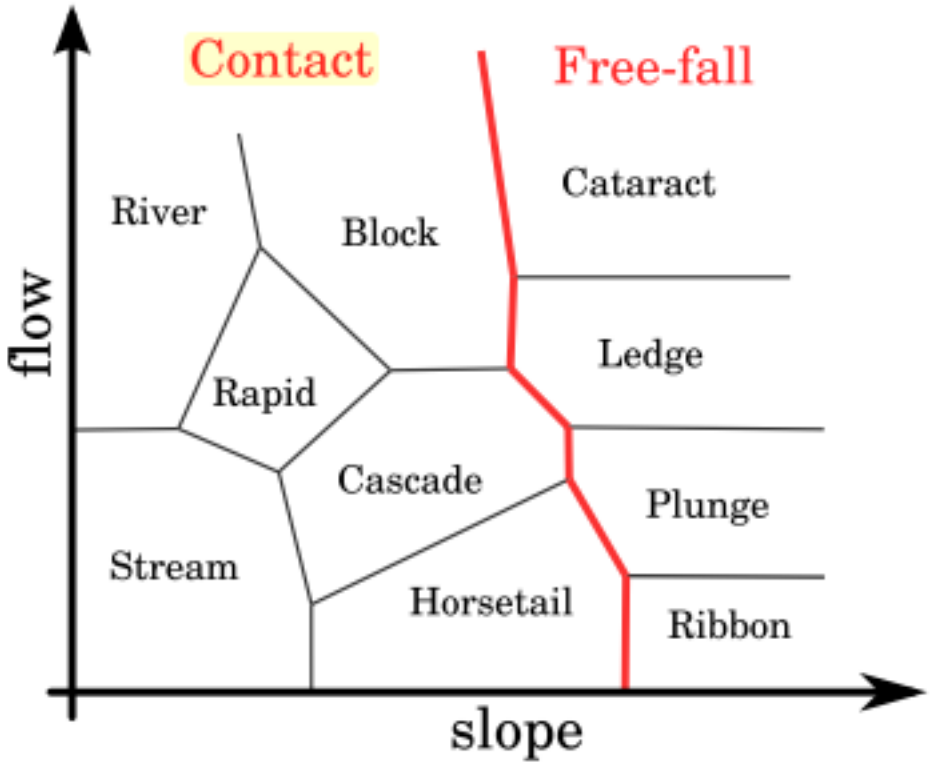

Figure 2.1: Waterfall classifications [Emi14]

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 [BA05] simulate the effect gravity has on water particles placed on the peaks of pre-generated ridges. To create the ridges, particle pairs are first placed at random locations on the terrain. These particle pairs are then randomly assigned a horizontal axis from which they iteratively distance themselves in opposite directions. At each iteration a new vertex is placed and its height decreased from the previous vertex based on a Gaussian distribution. To create the river networks, river particles are placed on the top of these generated ridges and a physical simulation which takes into consideration particle velocity, particle mass and surface friction is used to model the motion of these particles on the terrain. The path followed

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

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

In their work on modelling the effects of hydraulic erosion, t'Ava et al. [vBBK08] 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. [vBBK08] 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. [vBBK08], 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. [KMN88] and t'Ava et al. [vBBK08] attempt to produce plausible terrains by modelling these effects.

In the work by Kelley et al. [KMN88], the user specifies, on a horizontal plane, the terrain outline along with the main trunk stream. The terrain outline is used to configure the terrain extremities once ported to a three-dimensional space. The main trunk stream specifies the path which the highest order water stream should follow on the resulting terrain. Given this terrain outline and the position of the initial main trunk stream, the system iteratively increments the number of nodes which form the main trunk in order to add streams to the network. The number of new nodes added depends on the drainage area (surface area that a stream needs to channel) and the soil type as more resistant soil materials (e.g. stone) will be less influenced by water erosion than weaker ones (e.g. clay).

t'Ava et al. [vBBK08] 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).

Figure 2.2: Simulation of dissolution-based erosion erosion caused by water movement[vBBK08]

Whether modelling erosion indirectly like in the work by Kelley et al. [KMN88] which builds the terrain around models of erosion or directly like the work by t'Ava et al. [vBBK08] which simulates the effect of erosion in real-time, both succeed in producing plausible terrains with integrated river networks. Fine-control over the resulting terrain, however, is limited in the work by Kelley et al. [KMN88] due to extensive automation. This is overcome in the work by [vBBK08] 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. [vBBK08] are able to reproduce the effects of erosion in real-time.

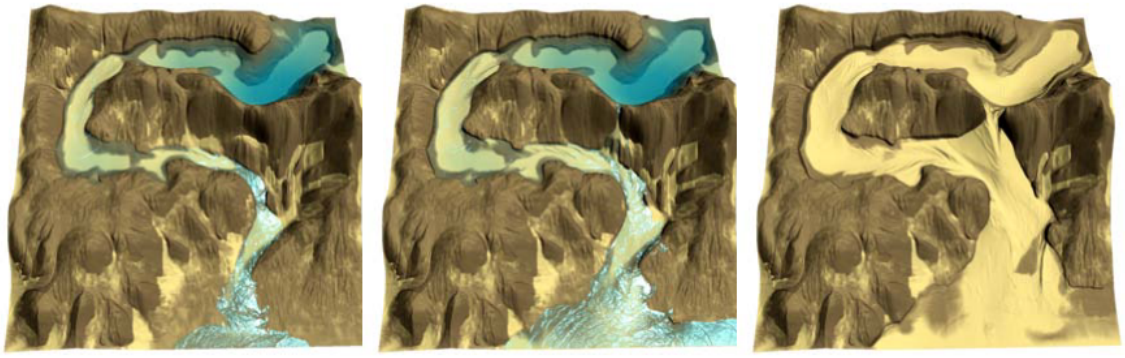

Figure 2.3: Simulation of the effect of force-based erosion caused by running water [vBBK08]

2.1.2.3 Rainfall simulations

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

Derzapf et al. [DGGK11] use such methods in their work based on procedurally generating virtual planets in real-time. To do so, only a very basic mesh-representation of the terrain is generated at first and detailed content is generated on demand as the user navigates through the virtual world. This method of adaptive rendering permits memory usage to me manageable whilst not compromising on realism. To ensure updates are performed in real-time, their algorithms are designed to make use of the massively

parallel architecture of GPUs.
To initialise the base representation of the planets, the system first creates the base mesh with all vertices representing the sea. The system then randomly assigns a certain number of these vertices to act as seed continent vertices to spread until a user-configured land-to-water ratio is reached. To place rivers, similarly to the work by [GGG+13] et al., they first locate continental points which are on coastal edges to act as river mouths. When such a vertices are found, adjacent continental vertices are iteratively selected pseudo-randomly and connected in order to form the river network.
To assign ground altitudes to connected river vertices the system employs the following formula, starting from the river mouth:

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

Where:

- a_v is the ground altitude of the current vertex.
- a_u is the ground altitude of the previously processed vertex (or zero if v is the first vertex).
- e_a is the average ground elevation.
- l_e is the length of the current vertex.
- $\xi \in [0, 1[$ is a uniformly distributed pseudo-random number.
- $a_{maxriver}$ is the user-configured maximum river altitude.
- l_r is the current river length.

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

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

Where:

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

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

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

2.1.4 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.

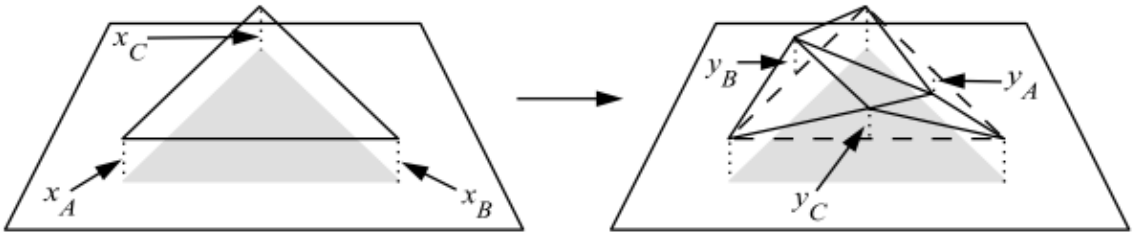

Figure 2.4: A single iteration of midpoint displacement for the creation of mountains [PHM93]. New vertices y_A, y_B and y_C are created and shifted vertically by a random offset

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

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

- An entry edge must split into an entry and a neutral edge.
- An exit edge must split into an exit edge and a neutral edge.

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

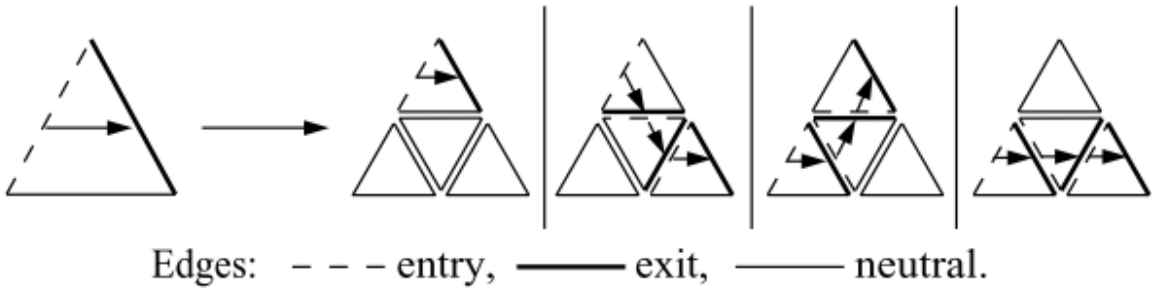

Figure 2.5: Single production of midpoint displacement adapted to river generation [PHM93]. Given the initial triangle, four valid split scenarios.

One difficulty with this technique is to ensure two adjacent triangles are coherent once split. That is, that the exit edge of one coincides with the entry edge of the other. To solve this, the location of edge vertices are used as the key to a random number generating hash table which, based on its output number, determines the segment that will be crossed by the river, if any.

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

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

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

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

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

Smelik et al. also use explicit techniques to create an interactive system permitting users to model a complete virtual world with content ranging from rural features (mountains, rivers, etc.) to man-made ones (buildings, road networks). When modelling the virtual world, interactions are split into two modes: **Landscape** and **Feature**. *Landscape mode* permits the designer to paint ecotopes onto the terrain using traditional image editing tools. These ecotopes are predefined by the user and encompass both elevation and soil material information. In *feature mode*, the user is able to place terrain content, including rivers. To do so, similarly to the interface provided by Emilien et al. [Emi14], the user sketches vector lines outlining the core path of the river and, based on this, a suitable course is plotted through the landscape. Other terrain features to which the river takes precedence adapt accordingly. For example, if the river is plotted to pass through a forest, trees on the rived bed and bank will be removed automatically.

Rather than placing vector-lines, the work by Soon Tee [Teo08] and t'Ava et al. [vBBK08] 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*

	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

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).

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. [EC15] permit users to insert individual plants manually by simply clicking a given location on the terrain. To overcome the tedious task of manually placing individual plants on large terrains, the system is able to analyse existing distributions for reproduction. For example, to generate a large forest, the user is only required to generate a small subsection which can then be used to reproduce it on any scale (figure 2.6)

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

In their work focused on improving the realism of roadside landscapes, Andujar et al. [ACV+14] use orthophotos as input to determine the location and properties of individual plants. Unlike ordinary aerial photographs, aerial orthophotos use normalisation techniques to take into account terrain relief and camera tilt. The result is an image with uniform scale throughout which, similarly to a map, can be used to accurately measure distances between points. These orthophotos are used to measure the distances between plants. To later reproduce the roadside landscape, they use a dart throwing algorithm to place individual plants whilst respecting the appropriate distances.

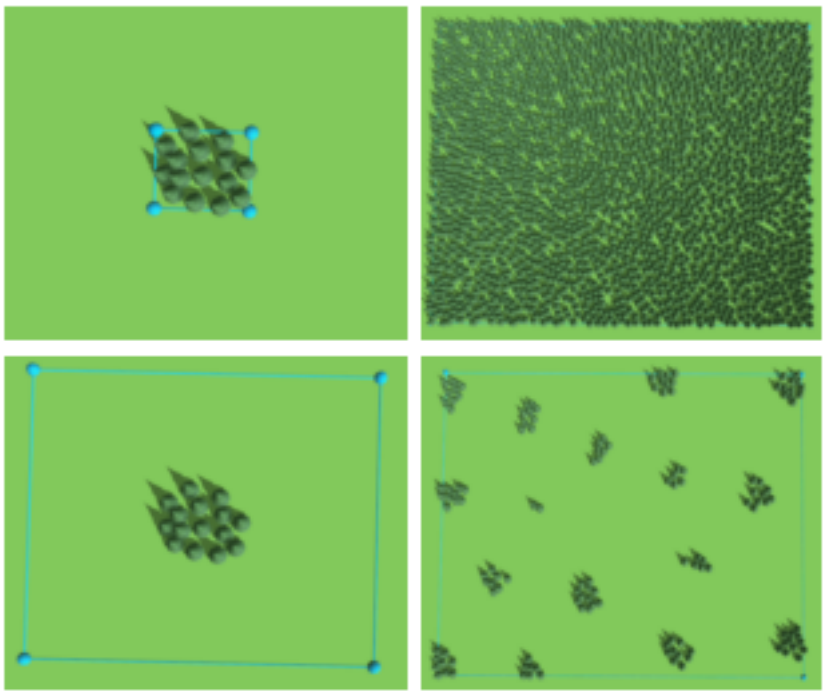

Figure 2.6: Using explicit placement as input examplars for reproduction [EC15]

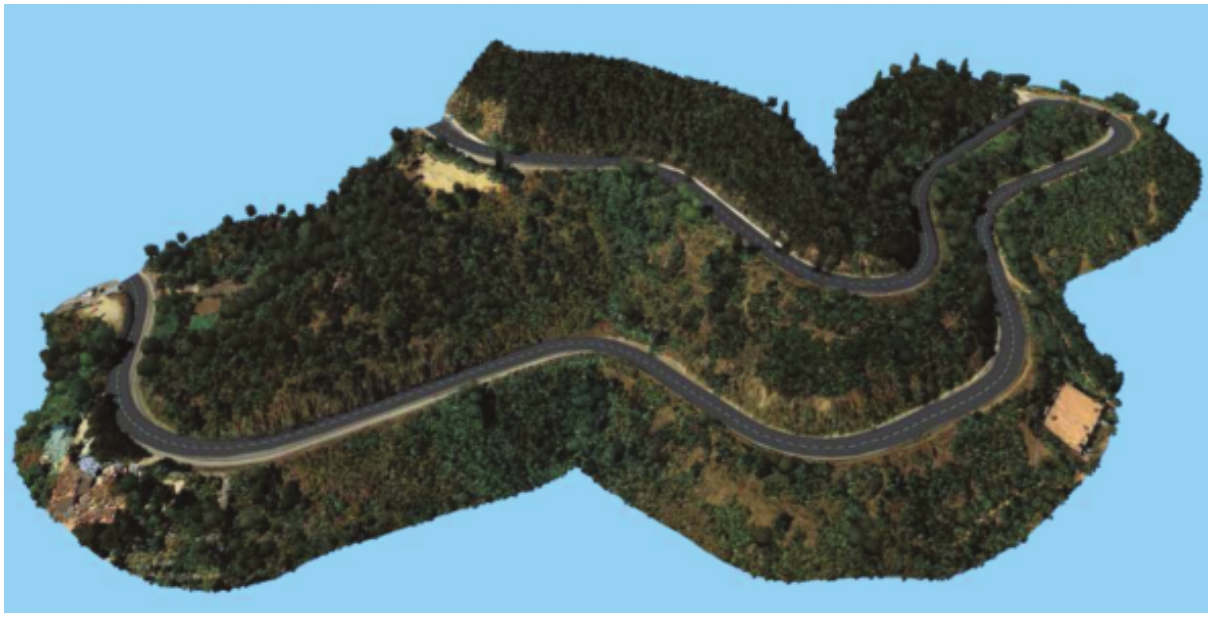

Figure 2.7: Reconstructed roadside vegetation using orthophotos [ACV+14]

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 explicitly placing individual plants from scratch.
Creating vegetation for large virtual worlds using these methods is extremely strenuous and, as a consequence, realism is often compromised.

