2001

Storage free terrain simulation

Warren Creemers
Edith Cowan University

Edith Cowan University

Copyright Warning

Storage Free Terrain Simulation

A Thesis Submitted by Warren Creemers.

In Partial Fulfillment of the Requirements for the Award of
Bachelor of Science, Honours (Computer Science)

At the
Faculty of Communications, Health and Science
Edith Cowan University.

August, 2001.

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Use of Thesis

This copy is the property of Edith Cowan University. However the literary rights of the author must also be respected. If any passage from this thesis is quoted or closely paraphrased in a paper or written work prepared by the user, the source of the passage must be acknowledged in the work. If the user desires to publish a paper or written work containing passages copied or closely paraphrased from this thesis, which passages would in total constitute an infringing copy for the purpose of the Copyright Act, he or she must first obtain the written permission of the author to do so.

DECLARATION

I certify that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously written by another person except where due reference is made in the text.

Signature

Date 15/1/2002

ACKNOWLEDGMENTS

I wish to thank Maurice Danaher for his valuable advice.

I would also like to thank Daphne Brosnan for her proofing and comments.

Contents

ABSTRACT

Landscape visualisation is the process of recreating a natural environment and displaying it in an interactive graphical simulation. To do this a terrain is displayed together with accompanying plant life and other objects.

Present landscape visualisation software is capable in theory of displaying very detailed and large landscapes. The software is also in theory capable of simulating environments with thousands if not millions of individually structured plants. In practice though the simulation of such landscapes requires such a large amount of storage space that it is not achievable on personal computers. Even storing small landscapes with a moderate amount plant life can be a major development problem.

The extent of this problem is such that modern simulators almost always exhibit the following limitations.

- When detailed landscapes are stored to the hard disk, the area of terrain covered is usually very small.
- When large terrains are stored to the hard disk the detail used is usually low.
- When detailed plants are used in a landscape only twenty or so plants are created and used over and over again in the landscape.

This work is an original approach to solving the storage space problem that involves not storing any landscape data to the hard disk at all. In this solution, instead of the landscape simulator displaying a landscape that is stored on a hard disk, the landscape simulator displays a landscape that is randomly generated. The landscape is produced on a need-to-know basis, the only landscape that exists in the simulator is the landscape that the user of the simulator can see. If the user's position in the landscape alters then the newly visible areas of landscape are created, and the areas no longer visible are removed from the simulator entirely. Areas of landscape being visited for a second time are always re-created the same way as they were originally created.

1.INTRODUCTION

"If a tree falls in the woods and no one is there to hear it, did it make a noise?"

-a Koans of Zen Buddhism.

This work depicts a new engine for computer games and simulators that involve the visualisation of landscape. The underlying basis of this engine is that anything outside the user's visual vicinity need not exist or be stored in the simulation. The engine depicted in this work is responsible for continuously creating a world for the user to see, replacing the traditional role of a human "level designer"*. The engine works by maintaining only the areas of landscape within the user's visual vicinity. When the user travels through a landscape simulation (using this engine), areas of landscape that enter the user's visual vicinity are created just before the user can see them. Areas of the landscape that leave the user's visual vicinity are destroyed and are no longer a part of the simulation. The engine described in this work maintains a constant environment by always recreating the same landscape each time a particular part of the environment enters the user's visual vicinity.

The visualisation of landscape involves two parts, the visualisation of terrain and geographical information and the visualisation of plants and other non-geographical items. Landscape visualisation is the process of continuously rendering terrain and non-geographical information to the screen in a real-time manner. Many games and real-time graphical simulations are set in outdoor landscapes and thus employ landscape visualisation.

Storage space is the major limiting factor in the visualisation of both terrain and non-terrain data. Although advancements in computer hardware, combined with recent research in the field of terrain visualisation, now allow for detailed terrains to be drawn, most users cannot afford to store the datasets required to take full advantage of the new technology. This work will remove the need for large amounts of storage space in landscape visualisation software. Removing the need for large amounts of storage space will allow visualisation of larger and more detailed landscapes than was previously possible on systems with limited storage capacity.

* A "level designer" is a person employed to design the layout of environments in computer games and simulations.

1.1 The state of technology and historical work

Landscape simulators currently follow the basic flow chart in Figure 1.1.
A landscape is created, typically by a human designer, and stored as part of the datasets used by a landscape simulator. There are two datasets in a landscape simulator, one that represents terrain data and one that handles non-geographical elements. When the simulation is executed the datasets are loaded from the storage device, processed in the simulator and then displayed to the screen.

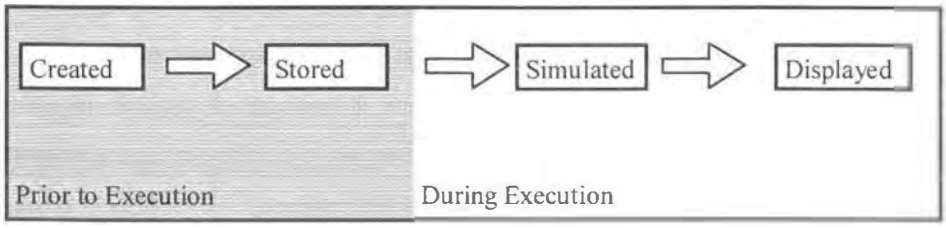


Figure 1.1: Flow chart for landscape data in a landscape simulator.

The process of terrain visualisation typically converts the terrain dataset(s) into a triangular mesh that represents a terrain. The datasets used in terrain visualisation are generally height values sampled at regular grid intervals. These datasets can be converted to a triangle mesh by constructing a lattice in 3D space and using the values from the datasets to displace the intersections of the lattice. It is difficult to render such a mesh to the screen however because of the amount of triangles involved. Modern hardware is not capable of displaying the amount of triangles present in a mesh in this form, if the mesh is to accurately depict a detailed landscape over a reasonable distance.

Past solutions to the high triangle count of terrain meshes involve only rendering objects that are close to the user's position and using a fog effect to hide the missing detail. Another solution is to decrease the resolution of the lattice used for the rendered triangle mesh. This reduction of resolution results in fewer but larger triangles, so the terrain becomes visible for a great distance but is severely lacking in detail. The limitations, caused by the large amount of triangles used in the visualisation stage of landscape simulators are the traditional bottleneck in displaying detailed and large terrains.

Current research has solved the display bottleneck for terrain visualisation. Viewed near ground level most of the triangles in terrain meshes are distant from the user. After

perspective is applied to the rendered image these triangles will only occupy a few pixels on the screen. Lindstrom et al. (1996) used this knowledge to create meshes that involved different triangle sizes. These meshes use smaller triangles near the user's viewpoint where detail is important, and larger triangles at areas distant from the user, where in reality detail would become blurred. By adapting the mesh to be optimised about the user's viewpoint a user could explore a terrain rich in detail and large in size. This technique became known in the industry as continuous level of detail meshes, abbreviated to CLOD.

Lindstrom's method was not perfect as it contained a visual disturbance known as popping. Popping occurs as the user approaches large triangles in the distance. The triangles in Lindstrom's situation are split into multiple triangles and the user can see the sudden increase in detail. This meant that details in the terrain would suddenly appear when a user got close enough. Rottger & Heidrich & Slusallek & Seidel (1998) devised a geomorphing algorithm that removed the effects of popping by detecting sharp changes in the terrain and using more detail to define these areas when viewed from a distance.

With the display bottleneck in terrain visualisation solved, the advancements in computer hardware allow a modern personal computer to display a large detailed landscape with varying plant life. However this new advance brings about a storage bottleneck in landscape visualisation. Currently most modern personal computers do not have the storage space necessary to store a large and detailed terrain. Personal computers also lack the storage space to store large amounts of individual and detailed non-terrain elements such as plant life.

During the last decade there has been little increase in the size of terrains used in simulations. The only increase in terrain sizes is due mainly to the increase in storage space available on storage devices. Currently a simulation's size is dependent on the storage limitations of the computer it runs on. Often the dataset for a detailed landscape of fair size may run into hundreds of megabytes. For extremely large datasets used in detailed simulations of entire planets the storage space is measured in gigabytes, or greater. Typically solutions to storage problems involve extrapolation or prediction of extra detail not stored in the dataset.