2.2.2 Probabilistic Placement

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. [EC15], Boudon et al. [BM07] and Lane et al. [LP02] use radial distribution analysis to convert to metric form the underlying plant distributions of input examplars. The data generated by the analysis stage can later be used to synthesise, at any scale, 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)

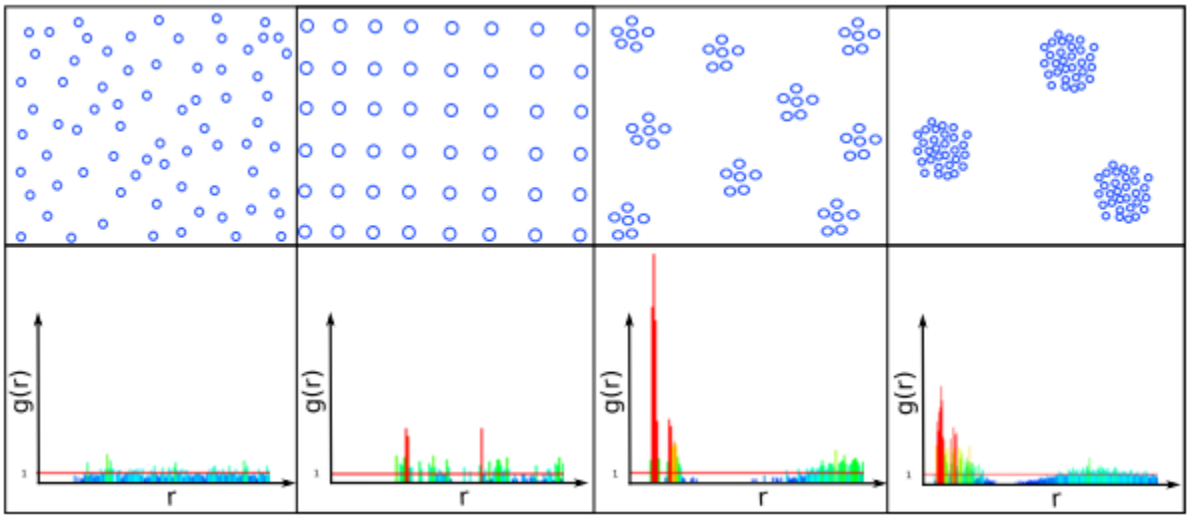

Figure 2.8: Point distributions with associated pair correlation histogram [Emi14]

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

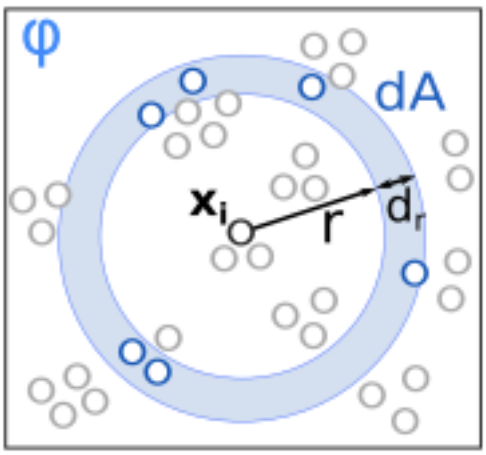

Figure 2.9: Radial distribution analysis

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 [HLT09] 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 /
- $h_{X,Y_k}(d(x_i, y_j))$ 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 [EC15] are able to perform analysis and reproduction in near real-time.

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

2.2.2.2 PREDEFINED ECOSYSTEMS

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

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

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

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

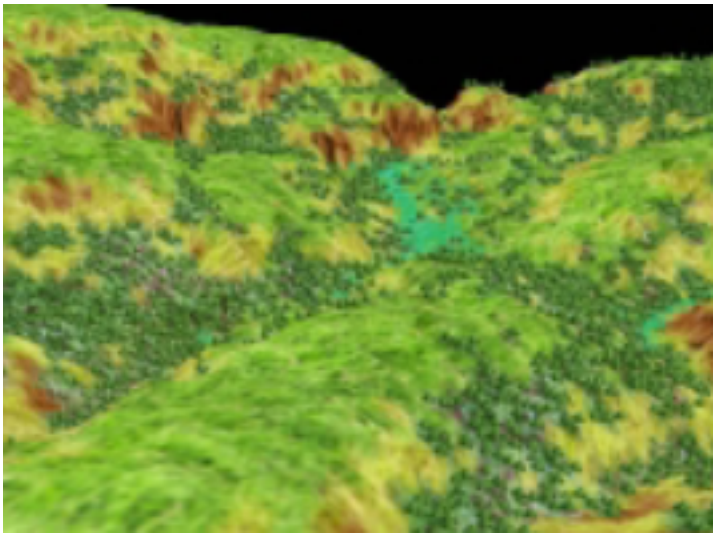

Figure 2.10: Vegetation generated using predefined ecosystems [Ham01]

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. [SSBR01, SED03] splits single plants into geometrical organs with unique light transmittance and reflectance properties. By doing so, light propagation within the plant can be simulated in order to determine the aggregated photosynthetic potential. This work, along with that of Yan et al. [Yan04], base their simulators on two vital and widely accepted laws of nature:

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

the frequency of these cycles is proportional to the sum of the daily average of the temperatures.

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

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

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

Where:

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

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

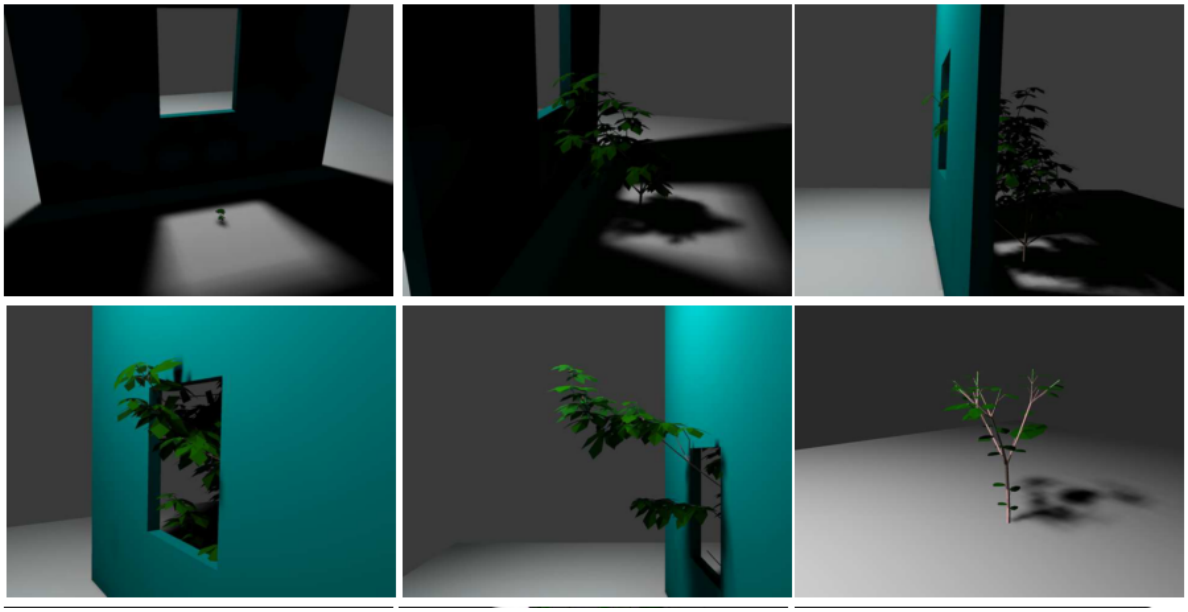

Figure 2.11: Plant growing towards light source [SSBR01]

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.
Such plant properties include: age; vigor; shade tolerance; humidity requirement and temperature requirements. Amongst others, the resources modelled include: available illumination; available humidity; temperature and slope.

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

Lindenmayer systems, commonly referred to as L-systems, use a formal grammar along with a set of production rules to iteratively create larger strings from a starting string called the axiom. Such systems are commonly used to model plants and plant growth [PL90, DC02, BPC$^+$12, PHM93].
An extension to basic L-systems, referred to as open L-systems, adds a communication grammar which permits the set of production rules to behave differently depending on predefined conditions [Pru96]. In their work modelling the growth of struce trees, Berezovskava et al. [BKK97] 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 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.

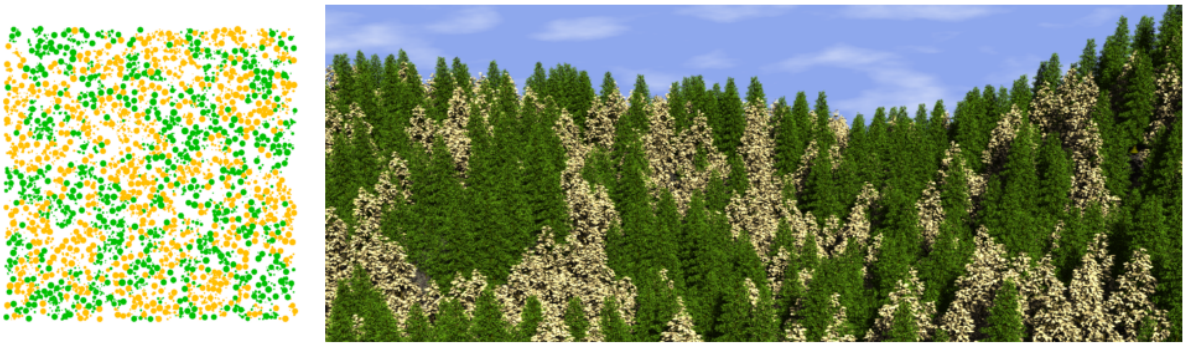

Figure 2.12: Plant placement using an ecosystem simulator modelled by L-Systems [LP02]. *Left:* Result of the simulation where orange circles indicate the positions of poplar trees and green circles the positions of spruce trees. *Right:* Reproduced virtual world where the location of individual plants 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. [SSBR01], 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

Terrain & Resources

The first step in creating virtual worlds is generating the base terrain on which features will be placed. Sequentially, terrain resources with direct influence on content need to be determined. To keep user input to a minimum, this must be done procedurally when possible.
This chapter discusses how a system which fitting this 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.
In order to determine suitable vegetation and river sources, resource information needs to be gathered. 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 & streams or water reserves. Techniques used to place such content are discussed in sections *Rivers & Streams* and *Water bodies* respectively.

3.1 Terrain & 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. Efficient 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.

3.1.1 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, represent the height of the individual terrain vertices. A Terragen height-map is a freely available and heavily used file-specification format which wraps raw height data with other important information such as base height, scale and dimensions.

3.1.2 Rendering Terrain

Once parsed, the height-map data is transferred to the GPU via a two dimensional texture for rendering. In order to better visualize the terrain relief, a BlinnPhong shading model is used when rendering the terrain. BlinnPhong is a popular and efficient shading model which uses a combination of camera position, light direction, surface normals, material and light color specifications to produce a visual appealing rendering which accentuates relief change.
Calculating the normal vectors for each point on the terrain is done using the algorithm outlined in figure 3.1. Each normal is calculated in parallel on the GPU, thus ensuring real-time results.

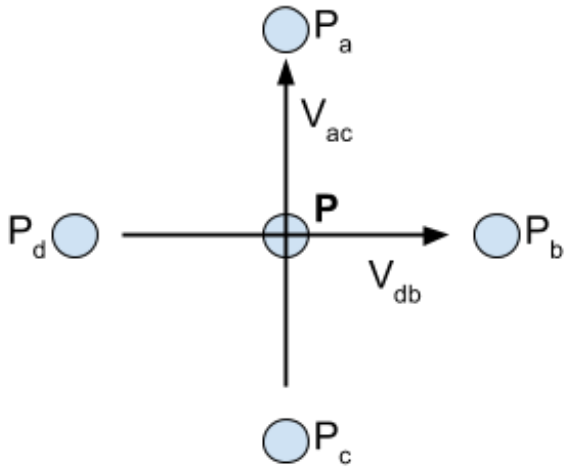

Figure 3.1: Terrain normals calculation. $N_P = V_{ac} \times V_{db}$ where N_P is the normal vector at point P and P_A, P_B, P_C and P_D are the direct points surrounding P in the X and Y direction.

3.1.3 Navigation

In order for users to successfully and intuitively navigate through virtual worlds, it is important to prevent any form of disorientation. Appropriate control sensitivity, logical points of references and predictable correlation between mouse movements and corresponding virtual world movements are all essential in preventing possible disorientation. A point of reference in virtual worlds is a point that doesn't change and around which all other points are free to move. Two common reference points are used when visualising a virtual 3-D space: the camera or scene objects.
Arc-ball is a popular technique in which the camera remains static and therefore acts as the point of reference. Arc-balls are popular in modern modelling software packages and permit users to interact with scene objects through intuitive click and drag gestures.
Navigation techniques where scene objects act as the point of reference are most common in video games where the camera moves around a static scene.
Static scene with moving camera is the most widespread navigation style and the most similar to the real-world where the earth acts as the static scene and us humans act as the moving camera. In order for a more variety of users (novice to computer graphic experts) to get up to speed with the controls efficiently, this is the navigation style used in our system.

Two different control types can be selected which are described in more detail in table 3.2: *first-person* and *click-and-drag*. The active control type is easily configurable, along with sensitivity parameters, through the applications settings interface. By providing multiple control-types and the ability to customise them the system can better cater to the requirements of a wider user-base.

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 movement	Vertical mouse movement
Click-and-drag	Horizontal click & drag	Vertical click & drag	Scroll wheel	Ctrl + horizontal click & drag	Ctrl + vertical click & drag

Table 3.1: Control types instruction sheet

3.2 Resources

Two core features of rural landscapes are vegetation and water networks. Accurately replicating these features is therefore critical to the resulting realism of the modelled virtual world.
The combination and annual variance of temperature, illumination and precipitation have a direct effect on vegetation and water networks. Therefore, to procedurally determine the vegetation and water networks, available resources must first be resolved. This section will focus on the approach taken to determine available resources throughout the terrain. Due to scope, not all resources could be modelled and this work focuses on: *Illumination*, *temperature*, *precipitation*, *soil humidity* and *slope*.

3.2.1 Illumination

Illumination and annual variation thereof greatly effects the type and density of vegetation that can grow. Whereas some plants prefer habitats with limited sunlight exposure (lilies, etc.), some strive in fully exposed environments (sunflowers, etc.).
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. How this is done and how the terrain illumination is calculated accordingly is discussed below.

3.2.1.1 Calculating the suns trajectory

The earth rotates around the sun with an axial tilt, also known as obliquity, of approximately 23.5 degrees (see figure 3.2). 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 a year (see figure 3.3). For the northern hemisphere, the day length will be at it's maximum during the June equinox and at it's minimum during the December equinox. At these moments in time, 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 orbits the sun. 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 (figure 3.4). By keeping the render window visible during this edit, modifications are clear to the user. When any of these values are changed, the position of the sun is automatically recalculated in real-time.

Orientation is displayed to the user at all times with the use of an overlay compass (figure 3.5) inspired by first-person video games. When in "orientation edit mode" the

[1] `http://en.wikipedia.org/wiki/Summer_solstice`
[2] `http://www.physicalgeography.net/fundamentals/6i.html`

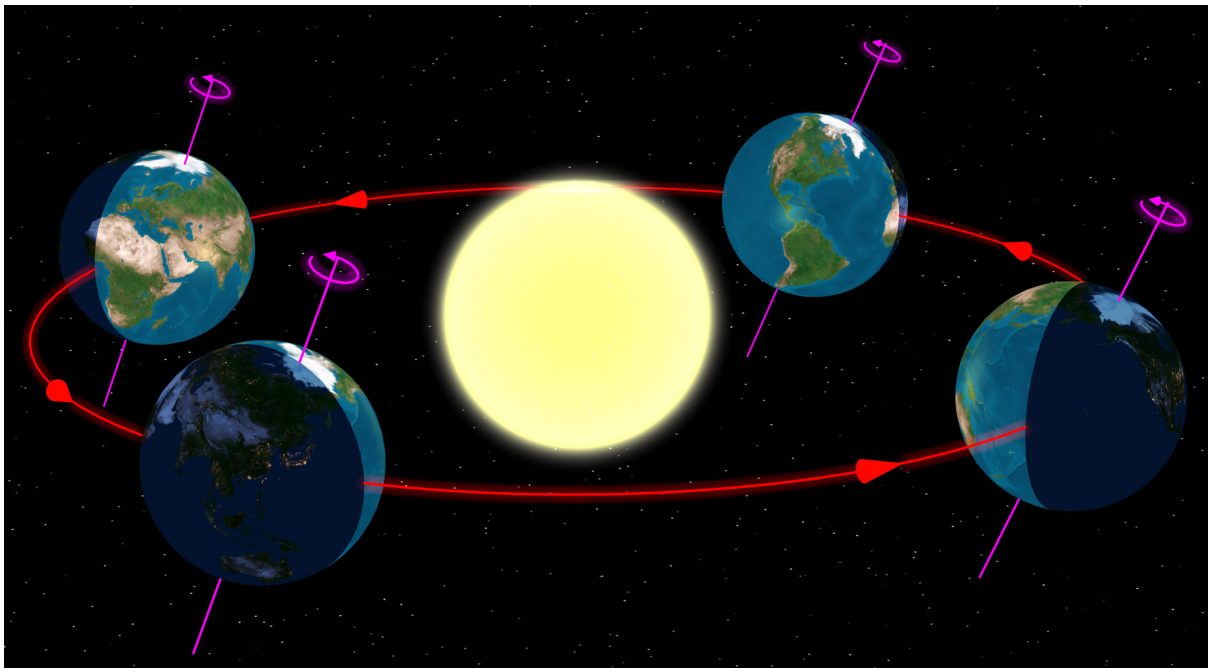

Figure 3.2: Earth orbiting the sun [1]

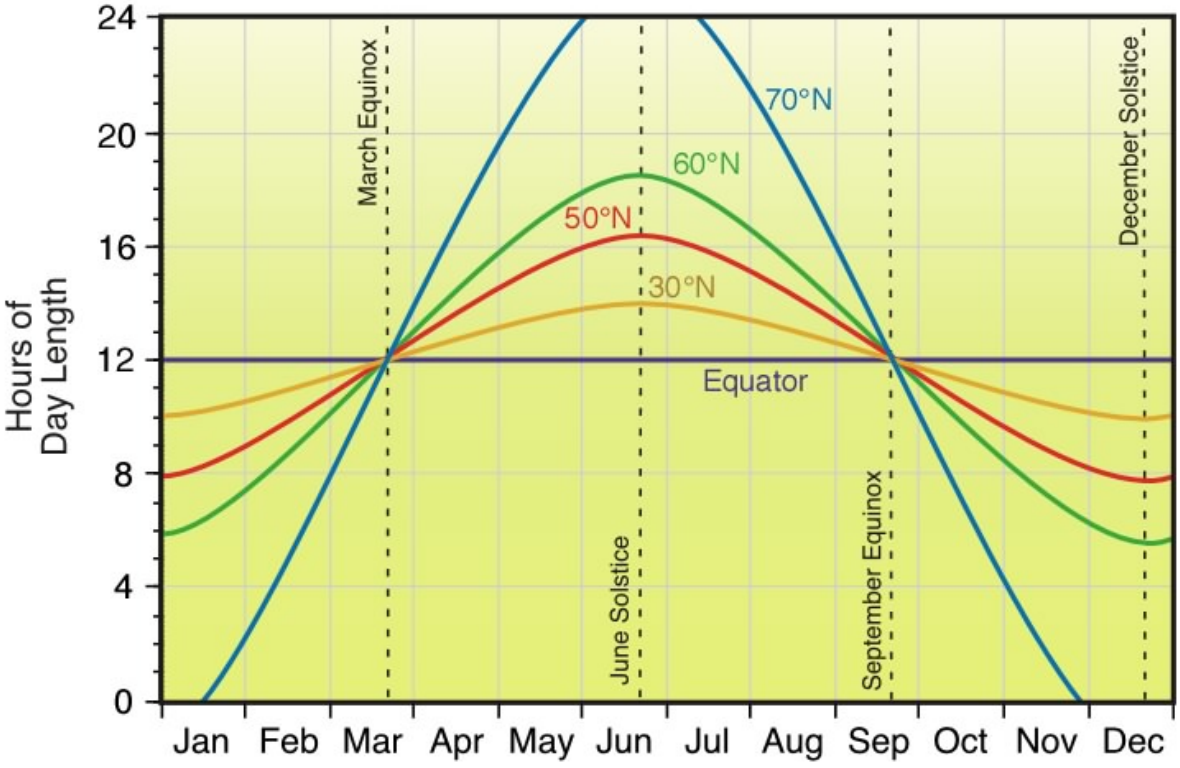

Figure 3.3: Variation in day length for different latitudes [2]