This work addresses the limits imposed on landscape visualisation caused by the need to store large datasets representing the landscape to be visualised. The solution presented here allows visualisation of a landscape that is defined procedurally and does not have a dataset. Figure 1.2 shows how the simulation engine presented in this work has no storage step in its execution. The solution presented in this work makes the assumption that the landscape to be visualised is a terrain of fantasy; one that does not exist in real life. The solution is a viewing

system that procedurally generates all the graphics that are to he displayed on a need-to-know basis.

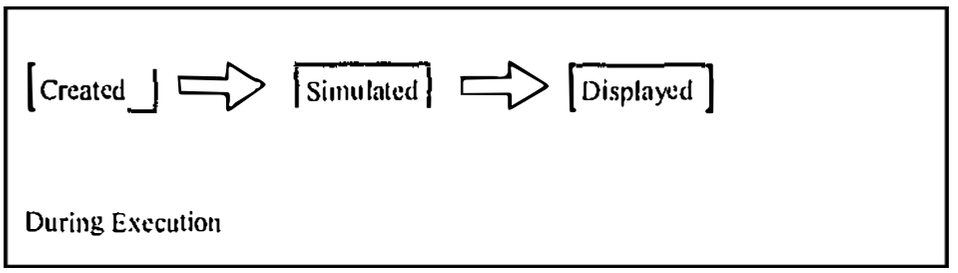


Figure 1.2: Flow chart for a Dynamic Landscape Simulator

The solution presented is only applicable to fictional simulations such as flight simulators and computer games. Geographic information systems cannot benefit from the solution presented since they must visualise terrain that actually exists. This lack of application to real landscapes is not important however because most geographic information systems have no problem dedicating many gigabytes of storage space to a simulation.

The technique of procedurally modelling natural objects applies both to the generation of terrain and non-geographical elements. Fractal math is the most popular approach used to procedurally model natural structures.

Barnsley et al. (1988) presented a collection of techniques to create fractal terrain for landscapes. These techniques are not meant for real-time generation, but are adapted for this purpose.

Lindenmayer (1968) developed a technique to represent and model plant life. This technique produced realistic three-dimensional structures that closely modelled plants and trees. A point of note on this technique is that the algorithms involved can be given different seed values to produce differently structured plants of the same variety.

This work adapts fractal generation techniques for the purpose of real-time modelling. The techniques will be used to procedurally generate a terrain and a population of plants which will combine to form a landscape. Furthermore, the landscape will be maintained on a need-to-know basis. This work uses this approach to maintain a large detailed terrain populated with unique and detailed structures for every tree in the landscape, something almost unheard of in real-time landscape simulations.

1.2 Rationale

When one is considering the impact of removing the storage problem it is necessary to analyse the effect that the employment of the suggested techniques will have on the game or simulation user. In the end it is the user's acceptance of this technology that will determine its success.

Because of the storage problem current terrain simulations lack variety in the graphics displayed on screen. Removing the storage problems associated with landscape visualisation will allow a user to explore a virtual world that contains unprecedented amounts of variation, with detail consistent to modern expectations. It is hoped that the increased landscape size and variation will result in more time being spent exploring the environment before the user gets bored. By using three-dimensional plant life with unique and individual structure the scenery throughout the simulation will be constantly varied. The variation and uniqueness of the plant life will reduce the rate at which the graphics will become familiar to the user.

1.3 Overview of the results achieved

There is a widespread belief that, in landscape visualisation, modelling of terrain and plant life should be carried out prior to the actual visualisation of the landscape. It is a primary goal of this work to show that the concept behind modelling terrains during visualisation is both practical and valuable.

The implementation provided with this document is probably the first terrain visualisation system that utilises real-time procedural modelling of both terrain detail and plant life as its source of information to be visualised. The integration of modern advancements in terrain viewing shows that this process is suitable for modern applications.

2. BACKGROUND INFORMATION

To fully understand the concept of modelling terrain data during visualisation it is important to understand how a landscape simulator stores terrain information. This chapter summarises the "height field grid" method for representing terrain data in a computer.

2.1 Height Field Grids for landscapes

A height field grid is a two-dimensional array composed of sampled terrain heights. To construct a height field grid a terrain is divided up into a grid, and at each grid point the elevation of the terrain is stored into a corresponding element of the array. This is shown graphically in Figure 2.1.

Dataset. (height fields)

6	3	4	5	7
6	2	3.5	4.5	6.5
1	0	0	0.5	2
1.5	0.5	2	4	6
1	0	-1	3	5

Terrain

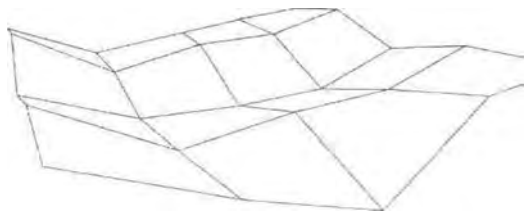


Figure 2.1: Height field data

Representing the data in the height field as a lattice mesh produces a surface that is suitable for rendering.

Though the height field storage format itself is used often, the plain vanilla surfaces generated by this technique are no longer used today because they produce too many triangles. Past simulations, like the one in Figure 2.2, would use these surfaces in a manner where the grid points were far apart and the user was not allowed to see distant elements of the surface.

Figure 2.2: A 1987 simulation using height field surfaces.

In many applications it may be useful to alter the resolution of a terrain mesh. Fortunately this is a simple technique, done by constructing a terrain mesh in such a way that it does not use all the data available from its dataset. Terrain meshes are lowered in detail by creating a mesh where elevations on the lattice represent an average of more than one elevation in the original dataset. The same technique is commonly used to resize a bitmap or picture. Figure 2.3 shows the result of reduced resolution in height field meshes.

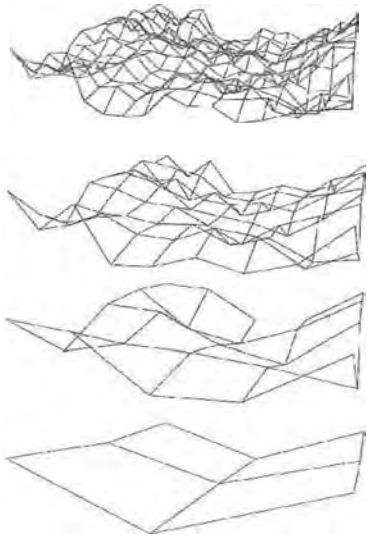


Figure 2.3: A terrain mesh with progressively lowered resolutions:

Different sampling resolutions is a simple way of allowing a terrain to be visualised at different detail levels. Importantly, this technique allows a variety of users to select a detail level appropriate to their hardware, and visualisation needs.

Altering sampling resolutions is not considered an effective solution to large triangle counts since it does not deal with the trade-off between the detail of the mesh and the system performance. This approach only allows users to choose how much detail they will trade for performance.

3. REVIEW OF THE STATE OF THE ART

This work brings together four different areas of research, namely:

- Terrain generation, using fractal math
- Continuous Level of Detail (CLOD) Meshes
- Plant generation, using fractal math
- Landscape simulation

The combination of techniques in the areas of terrain and plant generation contributes to an area of research called landscape simulation, also called environment generation. This work optimises the process of terrain generation to allow for landscapes to be created during a simulation's execution.

3.1 Terrain Generation Using Fractal Math

Techniques for generation of terrain have been developed for a while, although work in this area is often aimed at producing particular kinds of landscapes. Algorithms presented by Barnsley & Jacquin & Malassent &, Reuter & Sloan (1988) produce accurate mountainous environments. Kelly & Malin & Nielson (1988) devised a method to incorporate accurate streams and waterways, faithful to the principals of erosion. The general techniques presented by Mandelbrot (1977) can be adapted for environments such as valleys, plains and seabeds. Figure 3.1 shows a typical computer rendered landscape generated using fractal math.

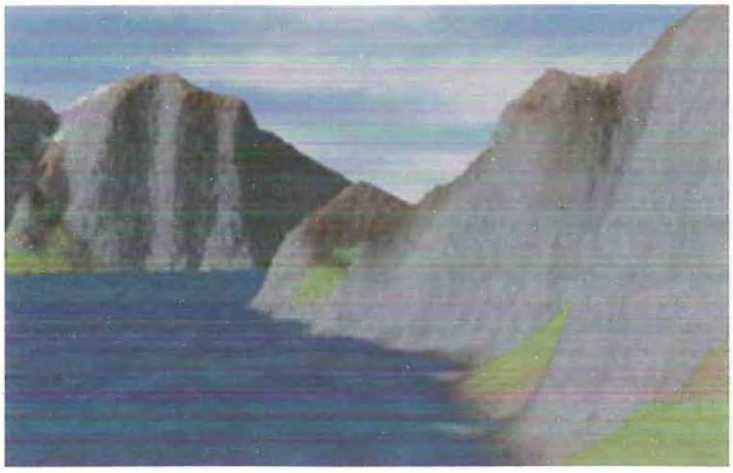


Figure 3.1: A Fractal Terrain Generated by a Popular Landscape Tool Vista Pro

More recently techniques known as multi-fractals have been developed to model more realistic environments. Multi-fractals create landscapes with different kinds of terrains mixed together. Multi-fractals achieve this mixing by applying different existing techniques to different parts of a terrain where necessary. Ebert & Musgrave & Peachey & Perlin & Worley (1998) present valuable information on the use and implementation of multi-fractals.

Though a detailed terrain is easily generated, it is expensive to store a large area of detailed terrain to hard disk. For this reason usage of such terrains is highly restricted in graphical simulations.

3.2 Continuous Level of Detail (CLOD) Meshes

View dependent Level Of Detail (LOD) meshes are of great focus in current real-time graphics research. These meshes can have different levels of detail present at different parts of the mesh, allowing application designers to choose where the detail will be present.

A technique whereby LOD meshes are continuously reorganised to suit a user's viewpoint was pioneered by Lindstrom et al. (1996), and is currently widely practiced in industry. Lindstrom's technique is known as the Continuous Level of Detail Mesh (CLOD) and the detail of this mesh can be dynamically changed at different points on the mesh.

The advantage of CLOD meshes is the reduction in the amount of triangles that need to be drawn to the screen due to the fact that larger (thus fewer) triangles are used in areas where detail is not visible to the user. The reduction in the amount of triangles accelerates the real-

time performance of these meshes, and importantly, the scene from the user's viewpoint will not have lost any visual quality.

In Figure 3.2a a CLOD mesh has been adapted so that the detail is clustered about a point near the centre. A user positioned in the centre of the cluster would see consistent image quality. This consistent quality occurs because the larger and more distant elements in the grid will appear very small. Figure 3.2b gives an example of how large distant triangles achieve consistent image quality. In this image the darker triangles are actually twice the size of the lighter triangles, but because of perspective they appear to be the same size. Rottger & Heidrich & Slusallek & Seidel (1998) and Hoppe (2000) present further work in highly developed CLOD meshes.

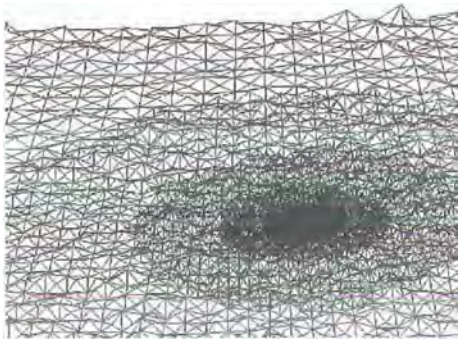


Figure 3.2a: A CLOD Mesh

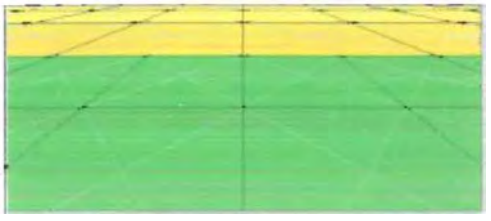


Figure 3.2b: A user's perspective of a CLOD Mesh

3.3 Plant Generation Using Fractal Math

The techniques for the generation of trees and plant life are well documented. It is not the author's intention to increase the functionality of previous algorithms in this field. The adaptation of existing plant synthesis research for this work is concerned with speeding up the existing algorithms, with the objective of applying them to a real-time system.

Most of the work available in the area of fractal generated plants was published in the late 1980's. Due to the time spent in developing fractal plant generation techniques most of the techniques are well established. Foley & Dam & Feiner & Hughes (1990) present a grammatical model of plant structure that allows definition of the structure of a plant using parallel graph grammar languages.

The growth models for the plants in this work will be based on the pioneering work presented by Reffe & Edelin & Francon & Jaeger & Puech (1988). There are other works that use non-

fractal techniques for plant generation, however fractal methods are the most common approach used for the modelling of plants in computer graphics. In order to create a tree structure we employ a recursive definition of a branch:

Branch *= line from point a to point b*
Branch *optionally creates n branches with an origin on line between points a and b*

Because there are multiple branches created on each branch this recursive technique is considered to be self-similar. It is the self-similar property of the recursive plant generation process that makes the process fractal in nature. Figure 3.3 shows the result of applying six recursions of this technique.

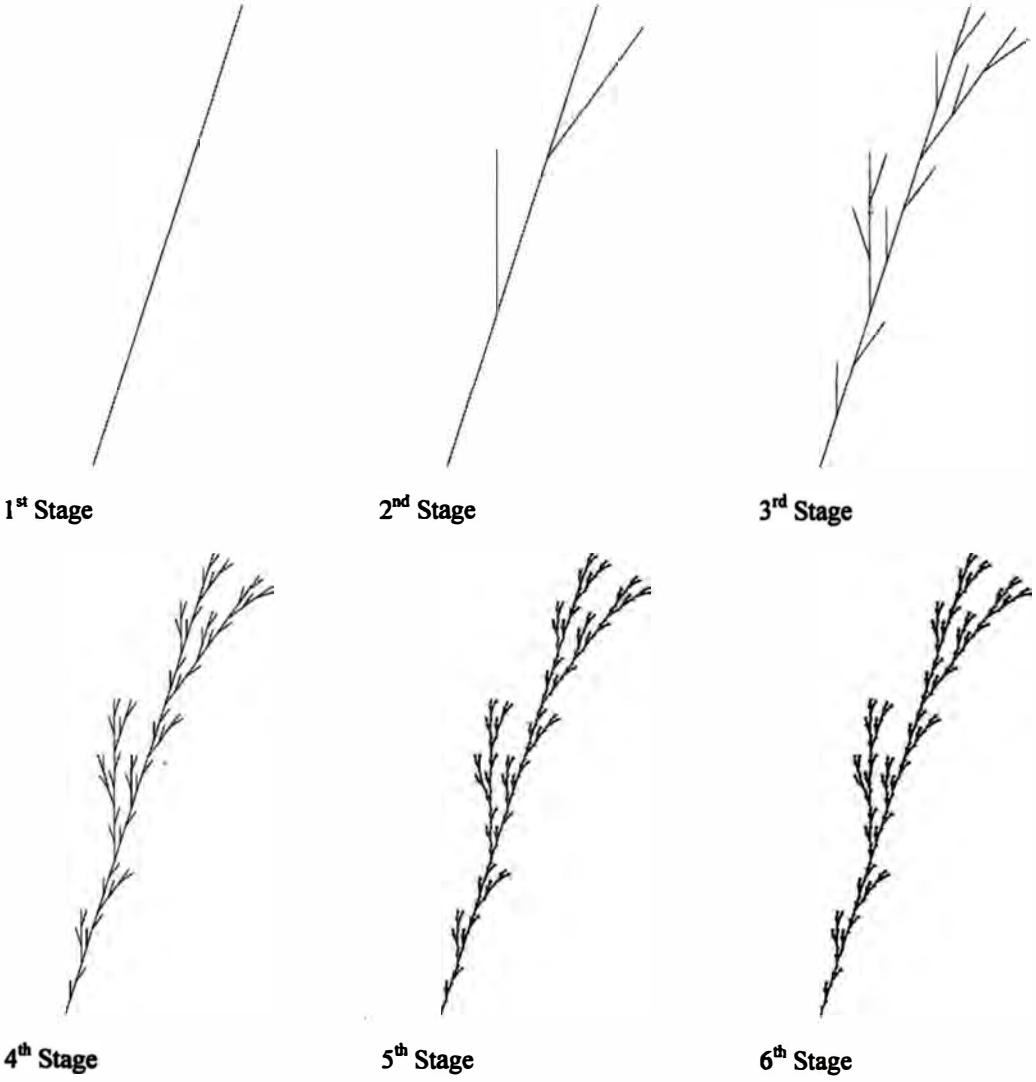


Figure 3.3: The process of fractal plant life generation

Each recursive stage refines the image, but each recursive stage is exponentially more time/resource consuming.