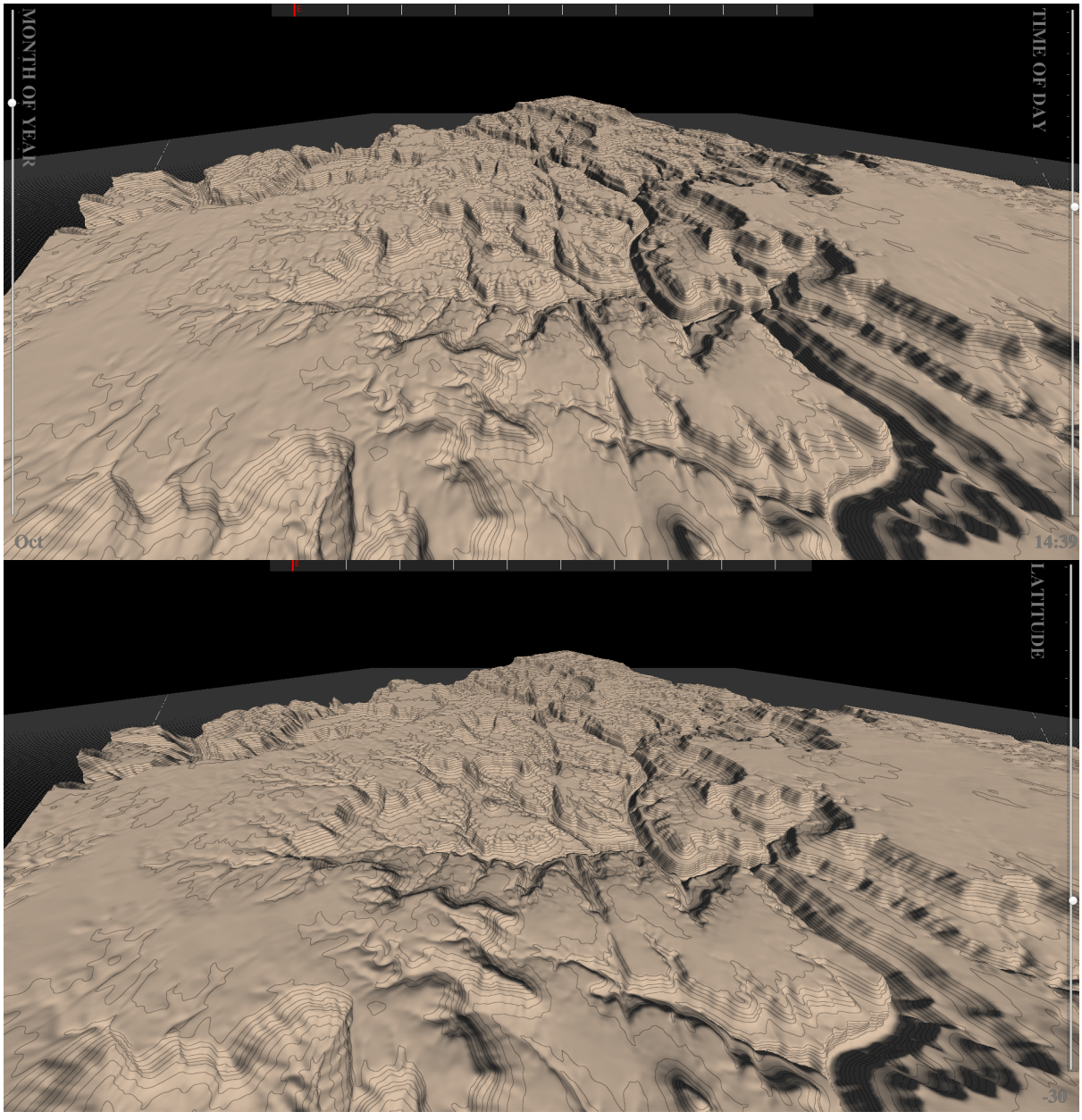

Figure 3.4: Time (top) and latitude (bottom) controllers

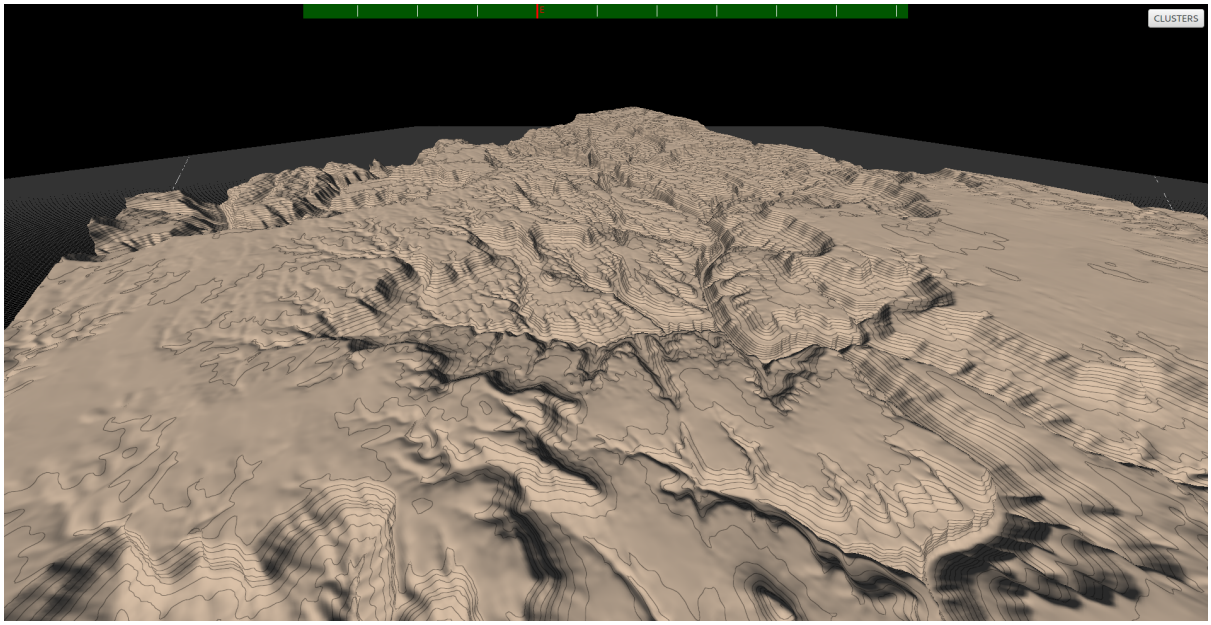

Figure 3.5: Orientation controllers

compass changes to green to inform 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 3.1. 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.

$$V_{RE} = R(V_N, \text{-}L, V_E) \tag{3.1}$$

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, only latitude is the determinant factor of daytime duration. In order to calculate the rotation axis V_R(m) of the sun at month m, axis tilt must be taken into consideration by further rotating V_{RE} using equation 3.2.

$$V_R(m) = R(V_{RE}, a_m, V_E) \tag{3.2}$$

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 3.1).
- V_E is the east-facing vector passing through the terrain center.
- a_m is the rotation angle calculated using equation 3.3 .
- $R(V_a, a, V_b)$ is the resulting vector after rotating V_a by a degrees around V_b

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.

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

$$tilt_{monthly} = tilt_{max}/3 \tag{3.3}$$

where:

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

3.2.1.2 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.
A spherical hierarchical acceleration structure is used to accelerate the ray casting process. This hierarchical acceleration structure acts as a tree structure to iteratively search for smaller intersection areas. Using this acceleration structure, illumination can be calculated for 4 million vertices (2048 by 2048 terrain) in just over 2 seconds (figure 3.6). As shown in figure 3.6, There is a linear relationship between vertex count and calculation time. (i.e vertex count).

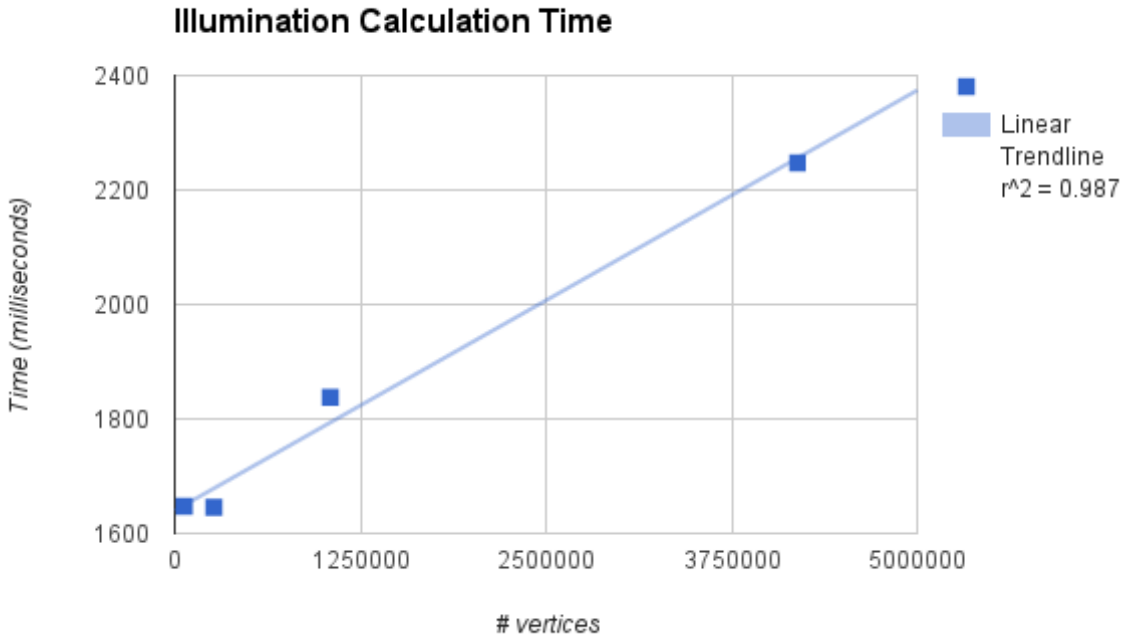

Figure 3.6: Illumination calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024 and 2048 by 2048.

3.2.1.3 Illumination overlay

To visually present terrain illumination to the user an *illumination overlay* can be selected which renders an overlay on-top of the terrain (see figure 3.7)

3.2.1.4 Daily Illumination

An important factor for plant growth is the hours of illumination received throughout the day. The illumination calculation is used (3.2.1.3) to determine this by going through each hour of the day consecutively and determining whether or not a vertex V is illuminated.

3.2.1.5 Daily Illumination overlay

Similarly to the illumination, a visual presentation of the daily illumination can be enabled as observed in figure 3.8.

3.2.2 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 variation thereof has a big impact on vegetation. For modelling purposes, it is acceptable to assume that the

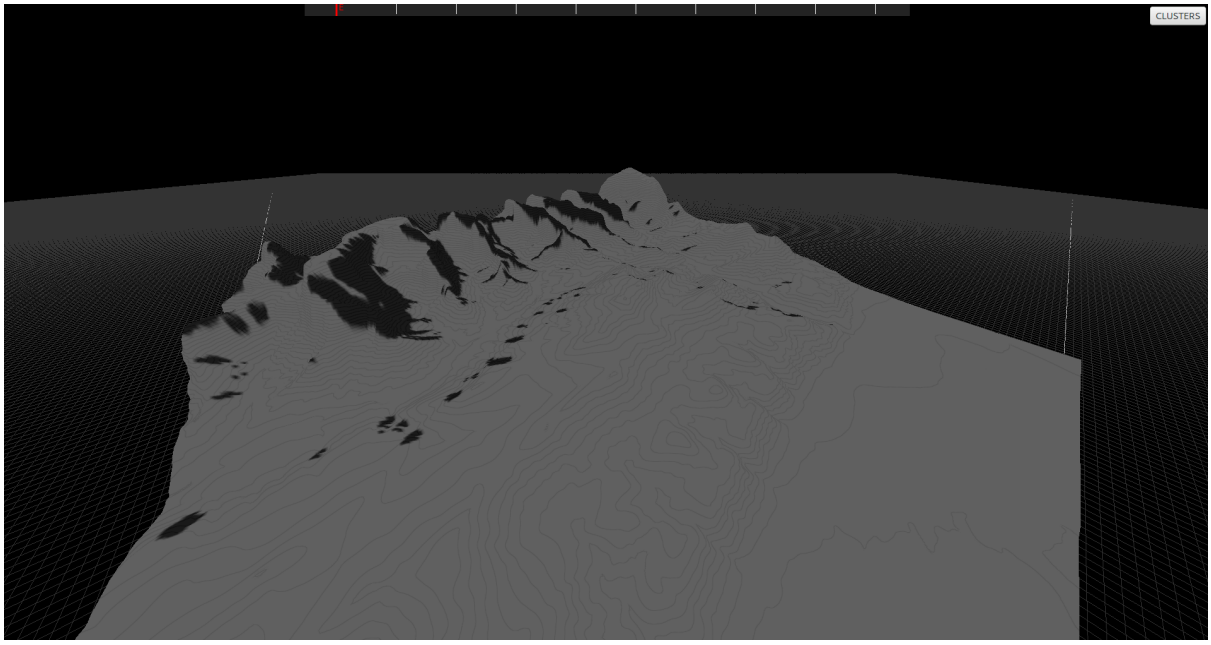

Figure 3.7: Illumination overlay. Illuminated areas are coloured white. Shaded areas are coloured black.

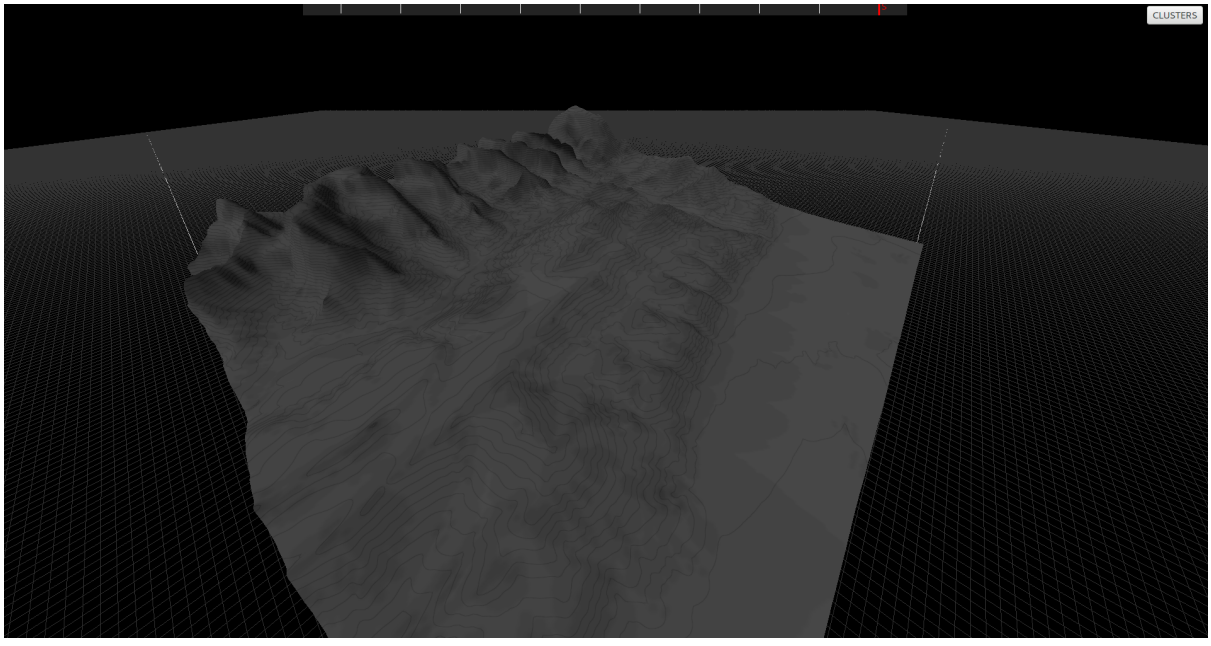

Figure 3.8: Daily illumination overlay. The brighter the area, the more illumination it receives throughout the day.

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 affect temperature and are necessary to model temperature in the system are: *Altitude*, $Temp_{december}$, $Temp_{june}$ and *Lapse rate*. The *altitude* of each point on the terrain is calculated automatically based on the loaded height-map. $\text{Temp}_{december}$ 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 be 6.4 degrees Celsius for each km of altitude gained. This is accepted as the average atmospheric lapse rate under normal atmospheric conditions [3].

Given this information, the temperature is calculated for any point on the terrain given the month and altitude using equation 3.4.

$$T(a,m) = T_{december} + (\frac{6 - |6 - m|}{6} \times (T_{june} - T_{december})) \tag{3.4}$$

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) terrain takes approximately 2 seconds. Figure 3.9 points towards a linear relationship between vertex count and temperature calculation time.

3.2.2.1 Temperature overlay

An overlay can be enabled to get an overview of the temperature at different locations on the terrain (see figure 3.10).

3.2.3 Precipitation

Precipitation is a core part of climate classification and, consequentially, plant life. Arid climates will have very limited annual precipitation and will be the bottleneck for organic life. Tropical climates, on the other hand, where precipitation is plentiful, has 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 of water which 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 (3.11).

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

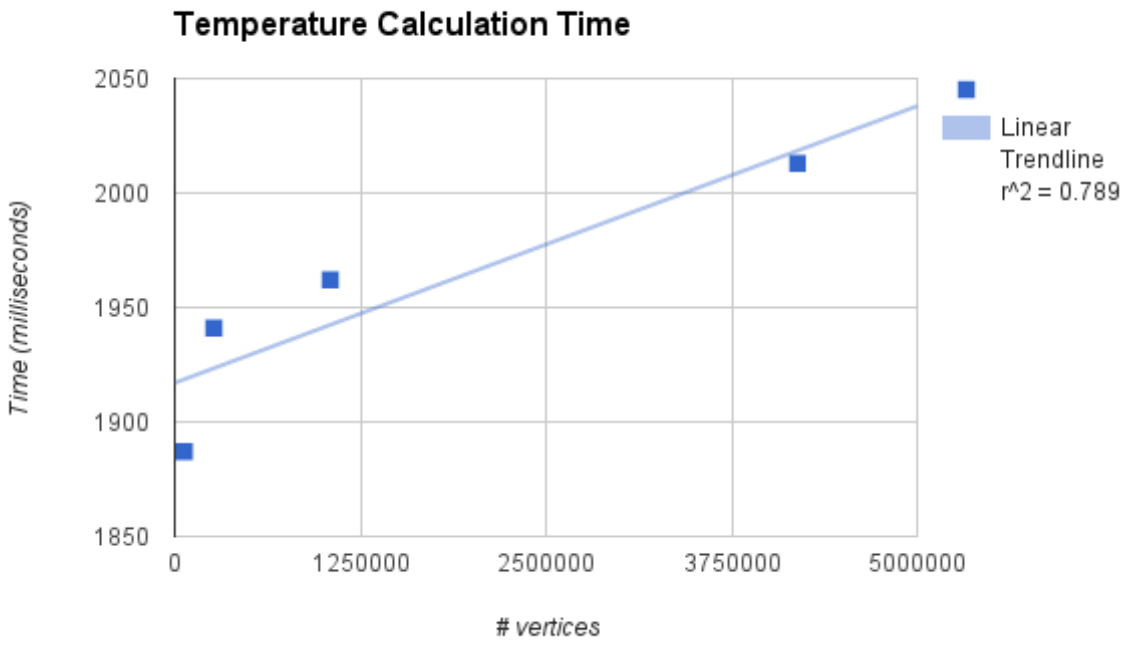

Figure 3.9: Temperature calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024 and 2048 by 2048.

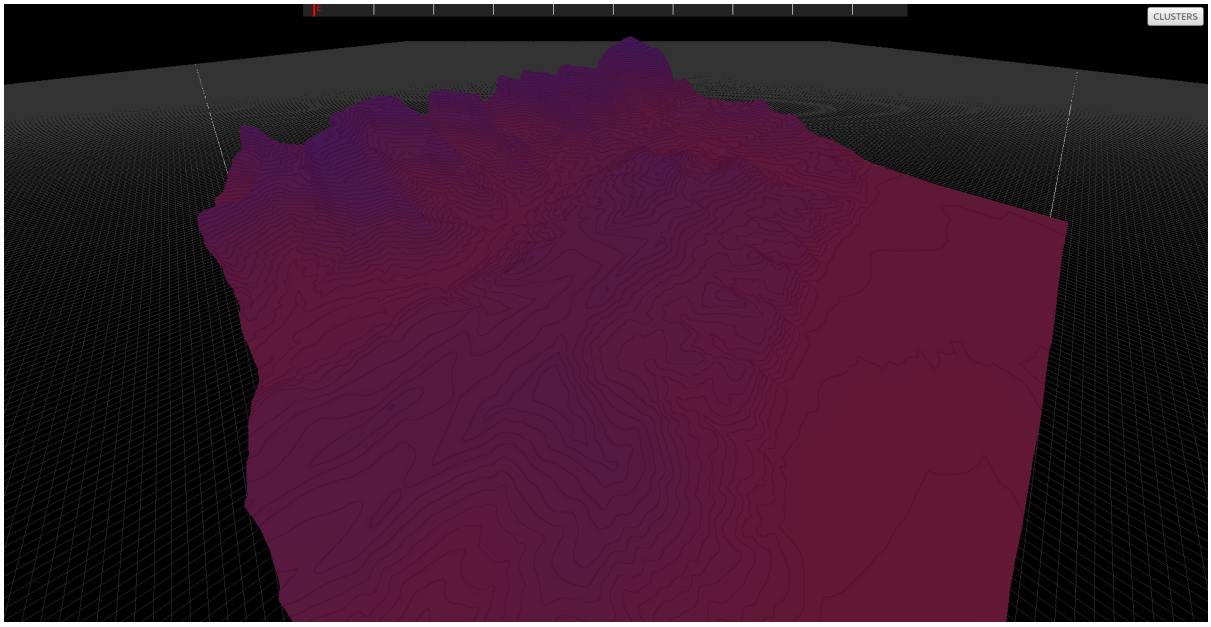

Figure 3.10: Temperature overlay. Colour spectrum ranging from blue (cold) to red (hot). As can be seen, the temperature drops with altitude.

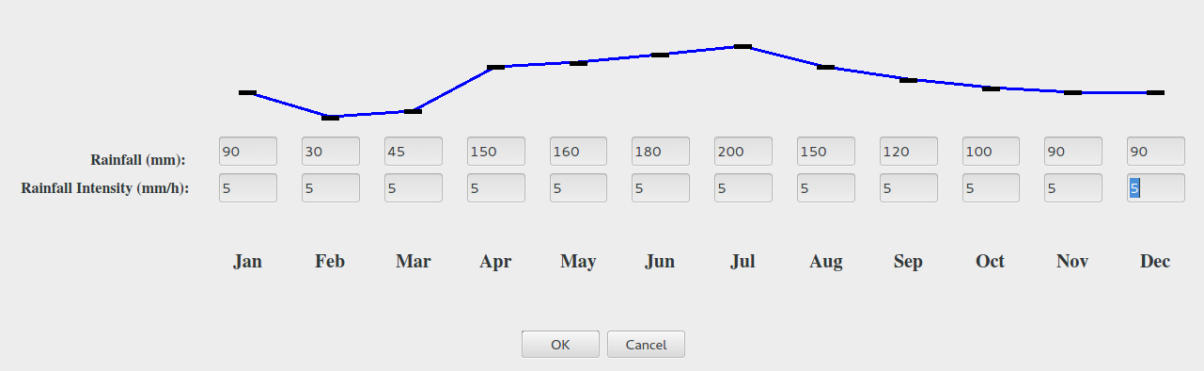

Figure 3.11: Specifying monthly precipitation and precipitation intensity. The user can enter values manually (using the input fields) or interact directly with the graph.

3.2.4 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 capabilities and will result in larger water build-up. In this work, the soil humidity is simply a measure, in mm, of the rainfall which is absorbed by the soil. This is determined using the precipitation information outlined above (3.2.3) along with the *Soil Infiltration Rate*.

3.2.4.1 Soil Infiltration Rate

The *soil infiltration rate* is a measure of the quantity of water which can be absorbed in a given time period. If the rainfall intensity exceeds the soil infiltration rate, it will result in water stagnation (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 3.2.

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

Table 3.2: Soil infiltration rates for different soil types [4]

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.

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

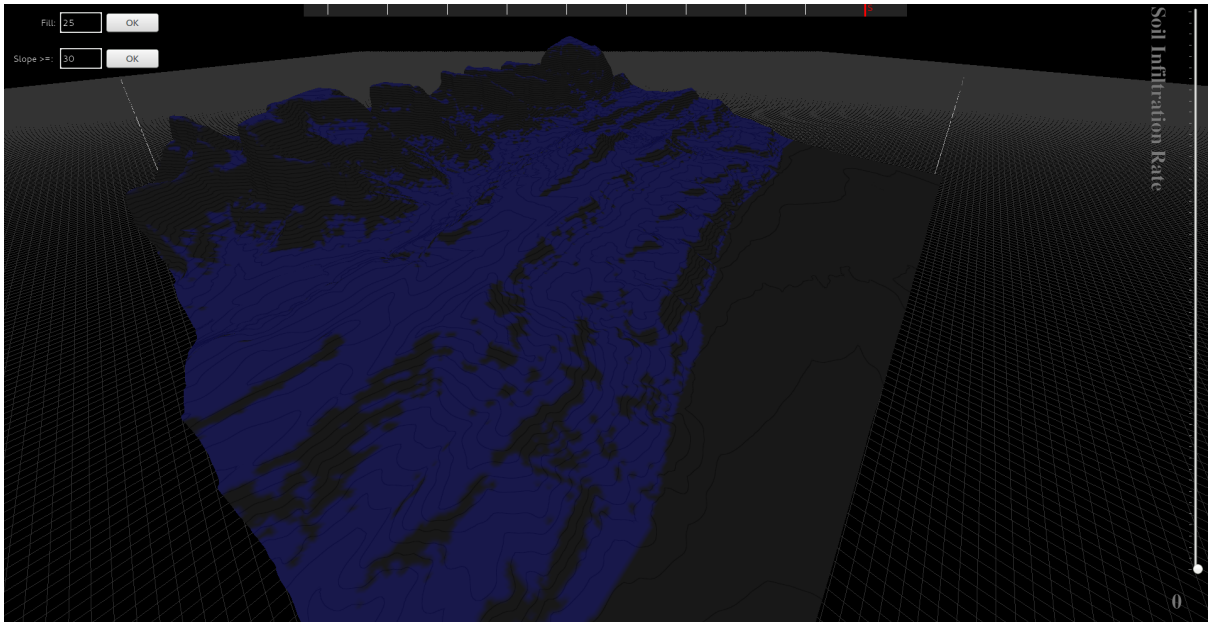

Figure 3.12: Editing the soil infiltration rate on the terrain. Top left are the controllers for the *filling* and *slope-based* tools. On the right is the slider to configure the soil infiltration rate of the *paint brush*.

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 (3.12).

These tools can be used interchangeably and users are encouraged to do so. Configuring the soil infiltration of the terrain in figure 3.12, 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.).

3.2.4.2 Monthly Soil Humidity Calculation

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 the rainfall which is actually absorbed by the soil for each month. This represents the soil humidity in our system and is calculating using equation 3.5.

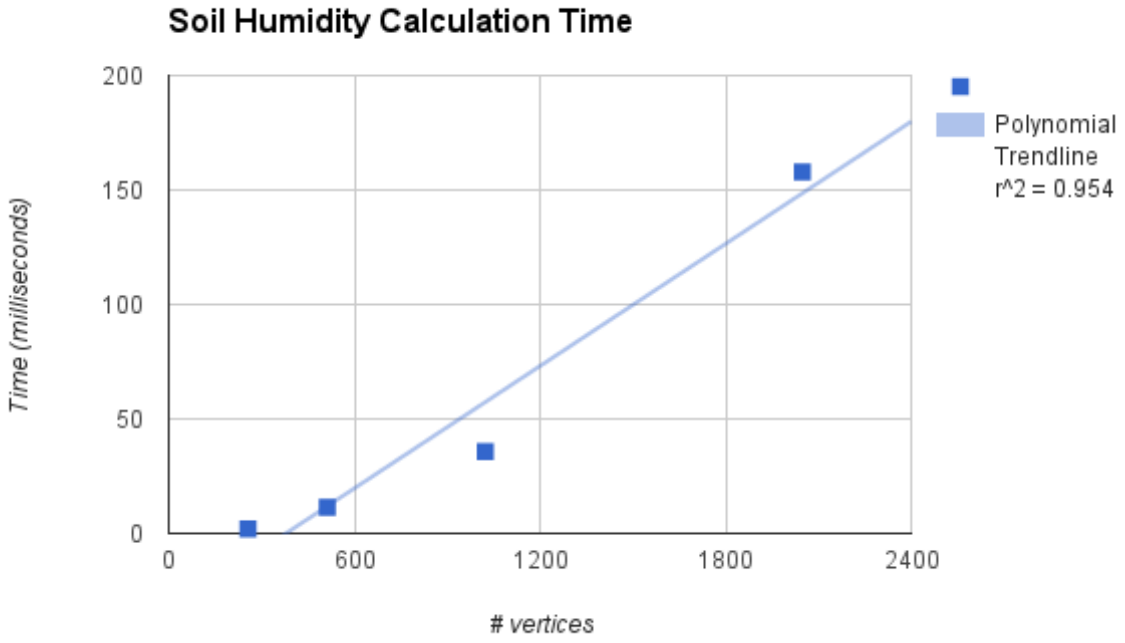

Figure 3.13: Soil humidity calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024 and 2048 by 2048.

$$S_h(R_q, R_i, S_{ir}) = \frac{S_{ir}}{R_i} \times R_q \tag{3.5}$$

where:

- S_h is the soil humidity, 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.

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 158 milliseconds (3.13). There is a linear relationship between vertex count and calculation time (3.13).

3.2.4.3 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

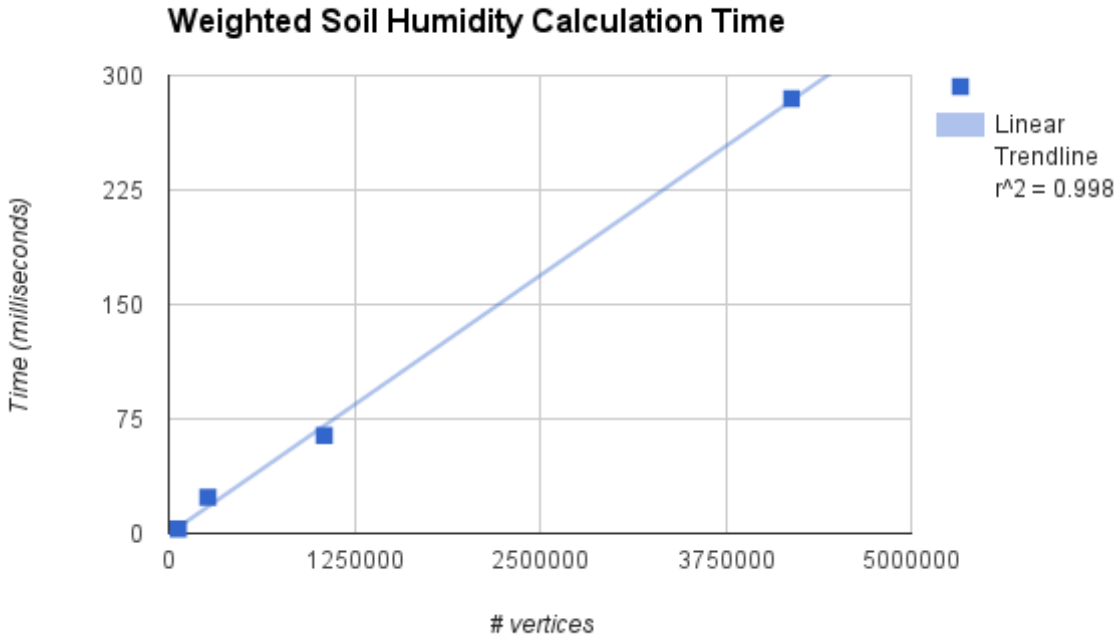

Figure 3.14: Weighted soil humidity calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024 and 2048 by 2048.

drought caused by an arid month can be counteracted to some extent by the precipitation of previous months. To model this, a moving weighted average soil humidity is calculated for each month using equation 3.6.

$$S_{wh}(m) = (\frac{1}{2} \times S_h(m)) + (\frac{1}{3} \times S_h(m-1)) + (\frac{1}{6} \times S_h(m-2)) \tag{3.6}$$

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 285 milliseconds (3.14). There is a linear relationship between vertex count and calculation time (3.14).

3.2.4.4 Soil Humidity Overlays

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 (see figure **??**).