Traditionally fractal plant synthesis is not a process used in real-time simulations. Simulations utilising plant synthesis generally pre-create plants prior to the level or simulation loading. Many simulations use plant meshes originally created using fractal techniques that are stored on the hard disk ready for use.

Various researchers have studied the rules of plant topology and created complicated algorithms that produce highly realistic results. The author directs the interested reader to Reffye & Edelin & Francon & Jaeger & Puech (1988, 151-158).

3.4 Landscape Simulation

Landscape simulation is the process of concurrently simulating terrain and non-terrain objects such as plants. A landscape simulator is generally an interactive program that allows movement of a user viewpoint through a landscape. The landscape simulator inherits the storage problems associated with terrain and plant generation. A scene like the one in Figure 3.4 demonstrates the detail that can be achieved from the use of fractal techniques. This scene stored in polygon form would easily require over a hundred megabytes of storage.

Figure 3.4: "Sunset Valley" by Sam Bowling, 3D Nature, LLC

The results of fractal synthesis techniques for terrain and plant life can be very detailed and lifelike, which has led to extensive use of fractal techniques in professionally rendered graphics. Combining a synthesised terrain with population of synthesised plant life creates a fractal landscape. To add realism the population of plant life is scattered around the landscape in a realistic fashion using the statistics of plant/ species distribution. Further realism is added by changing the texture that is used for different areas of ground, eg snow, rocks, grass, sand and dirt. Adding sky, fog and sunlight effects completes the scene.

4. PROBLEM STATEMENT

The aim of this work is to create a virtual computer environment rich in detail and enormous in size. The challenge is to deliver a solution that will not require large amounts of storage space and be functional on a personal computer.

The objectives of the proposed simulator are that the environment:

- be so large in size that a user would be unlikely to explore an entire environment.
- be so rich in detail that the human eye would survey the scenes of this world much like it would a photograph of a real landscape.
- contain a large degree of variation derived from the use of unique plant life.

These objectives will allow for simulation of entire planets.

The basis of the proposed engine is to implen .. virtual environment where the details beyond what the user can see are not maintaineu in memory. If a user travels through this environment the computer will use the user's position to decide what the user will sec. If the user returns to an area in this world that has already been visited, the computer will produce the same landscape that was previously shown to the user.

4.1 The Problem being addressed

The problem being addressed concerns managing and creating landscapes for computer simulations and games so as to achieve minimal storage requirements. The reason for rethinking the management of simulator environments is because current management techniques require too much memory to allow home users to have large detailed and varied landscapes. The game and simulation industry is constantly aiming to use landscapes that are larger, have higher detail and are more varied. Typically there is a trade off between the extent to which these elements are used in a computer-simulated environment, and the storage space required by the simulation. The objective of this work is to create a landscape simulator that removes the memory restrictions affecting the size and detail of terrain and the variation

and detail of plants. As shown in Figure 4.1 the major objectives of this work are to increase the size, detail and variation present in real-time landscape simulators. The diagram also shows that storage space is the common limiting factor in all of these objectives. A major aim of this work is to remove the trade-off between the qualitative objectives (size, detail and variation) and storage space.

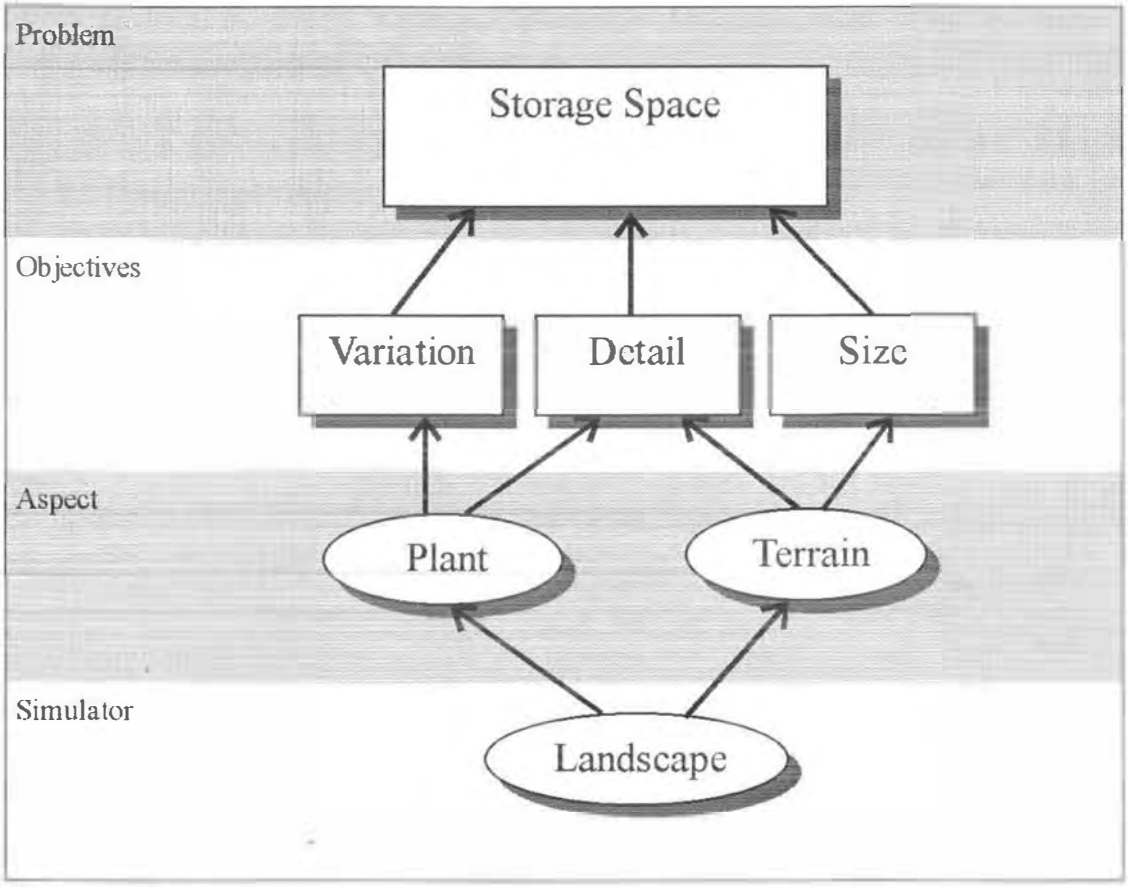


Figure 4.1: The impact of storage space

4.1.1 The problem of Landscape Size

Increasing the size of the terrain in a landscape simulation increases memory requirements. It is because of limited memory resources that common flight simulators typically have a terrain size limited to the size of a cityscape (30 – 60 km). This terrain size may still be considered large, but flight simulators sacrifice a lot of detail in the terrain to achieve this. In other words, a low-resolution terrain is used to compensate for the increased memory requirements of the larger area covered.

In first person point of view games such as "Quake" (1996), environment size is limited because of both the amount of triangles that can be stored and the amount of time needed to

design the level. Typically this type of game is limited to 1-5 buildings per level. Even if it were possible to store maps of significantly larger size, it is probable that the years it would take a human to design the map would be prohibitive.