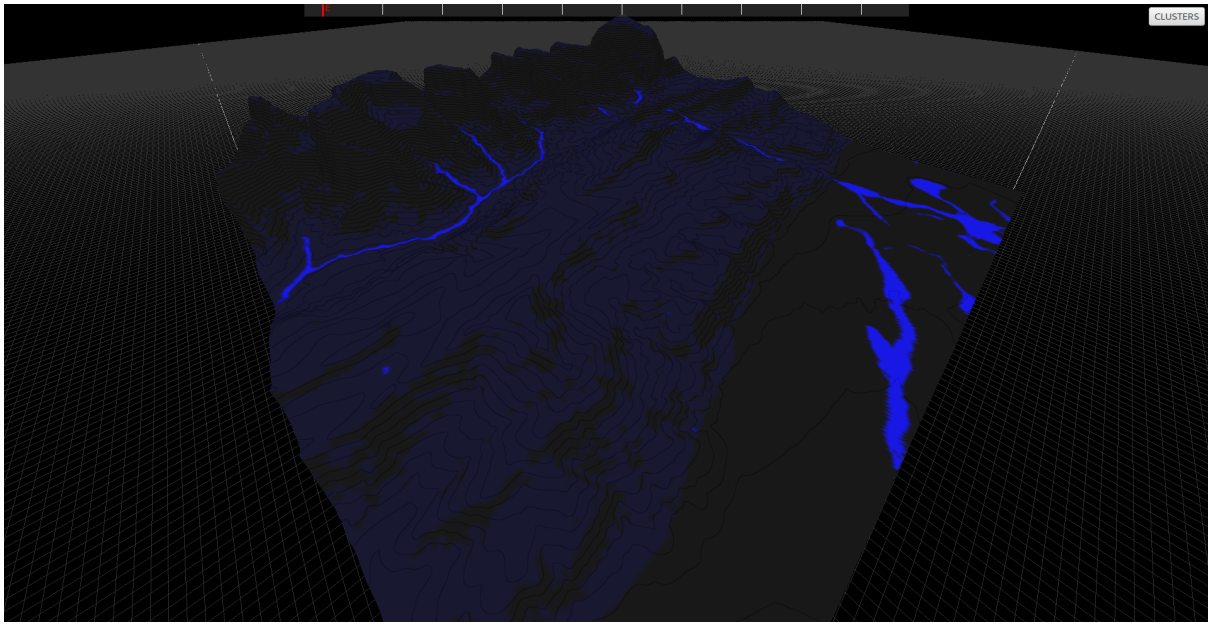

Figure 3.15: Soil humidity terrain overlay. Colour spectrum ranging from black (zero humidity) to blue (standing water).

3.2.5 Slope

Some plants, especially those with a big biomass, can struggle to grow on steep slopes. 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 (34 milliseconds for 4 million vertices) and the relation between vertex count and calculation time is linear (**??**

3.2.5.1 Slope Overlay

To better visualize the slope throughout the terrain, a *slope overlay* can be enabled (see figure 3.17).

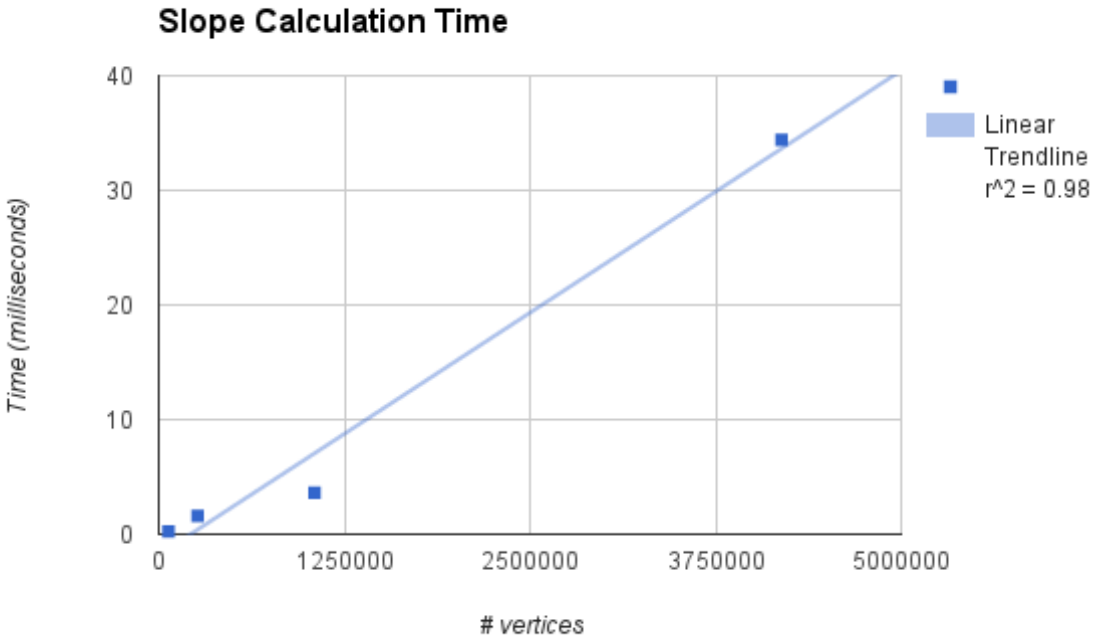

Figure 3.16: Slope calculation time based on vertex count. Analysis performed for terrains with dimensions: 256 by 256, 512 by 512, 1024 by 1024 and 2048 by 2048.

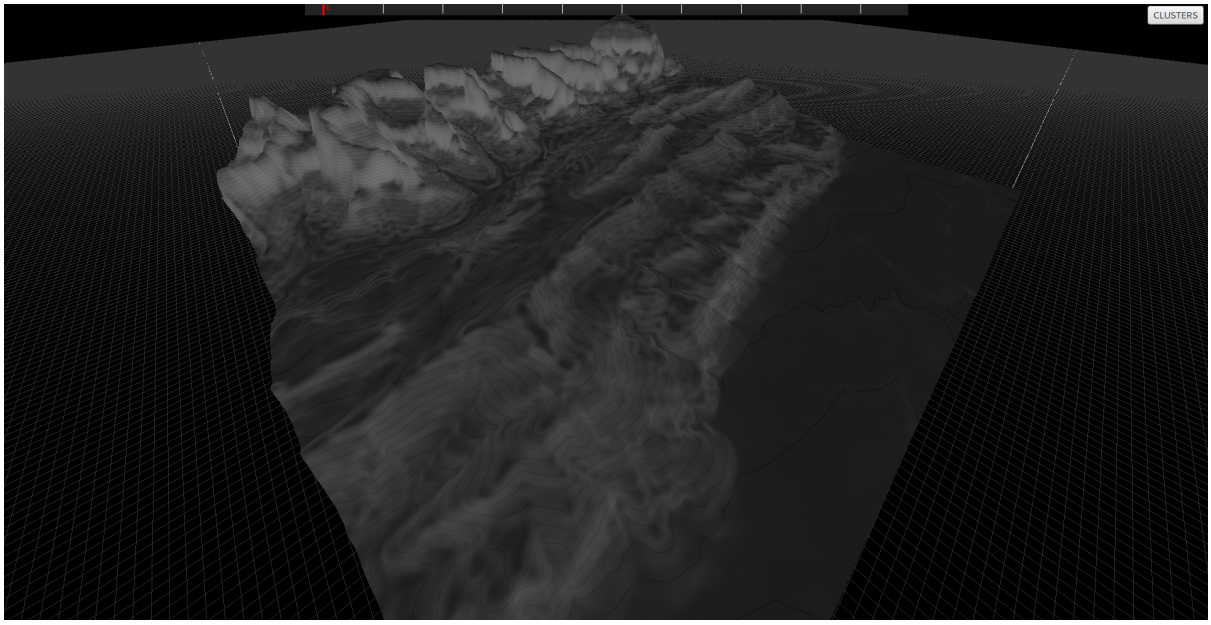

Figure 3.17: Slope visual overlay. Colour spectrum ranging from black (zero degree slope) to white (90 degree slope).

3.3 Rivers & Streams

Essential to the realism of virtual rural terrains are water networks. These water networks are constituted of rivers and streams and are the consequence of water being transported by gravity from higher to lower grounds. The algorithm used to model water movement on the terrain is outlined below along with details concerning the GPU implementation.

3.3.1 Algorithm Overview

Precipitation, precipitation intensity and soil infiltration is used to calculate the soil humidity, S_h (see equation 3.5). The value of which equates to the quantity of water, in millimetres, absorbed by the soil. The standing water, $W_{standing}$, is the quantity of water which isn't absorbed by the soil and can be calculated using equation 3.7.

$$W_{standing} = R_q - S_h \tag{3.7}$$

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, $\mathrm{W}_{standing}$, for each vertex, a hydrostatic pipe-model similar to that of Stava et al. [vBBK08] is used to determine water movement and build-up on the terrain. The hydrostatic pipe-model works by iteratively attempting to evacuate water from source vertex V to any of it's eight surrounding vertices.

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 surrounding 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 concept is identical when ported it to a third dimension.

Each vertex V_n, is characterised by it's position (n), terrain height ($TerrainHeight_n$), water height ($WaterHeight_n$) and absolute height ($AbsoluteHeight_n = TerrainHeight_n + WaterHeight_n$). Using this data it is possible to calculate the water evacuation capacity, WEC, of each terrain vertex (see equation 3.8). The value of which effects how much water is evacuated and how it is split amongst surrounding vertices (3.3.2).

$$WEC = 2 \times TerrainHeight_n - AbsoluteHeight_{n-1} - AbsoluteHeight_{n+1} \tag{3.8}$$

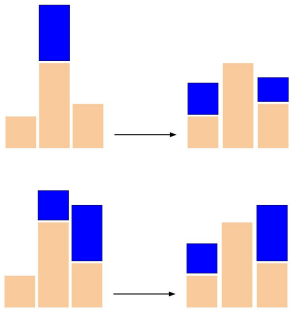

Figure 3.18: Example water-evacuation scenarios where all water can be evacuated from source vertex (middle).

3.3.2 Water Evacuation Approaches

Using the water evacuation capacity *WEC* along with table 3.3, one of three evacuation approaches are used: *all water is evacuated*, *a portion of the water is evacuated* or *no water is evacuated*. Each approach is discussed in detail here.

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

3.3.2.1 All water is evacuated

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 3.9 and 3.10. Examples of such situations are illustrated in figure 3.18.

$$W_{n-1} = \frac{TerrainHeight_n - AbsoluteHeight_{n-1}}{WEC} \times WaterHeight_n \tag{3.9}$$

$$W_{n+1} = \frac{TerrainHeight_n - AbsoluteHeight_{n+1}}{WEC} \times WaterHeight_n \tag{3.10}$$

3.3.2.2 A portion of water is evacuated

This approach is used when the water evacuation capacity, *WEC*, is not large enough to purge all water but sufficient to purge a subset thereof. The portion of water which can

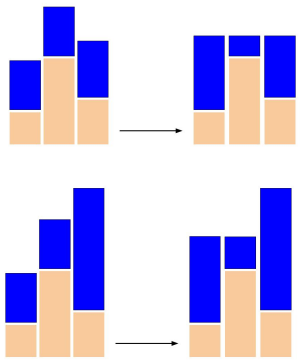

Figure 3.19: Example water evacuation scenarios when only a portion of water can be evacuated from the source vertex (middle).

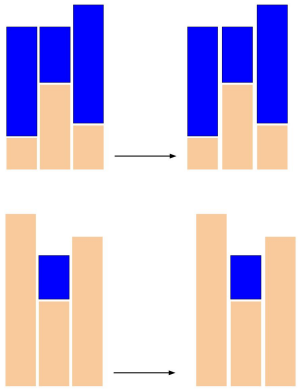

Figure 3.20: Example water evacuation scenarios where no water can be evacuated from the source vertex (middle).

be evacuated, $W_{evacuate}$ is calculated using equation 3.11. Examples such scenarios are illustrated in figure 3.19.

$$W_{evacuate} = AbsoluteHeight_n - W_{level} \tag{3.11}$$

Where:

- W_{level} is the water level to which water can be evacuated (see equation 3.12).

$$W_{level} = \frac{AbsoluteHeight_{n-1} + AbsoluteHeight_n + AbsoluteHeight_{n+1}}{3} \tag{3.12}$$

3.3.2.3 No water is evacuated

This situation occurs when there is no capacity for water evacuation and therefore no water can be transported to surrounding vertices. Examples of such scenarios are illustrated in figure 3.18.

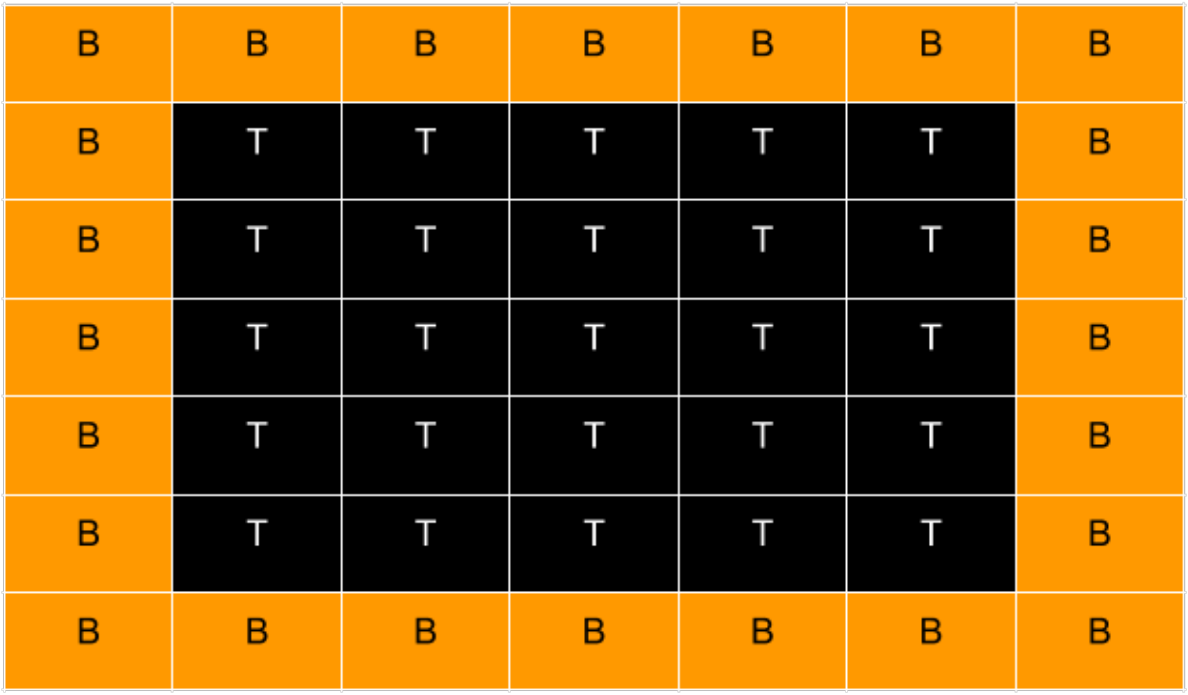

Figure 3.21: Border of vertices generated around the terrain to cater for water evacuation at the extremities. Orange vertices form the border.

3.3.3 Terrain Extremities

When attempting to evacuate water from vertices on the terrain extremities, one or multiple vertices are missing. 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 3.21. The height of each border vertex is calculated such that the slope remains unchanged. During the water-flow simulation, water is permitted to evacuate to border vertices but water cannot build-up on them.

3.3.4 Stopping the simulation

The flow of water on the terrain can be measured by keeping track of the number of vertices which were 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 grounds 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 flow during the course of the simulation.

The user is able to override this, however, if he wishes to model a different water network which requires a shorter or longer water-flow simulation.

3.3.5 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.

Below are discussed the challenges faced when implementing the water-flow algorithm to run on this heavily parallel architecture: *Memory management*, *Intra work-group data consistency management* and *Inter work-group data consistency management*.

3.3.5.1 Memory management

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 3.3.4).

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 3.4 summarizes the global memory allocations necessary for the GPU implementation of the water-flow simulation.

3.3.5.2 Intra work-group data consistency management

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 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 3.22).

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 3.3.3)
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 3.4: 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.

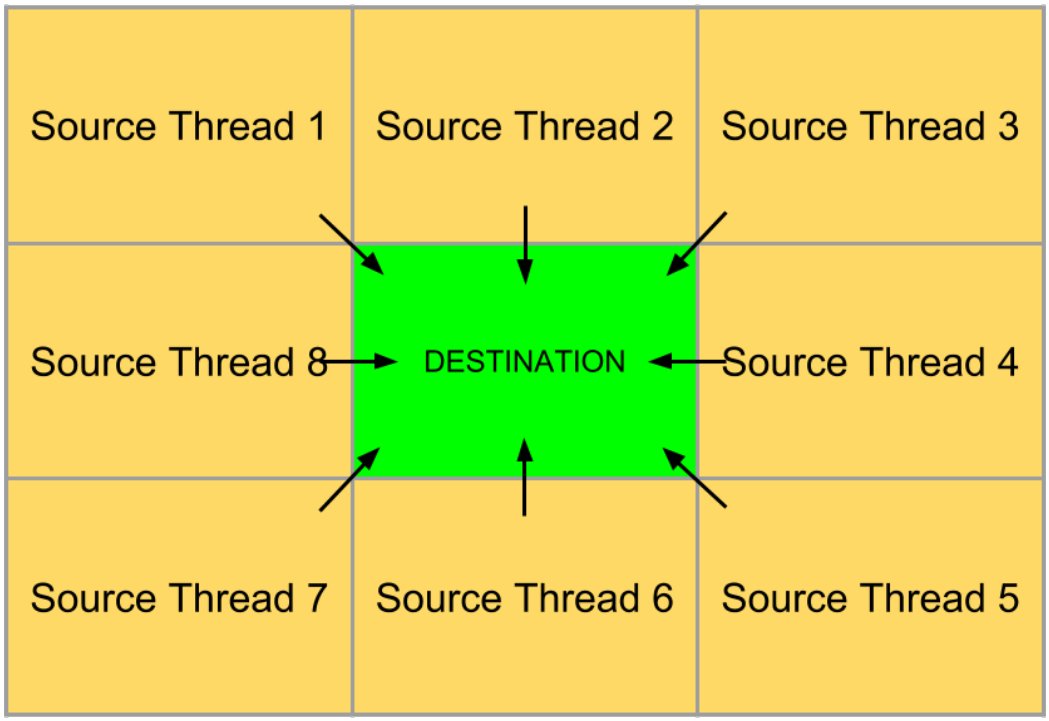

Figure 3.22: Memory conflict cause by 8 surrounding threads attempting to write to a single destination memory location.