4.1.2 The problem of Landscape Detail

The problems with increasing detail in landscape simulations relate to the amount of storage space required to increase the amount of detail present. This storage problem affects both the terrain and non-geographical visualisation systems.

4.1.2.1 Penalties of Terrain Detail

The resolution of a terrain is the number of points used in a height field to define an area, also expressed as the detail of the terrain. The resolution can be represented in points per kilometre. Figure 4.2 illustrates the effect of detail being increased by altering the resolution of a terrain.

Figure 4.2: Three terrain meshes each with different resolutions

The amount of memory used to store a terrain mesh is exponentially proportional to the terrain's resolution. Table 4.3 shows the increased memory requirements incurred by use of different terrain details in Figure 4.2.

Table 4.3: Memory requirement of terrain meshes in Figure 4.2

Terrain	Level of detail	Memory required (assuming 1 byte height fields)
A	1x	25 bytes
B	2x	81 bytes
C	3x	289 bytes

Figure 4.4 shows the relationship between increase in terrain detail and storage requirements.

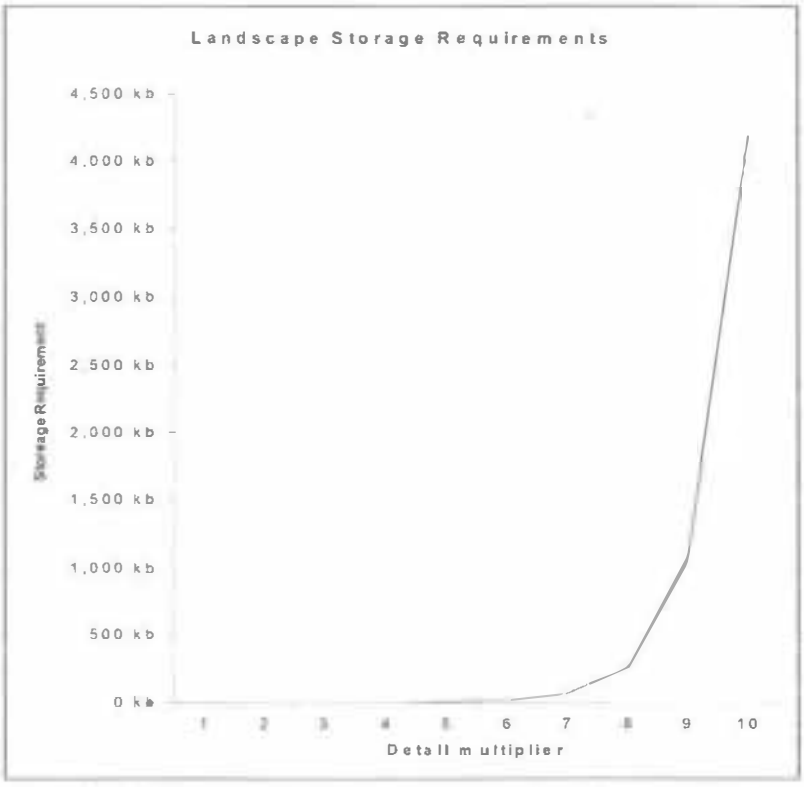


Figure 4.4: relationship between terrain detail and storage space

It is because of the relationship shown in Figure 4.2 that high quality terrain simulators typically require 50mb of memory or more to store landscapes that are 10km wide.

4.1.2.2 The problem of Variance in Non-Geographical Landscape Elements

There is a large storage problem concerned with non-terrain elements of an environment. Each rock, tree and other such elements in an environment require a bitmap or texture. If there are 200 trees in a scene it is not practical to create 200 bitmaps. It is because of this storage limitation that such items typically share a common bitmap. Commonly landscape simulators might only have two or three trees stored in memory and repeat them throughout the simulation. This repetition detracts greatly from the realism of the scene. A primary goal of this work, graphical variation of landscapes, is achieved by allowing individual plant lives to exist in a landscape environment.

There are now simulations that are using 3D meshes for their plants and rocks. This advance amplifies the problem of storage space since we now must store both a mesh and a texture for each tree.

4.2 Significance of the Problem

The vast majority of landscape simulators available to the consumer market deliver less detail and smaller environment size than they could. These limits of detail and size are brought about by storage space constraints on modern personal computers. The implementation of techniques presented in this study will remove the current limitations on the visualisation of landscapes imposed by the lack of storage space available.

4.2.1 Impact of the Work

The gaming community now stands at the edge of a new revolution in gaming technology, because the continuous level of detail engines, currently being developed, offer the possibility to display larger amounts of detailed terrain data on the screen.

Major gaming companies are now developing the CLOD engines that have been published in recent academic papers. Until recently most commercial CLOD engines were still under development. It is a goal of the gaming community that one day, soon, CLOD engines will allow a user to climb to the top of a skyscraper and look out over a city which can be explored. A major problem with this possible environment size is, if it were possible to render an entire city to screen, how will the map for an entire city of buildings be stored? Even if it was possible to store an entire city of buildings it is unlikely that a level designer would be willing to sit down for five years to create such a city. The work presented in this paper is designed to be a possible foundation for addressing these problems.

4.2.2 Impact of the Work on the End User

"Twenty years from now you will be more disappointed by the things you didn't do than by the ones you did do. So throw off the bowlines. Sail away from the safe harbour. Catch the trade winds in your sails. Explore. Dream. Discover."

-Samuel L. Clemens [author of Huckleberry Finn]

The author has taken the view that many computer games currently on the market provide a new means of exploration. This exploration occurs as users move through areas in the simulated environment for the first time. Computer games offer many of the same motivations that drive explorers such as the ability to find new areas of an environment and interesting environmental features that are worthwhile finding. There are however some motivational qualities missing:

- The feeling that an area is so large it could be explored forever.
- The aspect of there always being new things to see.
- The feeling that you are the first person ever to see some part of the planet.

It is hoped that this work will provide a method for including these missing qualities, and hence have a quick appeal to users.

5. METHODOLOGY OF CONTINUOUS TERRAIN GENERATION AND VISUALISATION

In order to create a real-time landscape simulator with dynamic non-stored graphics the terrain in the landscape simulator must be spontaneously created and never stored on disk. Furthermore the terrain must be created only around where the user is positioned. If we are to maintain a terrain around a user's position, it is common to create the terrain in small blocks which join together to form the terrain the user sees. These blocks are referred to as terrain pages. The advantage of these pages is that areas of terrain can be added or removed from the landscape without other parts of the landscape having to be recreated. Controlling the creation and removal of terrain pages is known as terrain page management. Terrain page management allows terrain local to the user's viewpoint to be maintained and updated when there is a change in the user's position on the landscape.

Terrain page management is crucial to the performance of the landscape simulator. Section 5.2 addresses the issue of how to create an efficient, high performance page manager. It is also necessary to create the terrain information for each page. The work presented by Magnenat-Thalmann & Thalmann (1985) is adapted for this purpose. If the terrain generation algorithm is seeded with the pages position on the terrain, it is possible to recreate pages the same way each time they are visited. By utilising this technique it is not necessary to permanently store terrain information for a terrain page, since the user is assured of seeing the same terrain information every time he/she visits the page.

The last part of this chapter addresses the issue of visualising the terrain generated by the terrain page manager. The terrain page management and terrain generation algorithms are integrated with an algorithm that creates level of detail meshes. It is via the integration of continuous level of detail meshes that continuous terrain generation is shown to be a practical and modern approach to terrain visualisation. The details of the CLOD algorithm presented in the implementation are presented in section 5.5.

5.1 Terrain Generation Algorithm

The goal of the terrain generation algorithm is to create height elevation data consistent with what the actual terrain is expected to look like. The terrain generation algorithm should be able to generate a piece of terrain given a seed value, environment parameters and a location. Furthermore given the same parameters the terrain generation algorithm should give the same resulting terrain each time it is run. Because the terrain generation algorithm is capable of recreating terrain identically, areas revisited by a user will always appear the same.

Before a terrain generation algorithm can be built, a suitable terrain representation format must be selected. Section 5.1.1 examines available storage formats currently used in geographic information systems, and shows why the Height Field Grid is the most suitable format for this work. Later in section 5.1.1.3, information is presented on the workings and maths of the Height Field Grid storage format.