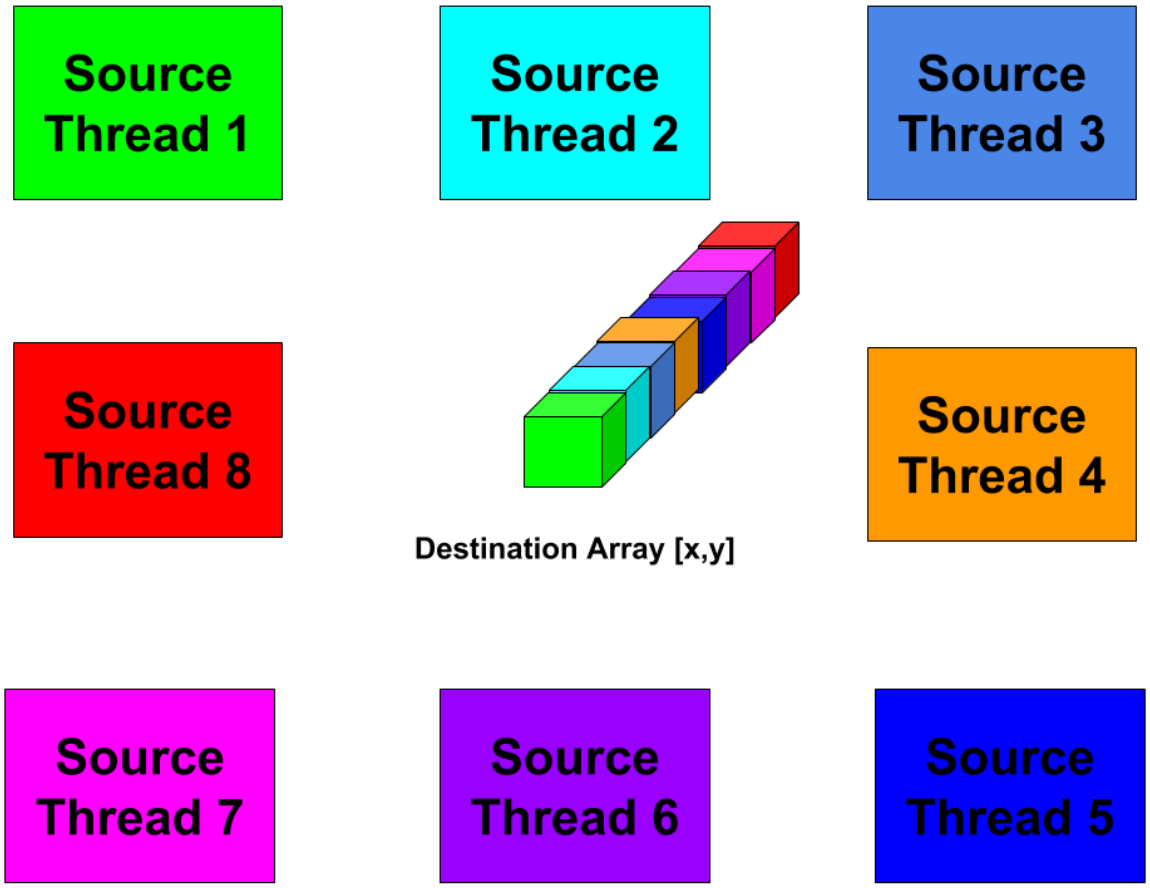

Figure 3.23: Memory conflict prevention by using a three dimensional array to aggregate the water to add.

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 figure3.23).

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

3.3.5.3 Inter work-group data consistency management

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 3.24), vertical direction (figure 3.25) and diagonal direction (figure 3.26). 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

Figure 3.24: Global memory allocation requirement (green) to cater for inter work group memory access in the horizontal direction. WG = work group

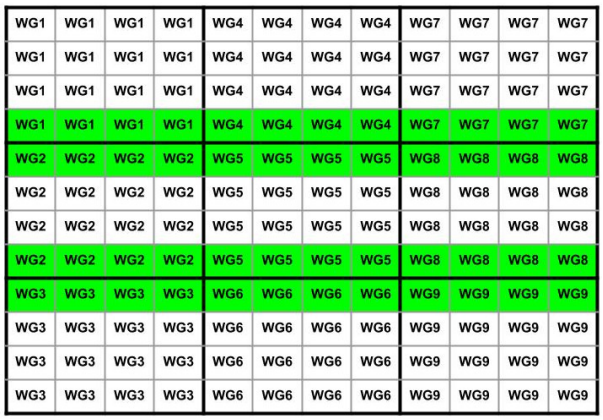

Figure 3.25: Global memory allocation requirement (green) to cater for inter work group memory access in the vertical direction. WG = work group

complete, this data is then reduced by selected threads and written back to the water height-map.

3.3.6 Performance

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

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

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

Figure 3.26: Global memory allocation requirement (green) to cater for inter work group memory access in the diagonal direction. WG = work group

As can be seen in the results summarised in figure 3.27, the water-flow simulation takes approximately 22 seconds to complete for a terrain with over one million vertices (1024×1024 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 3.28).

3.3.7 Results

The U.S. Geological Survey [5] freely provides detailed elevation data for the north American continent. Google-Earth [6] provides detailed satellite images of the same locations. To test the water-flow simulation, it is run on actual terrains generated using data from the U.S. Geological Survey. The resulting rivers and streams which form are compared with corresponding satellite images to see if the main water bodies match. For the tests to be as accurate as possible, nothing other than the water-flow simulation is performed on the terrains.
The results illustrated in figures 3.29, 3.30 and 3.31 show that the simulation successfully reproduces core water networks. Note that the replicated terrains also include water bodies and rivers which aren't 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.

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

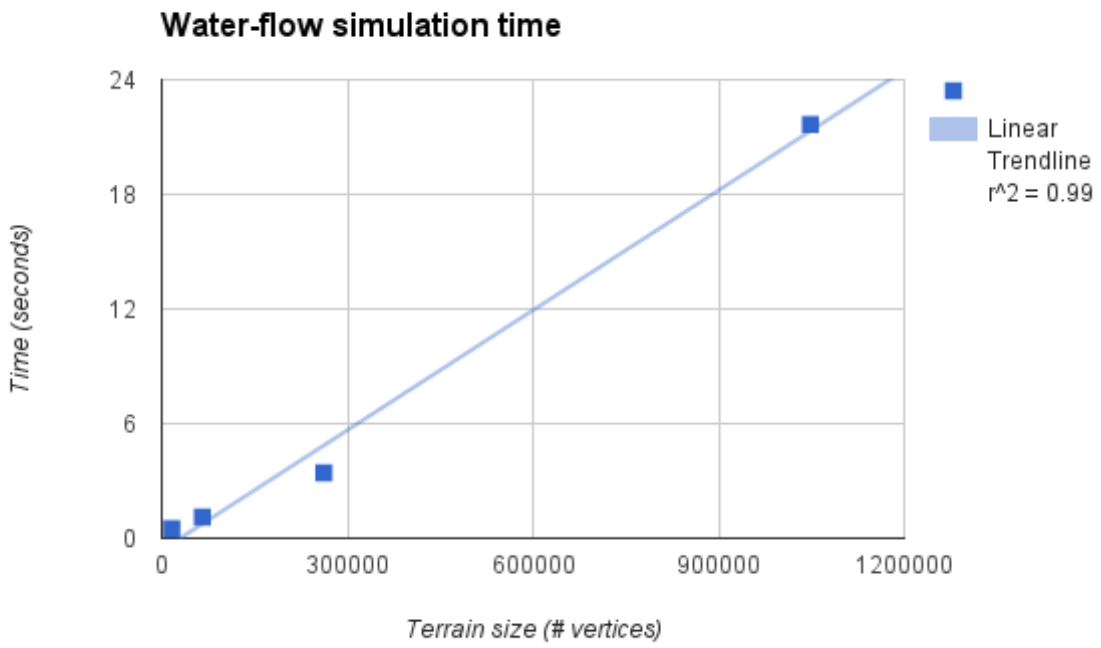

Figure 3.27: Total water-flow simulation time for 100 millimetres per terrain vertex. Analysis performed for terrains with dimensions: 128 by 128, 256 by 256, 512 by 512 and 1024 by 1024.

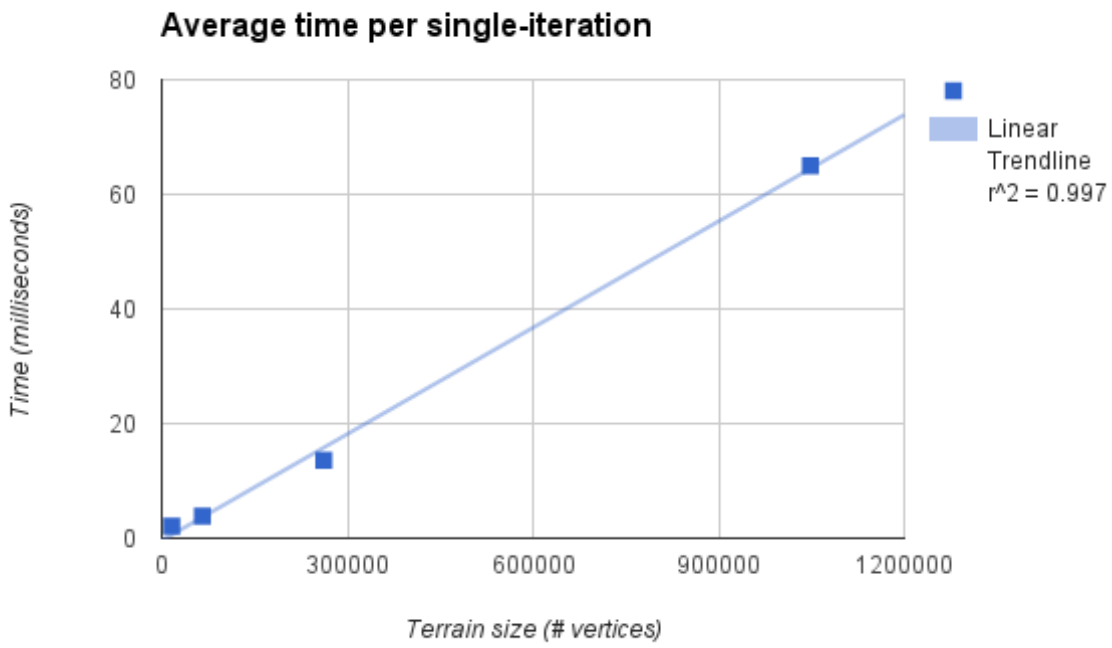

Figure 3.28: Average time for a single iteration of the water-flow for 100 millimetres per terrain vertex. Analysis performed for terrains with dimensions: 128 by 128, 256 by 256, 512 by 512 and 1024 by 1024.

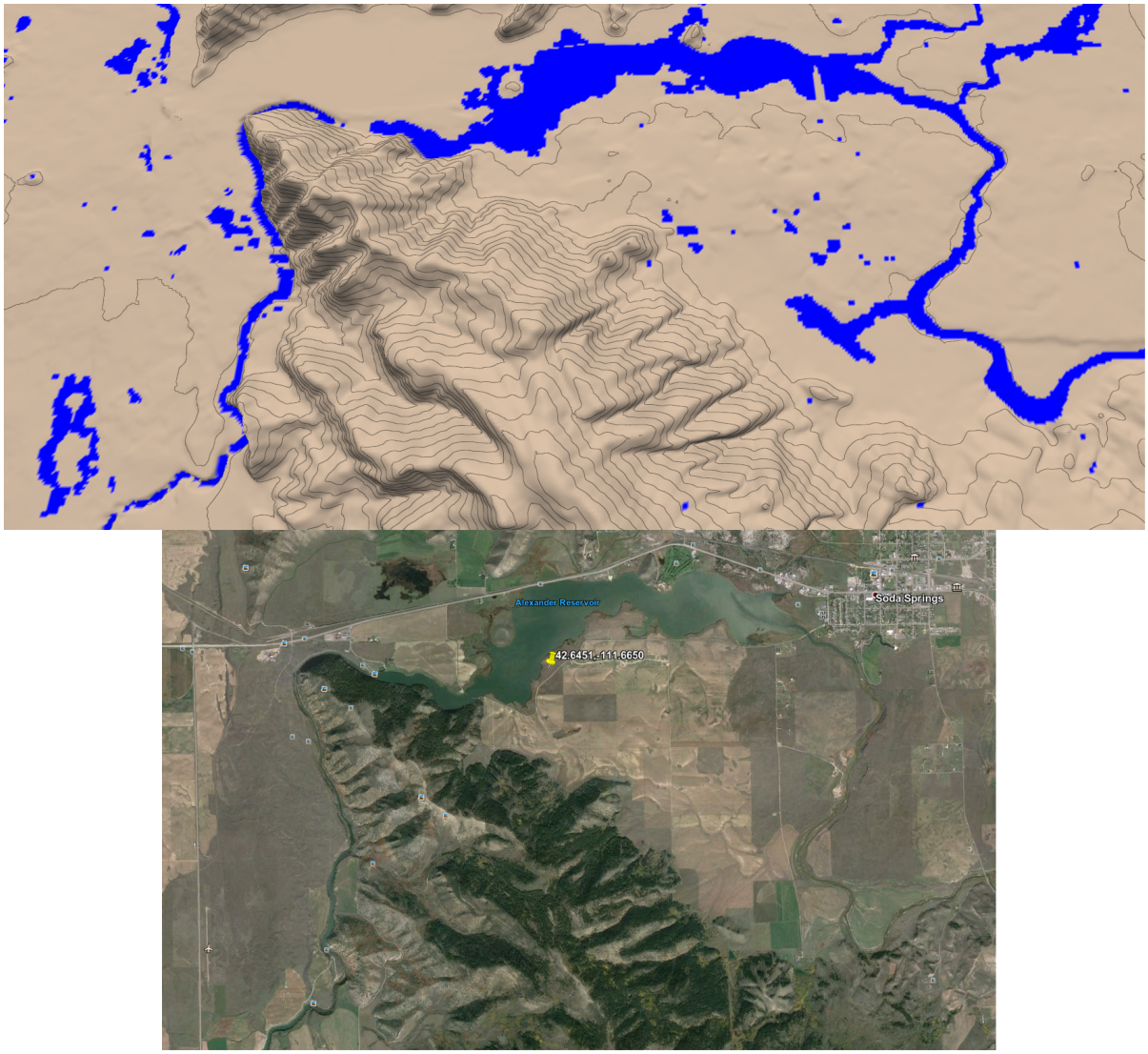

Figure 3.29: Comparison of real world water-networks (bottom) with those produced using only the water-flow simulation (top).

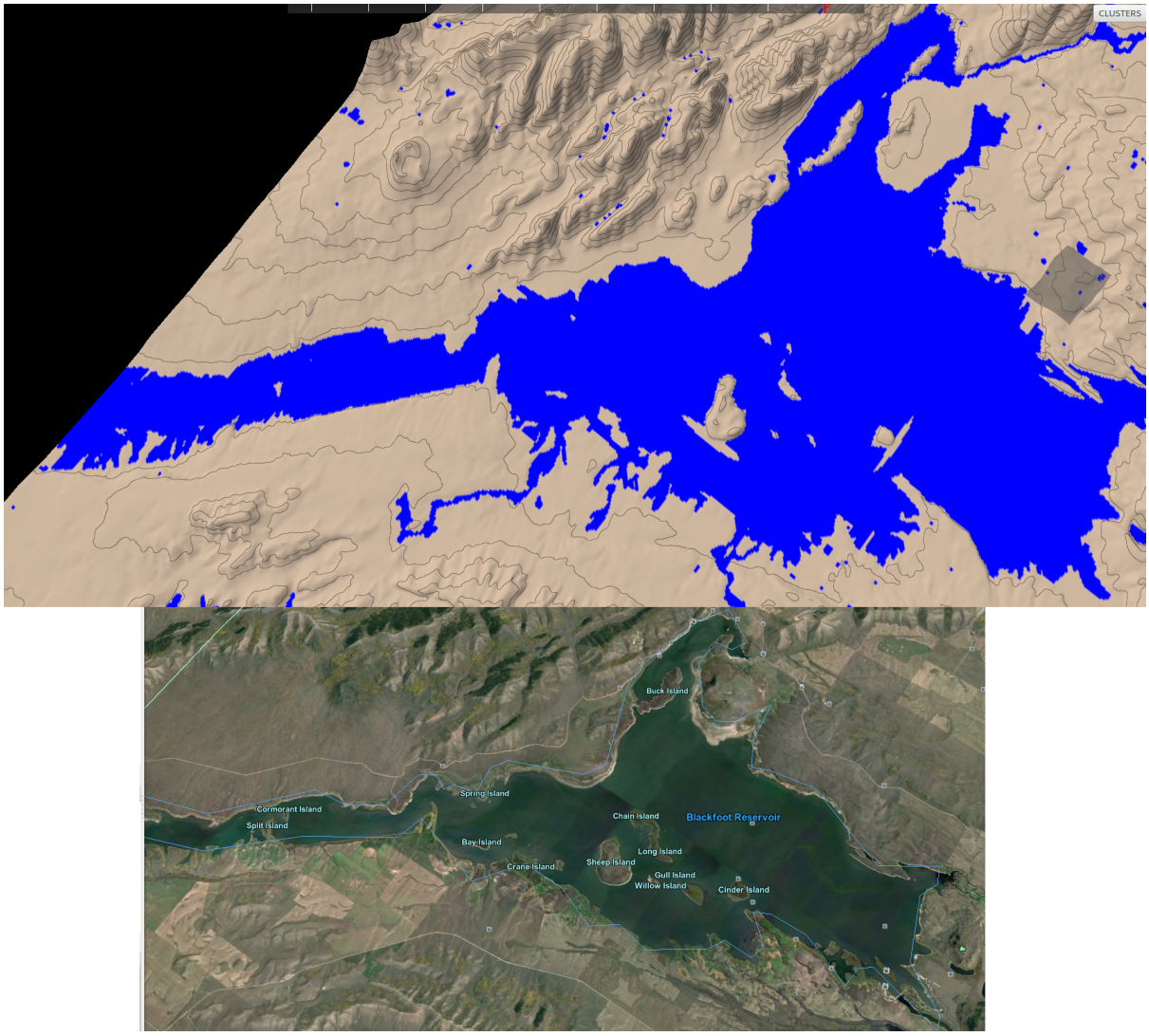

Figure 3.30: Comparison of real world water-networks (bottom) with those produced using only the water-flow simulation (top).