A terrain generation algorithm must be specially designed to suit the data format used to represent the terrain. The primary evaluation criteria of the terrain generation algorithms are the speed in which it executes and the realism of the terrain that it generates. In Section 5.1.2 the terrain generation algorithm used in the implementation is discussed.

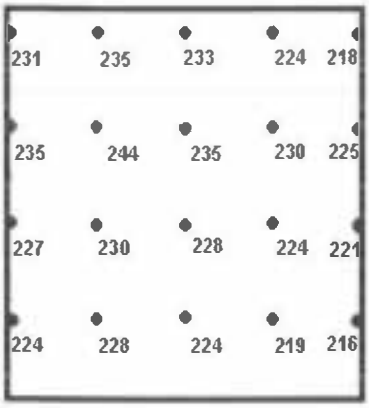
5.1.1 Terrain Representation formats

The problem of an appropriate storage format for terrain data is basically concerned with choosing a format for the storage of data to represent terrains. The data structure chosen will influence such things as memory requirements, processing speeds of the landscape simulation and structural limitations. Structural limitations refers to whether non-extruded features. eg. caves or over-hangs, can be represented. Thus the data format used will determine both the capabilities and efficiency of the landscape simulation.

The three different data structures used to store the representations of terrain in landscapes are:

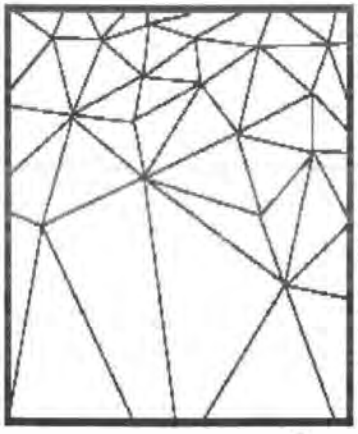
- Height Field Grid (Referred to as a Digital Elevation Matrix (DEM) by those in the geological information systems field)
- Triangular Irregular Network (TIN)
- Digital Contour Line

Figure 5. 1 shows how data in these three formats is stored. The 'National Imagery and Mapping Agency" (2000) present information on these formats, as well as sample terrain datasets. The formats presented here are standard formats adopted by government agencies and the geographic information industry. Any electronic terrain data purchased from data brokers or provided by government services will generally be stored in one of these formats.



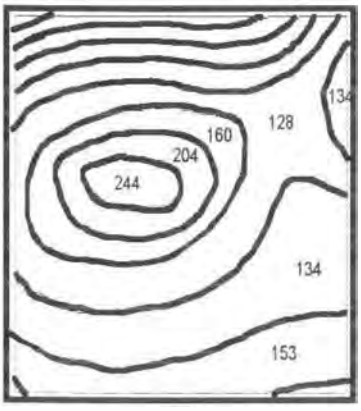
Height Field Grid or Digital Elevation Matrix

The height of the landscape is sampled at regular intervals. The information is stored in a 2 dimensional array. The array positions correspond to positions on the landscape and store the height of the landscape at the corresponding point.



Triangular Irregular Network

This format simply stores all the triangles required to define the landscape.



Digital Contour Line

Stores contour lines and relevant height data.

Figure 5.1: Available terrain data storage formats

5.1.1.1 Storage Considerations

The three formats are evaluated according to their performance and limitations. Bitters (2000) presents information covering this subject in further detail.

Height Field Grid

The Digital Elevation Matrix format has the fastest processing speed of the three possible formats. However the memory requirements of this algorithm are comparatively large. Because the matrix in this format holds only one value at each field there is no way to store non-extruded features such as caves. The algorithms used by a simulator using this format are simple to implement, and thus provide a good development time. Typical applications for this format include:

- Geological surveys
- Interpretation of satellite photography
- Computer games
- Flight simulators

Triangular Irregular Network

The Triangular Irregular Network format can be displayed efficiently, but altering detail in this format is expensive and thus it has a relatively slow processing speed. The memory requirements of this algorithm are the most manageable of the three types because it provides detail appropriate to the relief and jaggedness of the areas represented. This format is the only format to have no structural limitations, this means that it can model any geographical feature. The algorithms used by a simulator using this format are difficult to implement and are prone to bugs in their implementation. Typical applications for this format include:

- Computer games
- Flight simulators

Digital Contour Line

The Digital Contour Line format has by far the slowest processing speed and largest memory requirements of the three possible formats. The format is also not capable of storing non-extruded features such as caves. Implementation of a simulator using this format is difficult and generally this format is avoided. The main advantage of this format is that existing hardcopy contour maps can easily be converted to this format. Typical applications for this format include:

- Weather systems
- Geological information systems
- Geological archiving

When examining the available terrain representation formats for processing speed memory requirements and structural limitations, the height field grid representation format is the most suitable for this work. The primary reason for choosing height field storage format is that algorithms operating on data in this format run faster. As a major difficulty in implementing this work is to get the landscape simulator to run in real-time, the fastest processing speed of this algorithm makes it the first choice, regardless of its disadvantages.

5.1.1.2 The Height Field Storage Model

Sampling the height of a landscape at regular intervals over a rectangular grid is the basis of how data for this format is created. The storage format for data a height field grid is shown in Figure 5.2.

Figure 5.2: The height field data format

The data in Figure 5.2 would be stored in a computer's memory as the array shown in Figure 5.3.

```
{231, 235, 233, 224, 218        (row 1)
 235, 244, 235, 238, 225,       (row 2)
 227, 230, 228, 224 ,221};      (row 3)
```

Figure 5.3: A Height Field Data Array

Given that the distance between the samples in a height grid is known and constant, the position of the elevation data corresponds to its position in an array. To translate an elevation value in an array to a point in a 3D coordinate system the formula in Figure 5.4 can be used.

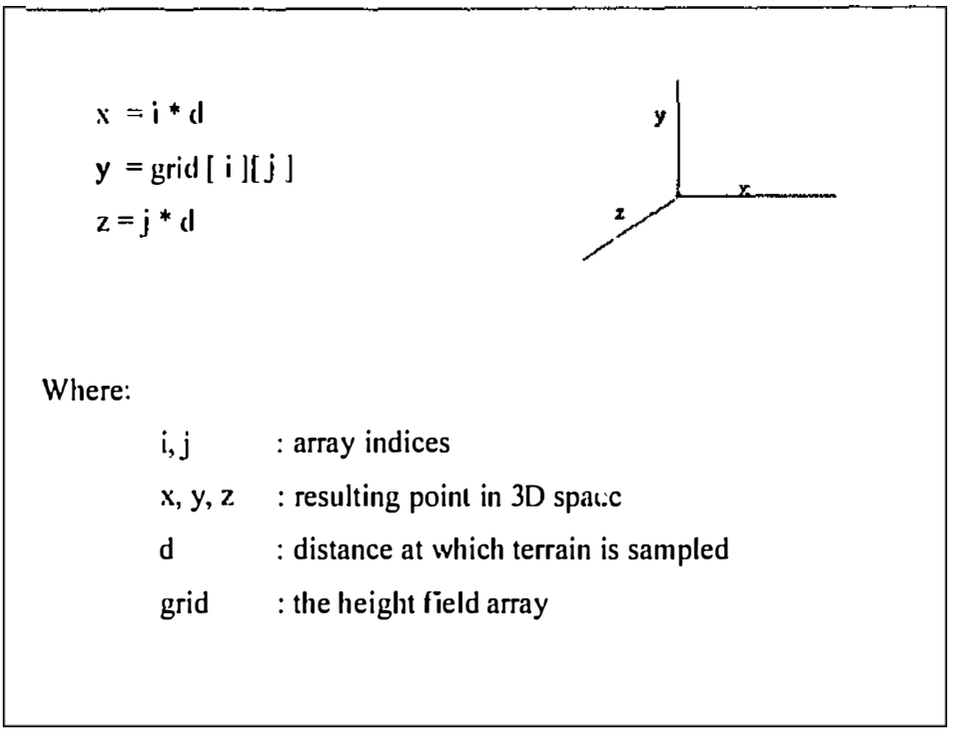


Figure 5.4: calculating 3D coordinates of data in a height field grid

Height field data is commonly derived from satellite imagery. Satellite sensors can produce images that record the height of terrain at regular intervals.