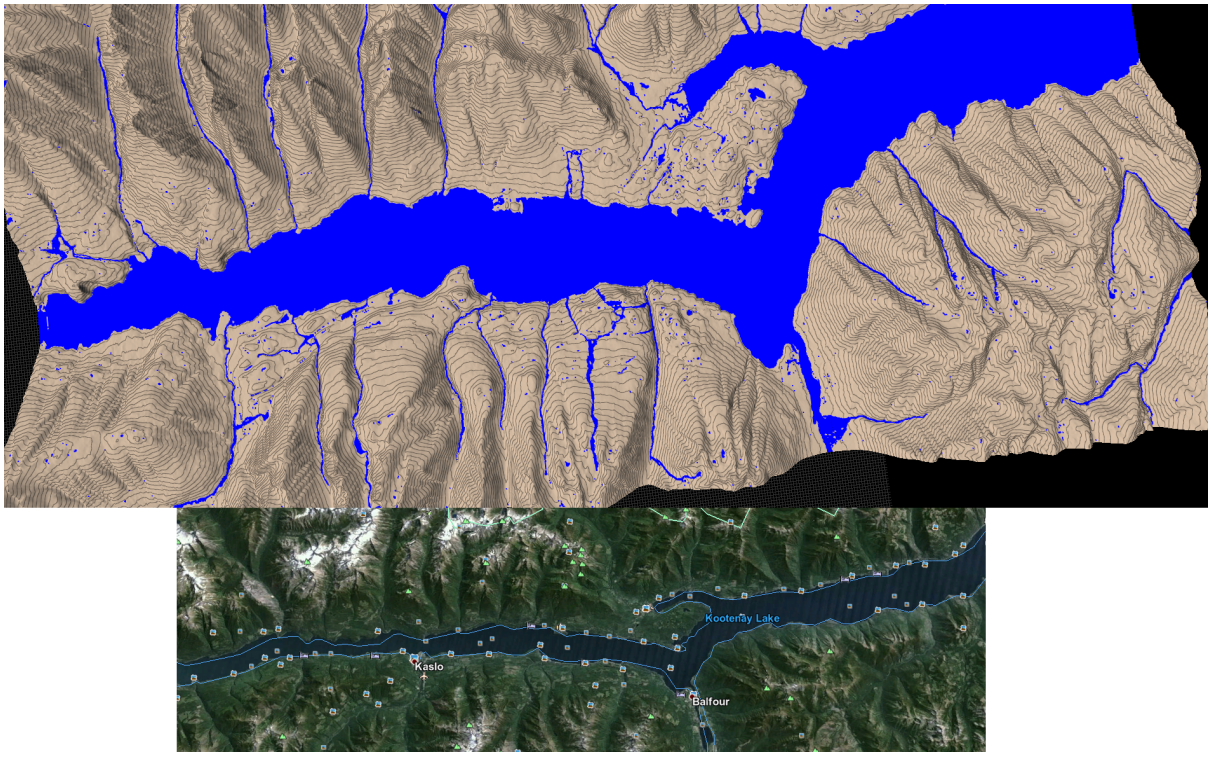

Figure 3.31: Comparison of real world water-networks (bottom) with those produced using only the water-flow simulation (top).

3.4 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 and groundwater. Users can place such water bodies by using the *flood-fill* discussed in more detail below.

3.4.1 flood-filling

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}. The process continues until there are no more valid points in which to propagate to or the terrain extremity is reached. When flood-filling is activated, the user is requested to specify the seed point by simply clicking it with the mouse pointer. The flood-filling algorithm produces the 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 a wrong seed point.

Figures 3.32 and 3.33 show that the tool can be used to successfully place different types of water bodies on terrains.

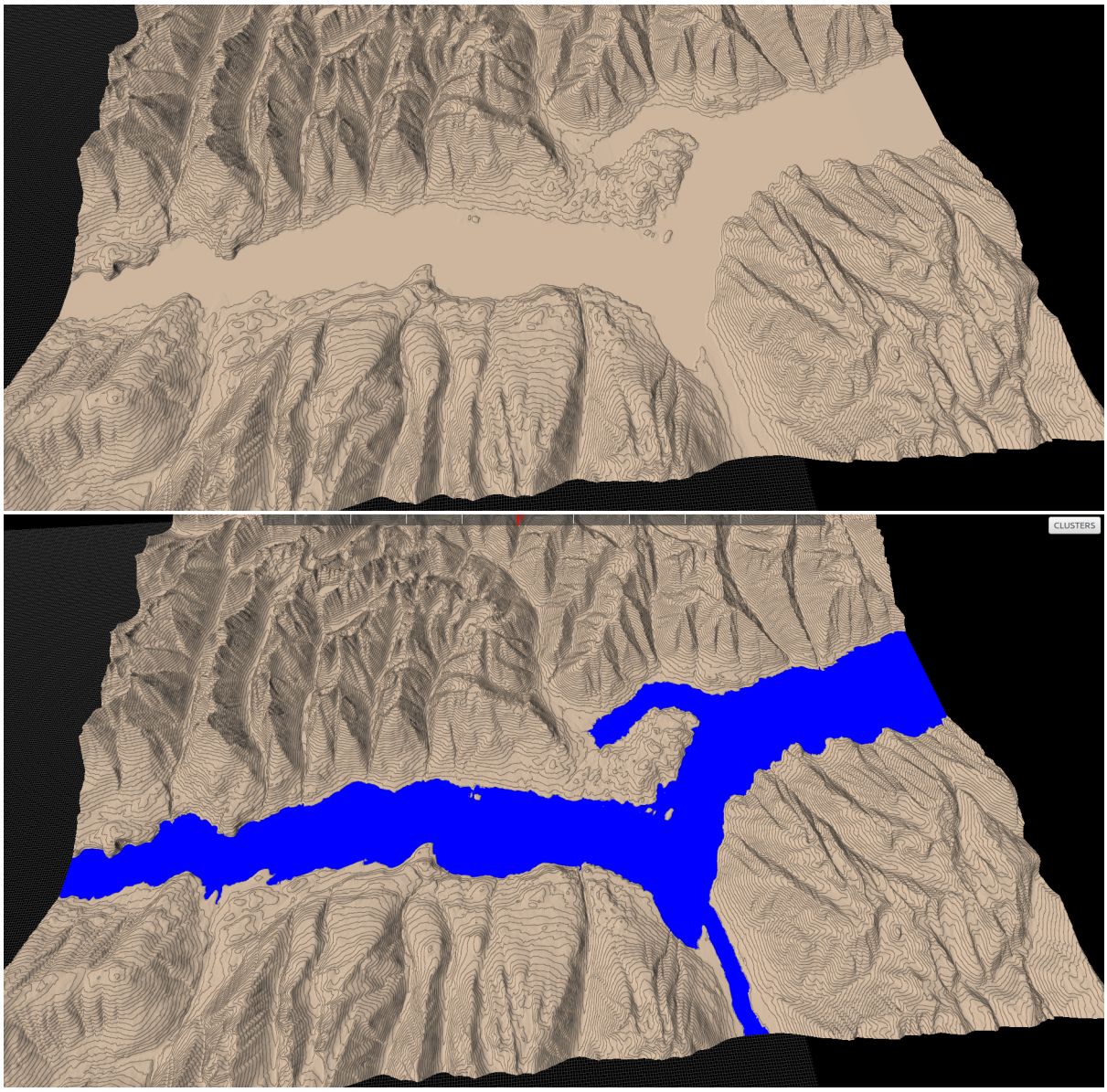

Figure 3.32: Terrain before (top) and after (top) using the flood-fill tool to place a lake.

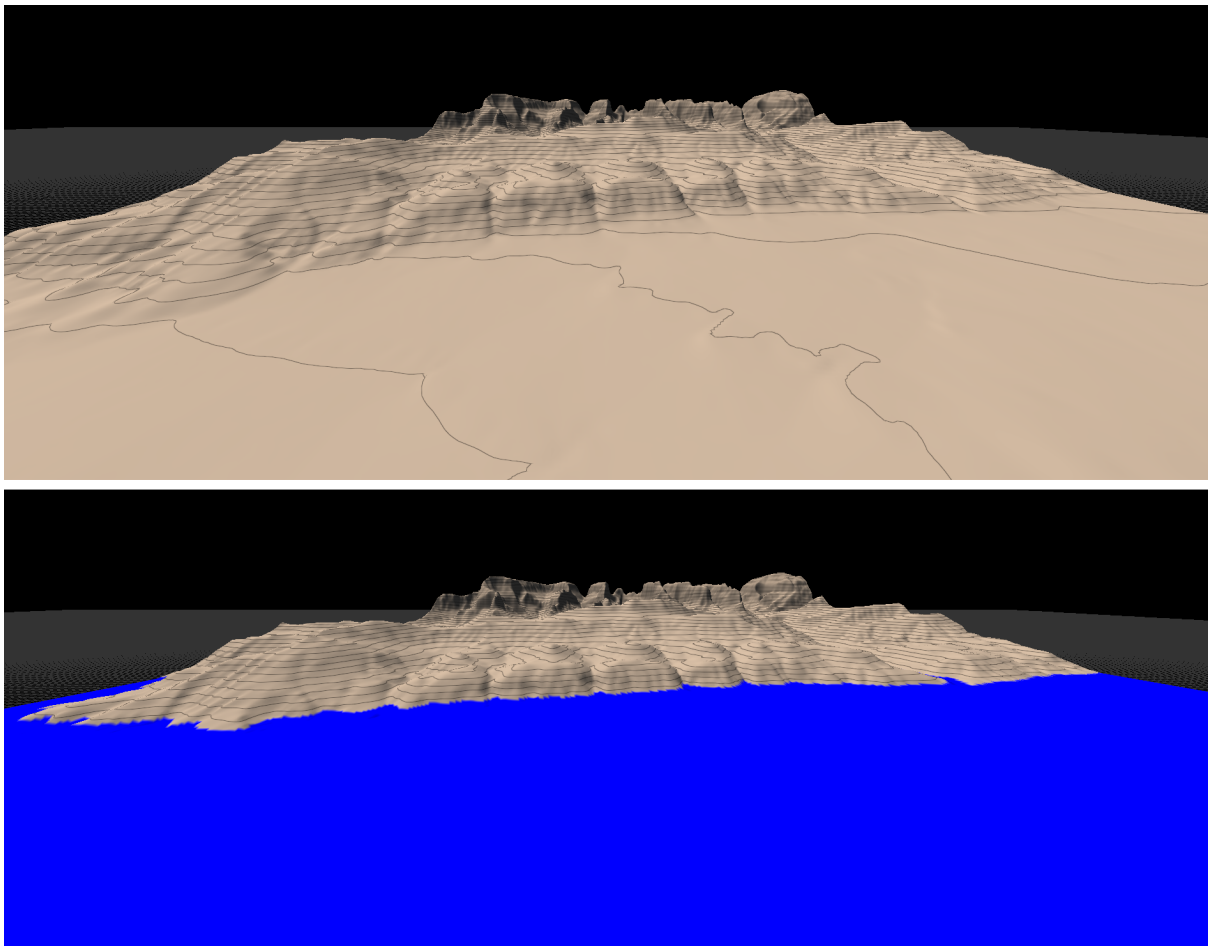

Figure 3.33: Terrain before (top) and after (top) using the flood-fill tool to place a large water body (sea/ocean).

Chapter 4

Clustering

Each point on the terrain has a number of associated resource properties (summarised in table 4.1). These properties are used as input to an ecosystem simulator to determine vegetation and vegetation density. Running an ecosystem simulation for each point on the terrain would be excessive, however, both in terms of computation time and data overlap. The data overlap would be the result of simulations being run for points with very similar resource properties.

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 4.1: Resource properties associated with a single point on the terrain.

To make the number of ecosystem simulations which need to be run manageable and ensure each one generates distinct distribution data, clustering is performed on the terrain based on the resources of individual points. The type of clustering algorithm used is *K-Means Clustering*, discussed in the dedicated *K-Means Clustering* section. To accelerate the clustering algorithm and strive for real-time results, a GPU implementation was made. Details of which can be found in the *GPU Implementation* section. To conclude the chapter, performance and results are analysed in sections *Performance* and *Results* respectively.

4.1 K-Means Clustering

K-means is a well known and broadly used clustering algorithm which is used to group numerical data into clusters based on their euclidean distance from a set of K cluster means.

Given a set of data points P and a configured value k, a generic k-means clustering algorithm performs the following:

1. Choose K points at random to act as the seed cluster means CM.
2. Iterate through every data point P_n from P and calculate its Euclidean distance with each cluster mean of CM using equation 4.1. 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.

$$D(A, B) = \sum_{n=1}^{nvalues} (value_n(A) - value_n(B))^2 \tag{4.1}$$

Where:

- **A** and **B** are either data points or a cluster mean.
- **nvalues** are the number of numerical values associated with each data point.
- $\mathbf{value}_n\mathbf{(A)}$ is the data value n of data point A.

4.1.1 Customising the Euclidean Distance Calculation

The Euclidean distance calculation outlined in equation 4.1 works well when all values are in the same dimension. 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 musch of an influence on clustering than a one degree change in temperature. To equalise the effect on clustering across all resources, weighting is used when calculating the Euclidean distances (see table 4.2).

Value	**Weighting**
Slope	1
Temperature	1
Illumination	1
Soil Humidity	0.1

Table 4.2: Resource weighting when calculating Euclidean distances between.

4.1.2 Choosing Seed Cluster Means

A downside of classic K-means clustering techniques is that they are non-reproducible. This is because the final clusters depend heavily on the initial seed cluster means. As these are selected at random, different runs will result in different clusters. Reproducibility is important here, however, as if the clusters can't be reproduced neither will the final terrain.

A solution to this problem is to initialise the cluster means using pre-determined points on the terrain rather than at random. However, it is also good for the initial seed points to contain distinct resource properties in order for the final clusters to form faster. To attempt to fulfil both these requirements, the seed points are selected at equal distances on the terrain diagonal. This ensures reproducibility as the same seed points will always be selected. This attempts to cater for the second requirement also as it ensures the seed points will have good terrain coverage.

4.1.3 Configuring the Number of Clusters K

When vegetation is to be placed on the terrain, K ecosystem simulations will need to run. Although larger values of K will potentially result in more realistic vegetation distributions, it will also require more processing time. As a consequence, choosing a value for K is about finding a balance between realism and processing time. This balance depends on user requirements which is why it is possible to configure it manually.

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 4.3: 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 count respectively.

4.2 GPU Implementation

Performing K-Means clustering is an *O(KN)* problem where K is the number of clusters. As a consequence, given a number of clusters K, the processing time will increase linearly with the size of the terrain. For large terrains with millions of vertices, this would take time and, consequentially, have a negative effect on user experience. To accelerate the clustering process and improve user-experience it is implemented to make use of the heavily parallel architecture of the GPU.

As mentioned previously, copying data to/from CPU to/from GPU is a costly process. To prevent these costly transfer operations, all data required for the clustering algorithm (table 4.3) is copied to the GPU at the start of the clustering process and no further transfers are performed until completion.

4.2.1 Calculating cluster membership

Given the cluster means for iteration i, the process must determine to which cluster each terrain vertex belongs. The cluster membership of individual terrain vertices are calculated in parallel.

4.2.2 Calculating the new cluster means

Once all terrain vertices have been assigned to a cluster, the new cluster means need to be calculated. To do this, all terrain vertices must be processed in order to determine their cluster memberships and contribute to the new mean.

Parallelism is achieved by calculating the new means in parallel. Within a work group (group of cores which are guaranteed to run in parallel and have access to shared mem-

ory), when each core calculates it's distance from individual cluster means, it loads its associated resource data into a unique index of fast-access shared memory. It also stores in shared memory the calculated cluster membership. This data is then used to calculate within each work-group, k new work group cluster means, $ClusterMean_{kxy}$, in parallel. The k work-group cluster means are then stored to global memory along with the member count for each cluster.

Finally, the global cluster means are calculated by k cores in parallel using the work group cluster means and the associated member counts as weightings.

4.2.3 Storage Optimization

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. Calculating the Euclidean distance from a given point A to a cluster mean is identical to calculating the difference between two points on the terrain. As a consequence, it is only necessary to use a single months temperature data to establish terrain clusters, saving vital GPU storage space.

4.3 Performance

As mentioned previously, the user must specify the requested cluster count k. In order to find a suitable value for k, the user will need to trial a number of different values. To minimize any negative effect on user-experience, therefore, it is important 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.

4.3.1 CPU Performance

Figure 4.1 shows the time it takes for the clustering to run with different terrain sizes and number of clusters. Notable results from this data are:

- Ten clusters take on average five times longer to generate than a single cluster.
- The same number of clusters take on average four times longer to generate for a terrain of size 512 by 512 than for terrain of size 256 by 256. This increases to six when comparing a terrain of size 512 by 512 with one of size 1024 by 1024.

Although the clustering time is reasonable for smaller terrains, this processing time increases sharply with the terrain size. This is especially true when combined with an increase in the number of clusters to generate.

4.3.2 GPU Performance

The same tests run using the GPU implementation are displayed in figure **??**. Notable results are:

- Ten clusters take on average twice the time of a single cluster to generate.
- The processing time increases proportionally with the number of terrain vertices.

Because different clusters are managed in parallel on the GPU, increasing the cluster count has a much smaller impact on performance than for the CPU implementation.

Individual terrain vertices are managed in parallel on the GPU. Although this greatly speeds up the clustering process, because the number of terrain vertices far outnumber the number of GPU cores, the clustering time is still heavily impacted by increases in terrain size.