5.1.2 A Terrain Generation Algorithm

Given that we are using the height field representation model for our terrain data, there is a terrain generation algorithm that operates directly on data in this format. This terrain generation algorithm is called midpoint displacement.

5.1.2.1 Midpoint Displacement

A landscape can be considered a fractal surface with nearly infinite surface area. For a description of fractal surfaces see Mandelbrot (1977). From a distance we can approximate a landscape by a line that follows the silhouette of the horizon. As we approach the horizon the line has to deviate for boulders and variances in the ground. As we get even closer the line has to follow pebbles, then grains of sand and so on. Basically anywhere we think we see a straight line, when we look closer we see that it is not actually a straight line. Curved lines that seem to have a constant derivative, upon closer examination are actually composed of lots of smaller lines.

Since, in a landscape, every line is composed of more lines, it is impossible to determine a derivative for any of the lines. With no derivative we cannot determine the surface area of the

terrain. Figure 5.5 shows the increasing detail present in a fractal surface, by using photographs of a real landscape.

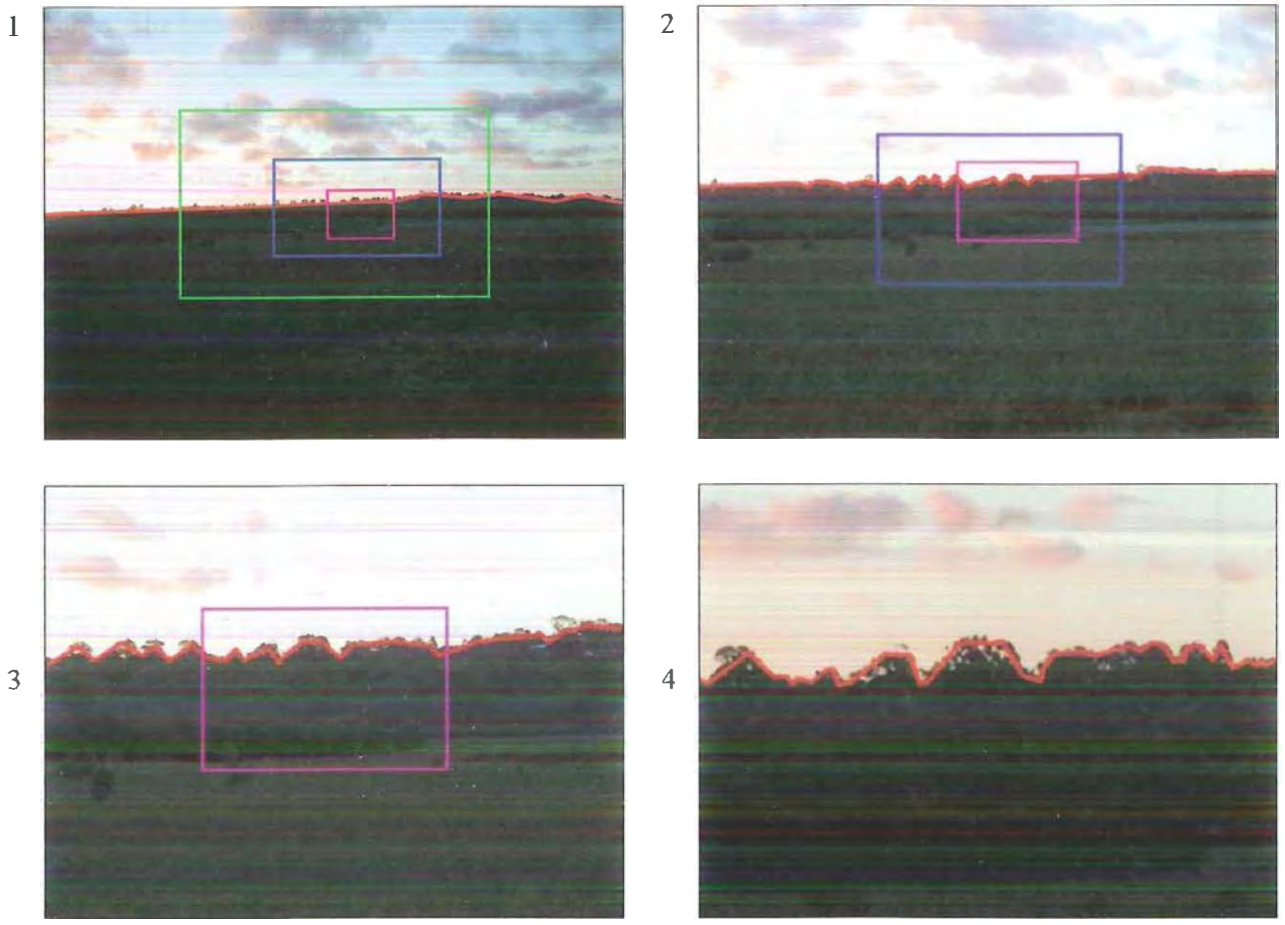


Figure: 5.5: Landscape detail

These photographs in Figure 5.5 were taken with a zoom camera. Note the increasing complication in the horizon as the camera zooms in on the scene.

Terrain is generated (simulated) using a midpoint displacement function. The midpoint displacement method is best demonstrated in two dimensions: Figure 5.6 shows the process of midpoint displacement.

First we take a straight line (1) , and bisect it into 2 lines. The bisection is made in the middle of the first line.

The point where the two new sub-lines connect is displaced, either above or below the midpoint of the line. This gives us two lines in the rough shape of a hill or valley (2).

We repeat this process to the two new lines, creating 4 lines (3). This time though we use a smaller displacement at the midpoints.

This gives us a recursive algorithm that produces a jagged line representing a silhouette of a landscape.

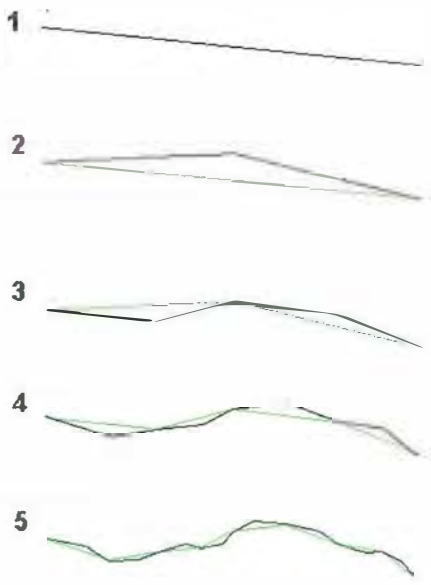


Figure 5.6: Midpoint displacement in two dimensions

Note how the progressively generated lines in Figure 5.6 resemble the progressively magnified horizon lines in the photographs in Figure 5.5. The above phenomenon or style of line is known as fractional Brownian motion. If the displacements of the midpoints are determined with a random number generator we have a mathematical noise known as brown noise.

5.1.2.2 Midpoint Displacement in Three Dimensions

To adapt the Brownian motion technique (in 5.1.2.1) to three dimensions we follow the workings of Barnsley et al. (1988). This work can be applied directly to a height field grid.

We start the process by generating 4 random points at the corners of the height field grid, this provides the data with which we begin our process. This is shown in Figure 5.7.