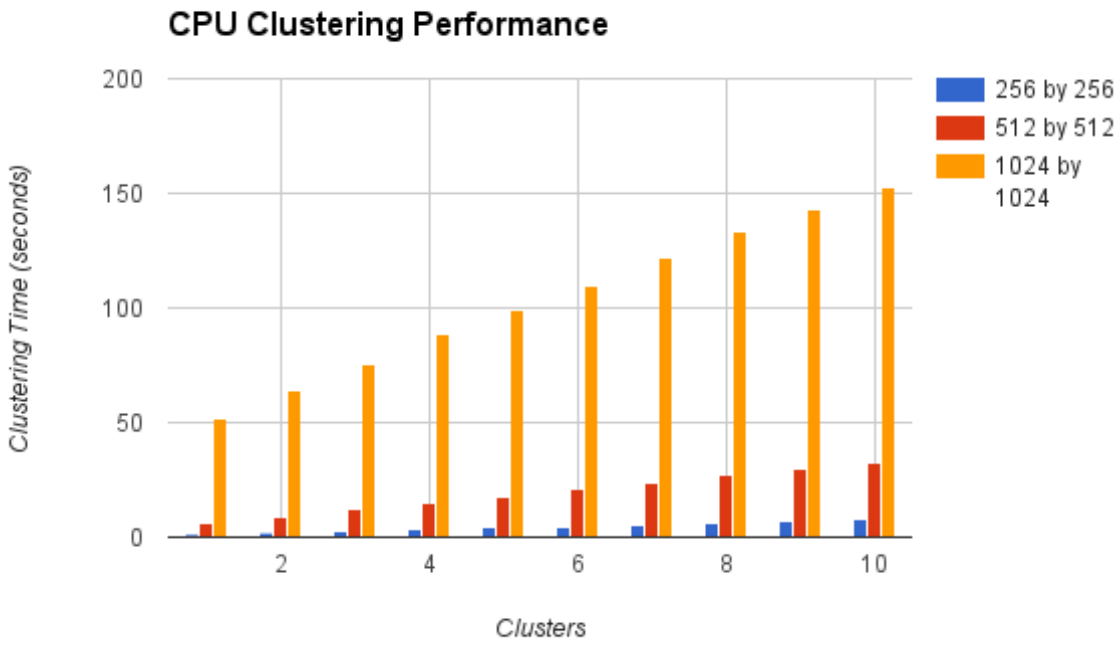

Figure 4.1: Time it takes for the clustering process to complete on the CPU depending on the cluster count. The analysis was performed for terrains of size: 256 by 256 (blue), 512 by 512 (red) and 1024 by 1024 (orange).

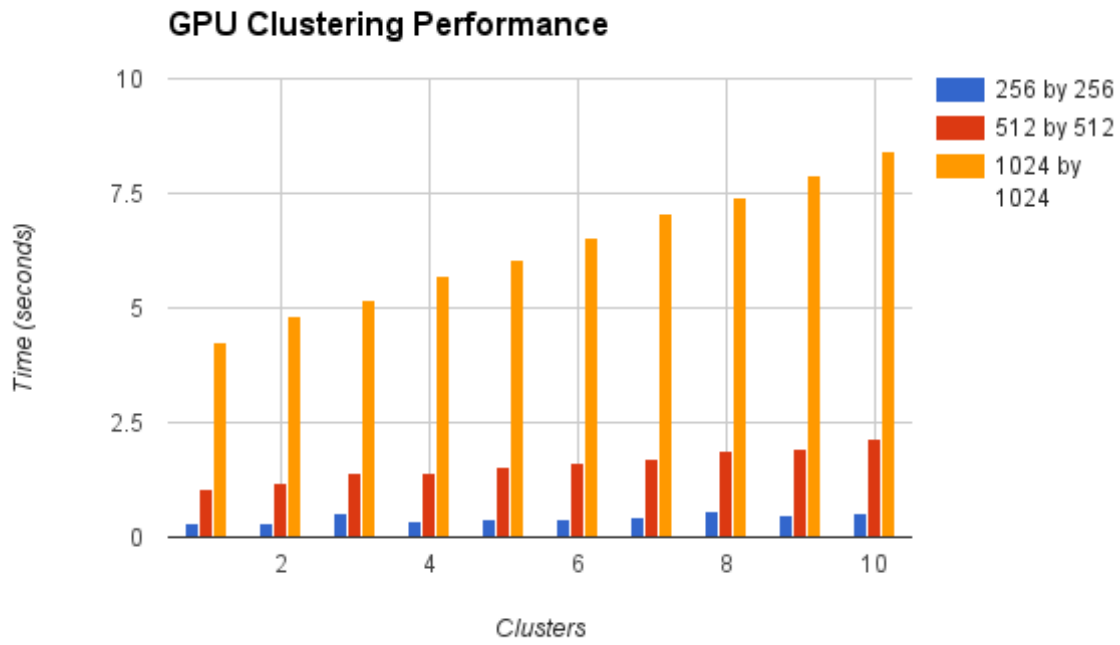

Figure 4.2: Time it takes for the clustering process to complete on the GPU depending on the cluster count. The analysis was performed for terrains of size: 256 by 256 (blue), 512 by 512 (red) and 1024 by 1024 (orange).

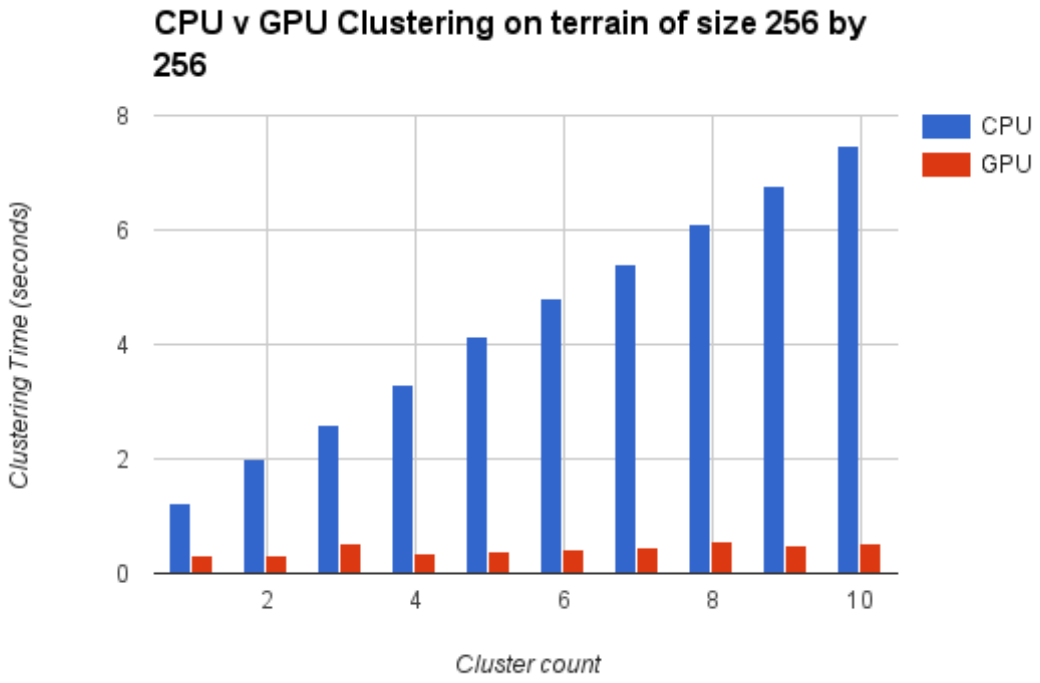

Figure 4.3: Comparison of CPU (blue) and GPU (red) clustering times on a 256 by 256 terrain.

4.3.3 GPU Speed-up

Figures **??**, **??** and **??** compare the CPU and GPU clustering processing time for square terrains of size 256, 512 and 1024 respectively. These graphics show that the GPU greatly outperforms the CPU, irrespective of terrain size and cluster count. Also visible in these graphics is the increased sensitivity to the cluster count of the CPU implementation where the clustering time increases much more rapidly.

As well as confirming the increased sensitivity to cluster count of the CPU implementation, figure **??** which plots the GPU speed-up for each terrain size and cluster count, shows that this speed-up also increases with terrain size.

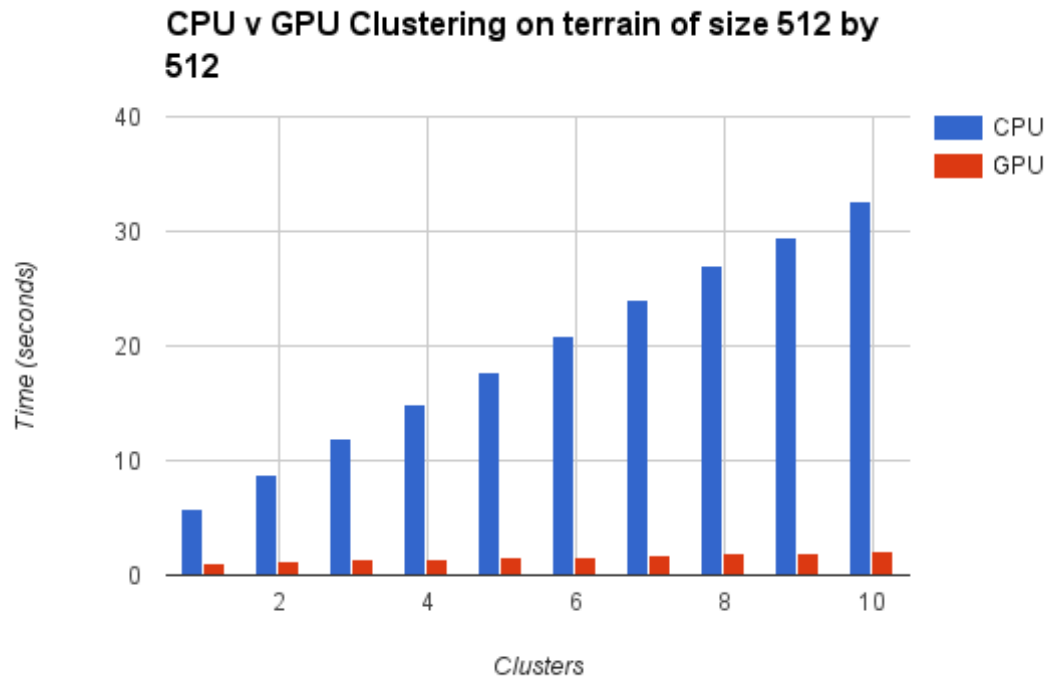

Figure 4.4: Comparison of CPU (blue) and GPU (red) clustering times on a 512 by 512 terrain.

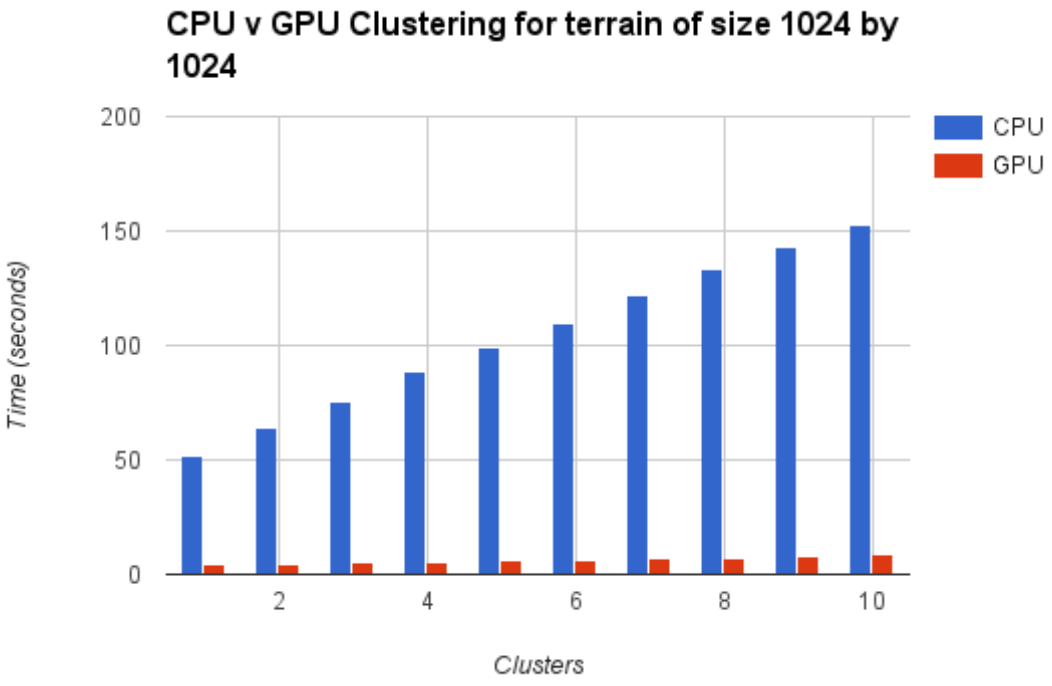

Figure 4.5: Comparison of CPU (blue) and GPU (red) clustering times on a 1024 by 1024 terrain.

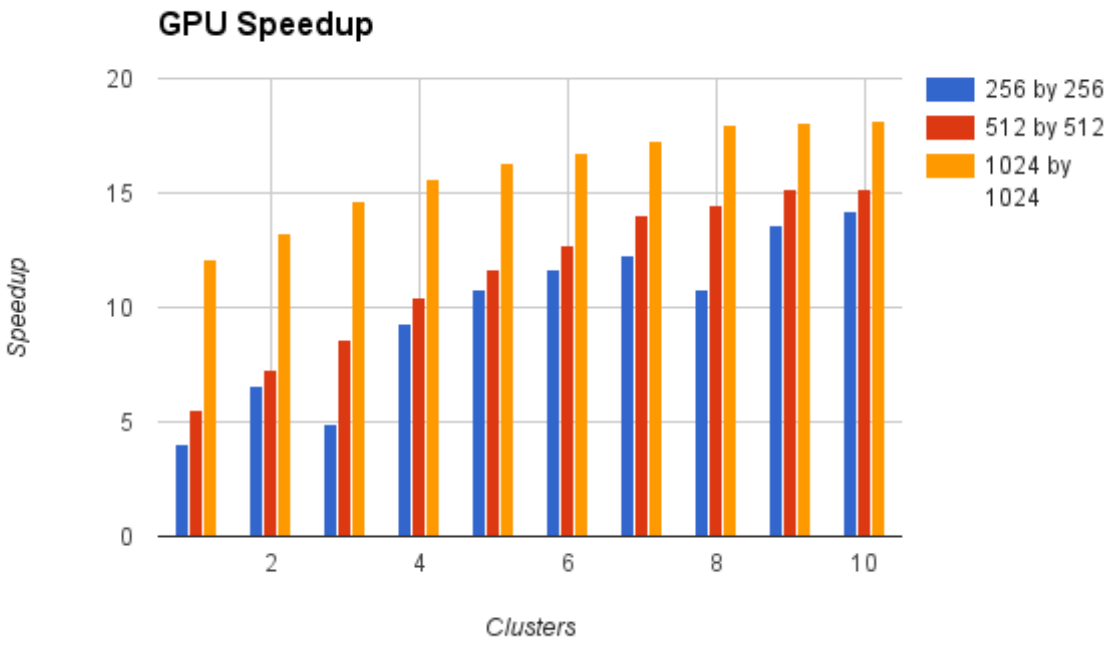

Figure 4.6: Calculated clustering speed-up of the GPU implementation compared to the CPU implementation for square terrains of size 256 (blue), 512 (red) and 1024 (yellow).

Bibliography

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

[aEL99] a.a. Efros and T K Leung. Texture synthesis by non-parametric sampling. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 2:1033 – 1038, 1999.

[BA05] F Belhadj and Pierre Audibert. Modeling landscapes with ridges and rivers: bottom up approach. *Proceedings of the 3rd international conference ...*, 1:1–4, 2005.

[BKK97] F. S. Berezovskava, G. P. Karev, and O. S. Kisliuk. A fractal approach to computer-analytical modelling of tree crowns. pages 323–327, 1997.

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

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

[DC02] Oliver Deussen and Carsten Colditz. Interactive visualization of complex plant ecosystems. *Proceedings of the conference on Visualization '02 (VIS '02)*, pages 219–226, 2002.

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

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

[DP10] Jonathon Doran and Ian Parberry. Controlled Procedural Terrain Generation Using Software Agents. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):111–119, June 2010.

[EC15] Arnaud Emilien and Marie-Paule Cani. WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds. *Siggraph '15, TO BE PUBLISHED*, 2015.

[Emi14] Arnaud Emilien. Création interactive de monde virtuels. 2014.

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

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

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

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

[GMSe09] James Gain, Patrick Marais, and Wolfgang Straß er. Terrain sketching. *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D '09*, 1(212):1–8, 2009.

[Ham01] Johan Hammes. Modeling of Ecosystems as a Data Source for Real-Time Terrain Rendering. *Framework*, pages 98–111, 2001.

[HLT09] T Hurtut, PE Landes, and J Thollot. Appearance-guided synthesis of element arrangements by example. *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, pages 51–60, 2009.

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

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

[Lew99] Philip Lewis. Three-dimensional plant modelling for remote sensing simulation studies using the Botanical Plant Modelling System. 1999.

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

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

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

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

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

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

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

[SKG+09] Ruben M Smelik, Klaas Jan De Kraker, Saskia A Groenewegen, The Hague, Tim Tutenel, and Rafael Bidarra. A Survey of Procedural Methods for Terrain Modelling . 2009.

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

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

[vBBK08] Ondej Šťava, Bedich Beneš, Matthew Brisbin, and Jaroslav Kivánek. Interactive terrain modeling using hydraulic erosion. *EG CA - EuroGraphics Symposium on Computer Animation*, 2008.

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

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