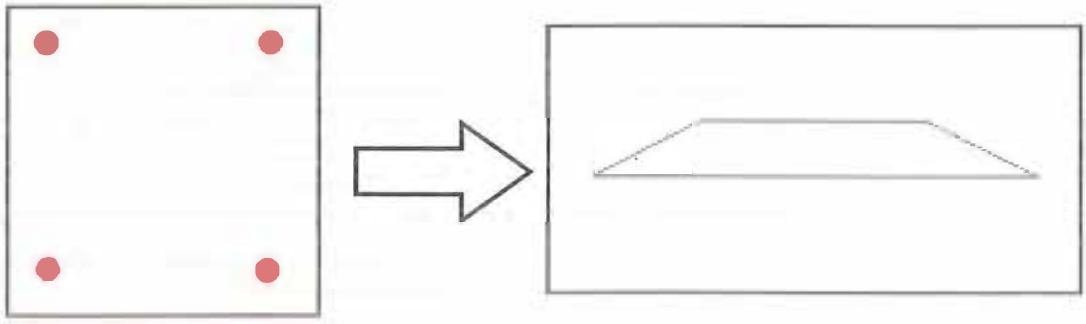


Figure 5.7: Midpoint Displacement, Initial state

First step: Define a height for the point at the centre of the 4 points of data that we have already created, and calculate the height by taking the average height of the four points (above left, above right, below left and below right) around the new point. This is shown in Figure 5.8. To complete this step the new middle point is displaced by a random quantity, making the point either higher or lower than the average of the points around it.

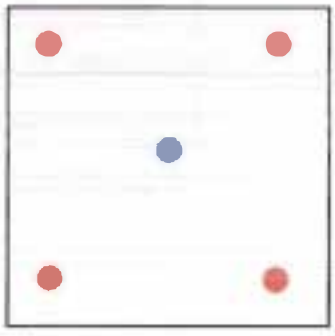


Figure 5.8: Midpoint Displacement, step 1

Second step: Define a height for the points above, below, left and right of the midpoint in step 1. The height of any one these points is determined by averaging the height of the existing points above, below, left and right of the point. This average height is then displaced with a positive or negative random quantity. The result is a new grid of defined points (with Brownian motion), twice the resolution of the previous grid, see Figure 5.9.

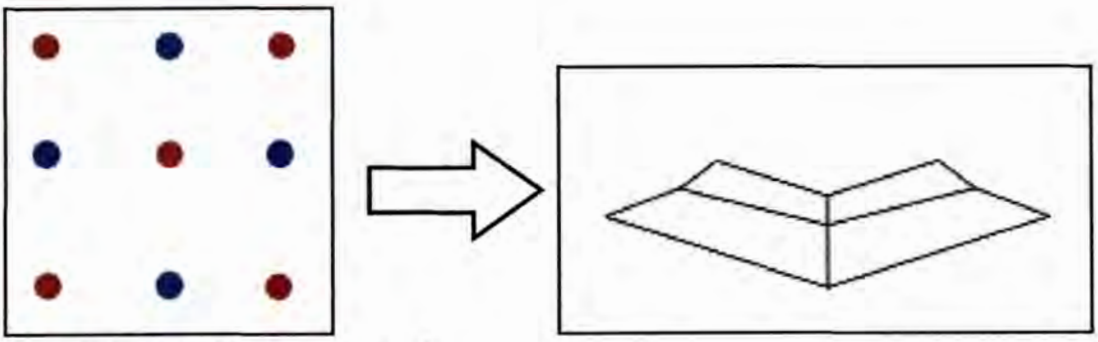


Figure 5.9: Midpoint Displacement, step 2

To increase the resolution of the grid we repeat the displacement process. To achieve Brownian motion in this grid, we must ensure that the average displacement of the midpoints decreases with each recursion. This will give the result shown in Figure 5.10.

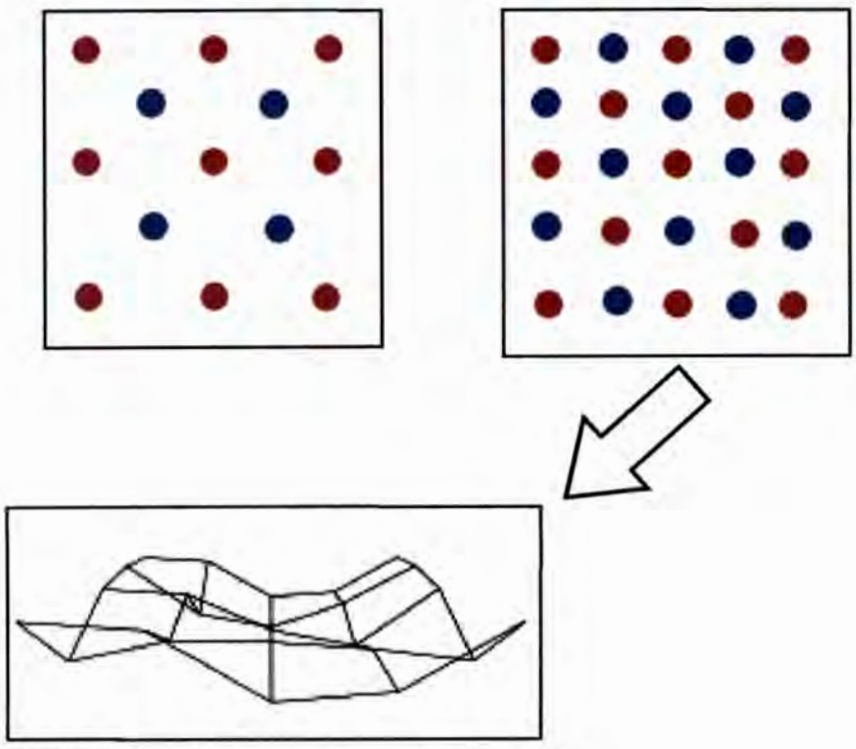


Figure 5.10: Midpoint Displacement, repeating steps 1-2

The result of this procedure produces a three-dimensional fractal surface that closely resembles terrain layout. To ensure this surface resembles a landscape all random variables generated must follow a Gaussian distribution (standard pseudo random number generators produce normally distributed random series). Generally a normally distributed series of random numbers can be transformed into a Gaussian distribution using a transformation function. For information on this technique see Barnsley et al. (1988) (There is also an implementation available in the source code accompanying this thesis).

5.2 Terrain Page Management.

The midpoint displacement algorithm can be used to terraform a page of data in a page management scheme. The objective of page management is to divide a landscape into multiple blocks or pages which tile together to form a visible landscape. By creating individual landscapes for each page, each time the page is created we have the basis for on demand creation of landscapes. The use of pages also increases the speed of landscape creation by reducing the amount of overall recursion present in the creation of a terrain. These pages are created individually using corresponding boundary values and tiled together to form a terrain. The use of terrain pages prevents the midpoint displacement algorithm from having to generate the entire map at once, thus allowing for efficient generation of only the terrain that is needed.

5.2.1 Existing Terrain Page Management Techniques.

A major challenge to the terrain generation algorithm is the maintenance of terrain around the user's viewpoint. This algorithm is intended to only generate the terrain that is around the user's viewpoint and visible to the user. To achieve continuous generation a page management algorithm is constructed. The algorithm presented in section 5.3 is an approach devised by the author especially for this work. For information concerning existing page management algorithms see Eberly (2001).

In the field of page management:

- A page is defined as a height field grid.
- A map is defined as a collection of tessellating pages.
- A submap is a collection of adjoining pages usually representing the user's visual vicinity.

An entire map (a planets worth) of pages is typically too large to be stored in memory. To work around this limitation this work maintains a collection of pages called a submap. This submap is defined with the user's point of view being in the centre of the submap. The submap is responsible for storing all terrain data visible to the user at the current point in time. The CLOD algorithm employed to visualise the terrain needs to operate directly on this submap. The paging algorithm employed for creating and removing of pages in the submap must be highly efficient since it is concerned with frequently paging large amounts of memory. For further discussion on page management concepts see Mauro (2000).

5.2.2 Submap Page Management Algorithm

It is the user's movements that trigger page creation and deletion. With this in mind examination of the page management algorithm will focus on how the algorithm responds to the user's movements.

The submap works by maintaining the user's position in the middle of the map. As the user moves a distance equivalent to one page the submap is adjusted around the user's position. Two events occur to make this possible:

- New pages are created as the user comes within viewing distance of them.
- Pages in the submap are removed as the user moves away from them and they can not be seen any more.

5.2.2.1 A Simple Approach to Submap Page Management.

Figure 5.11 is an example of a simple approach to page management. This example will be used to explain the simple approach to page management:

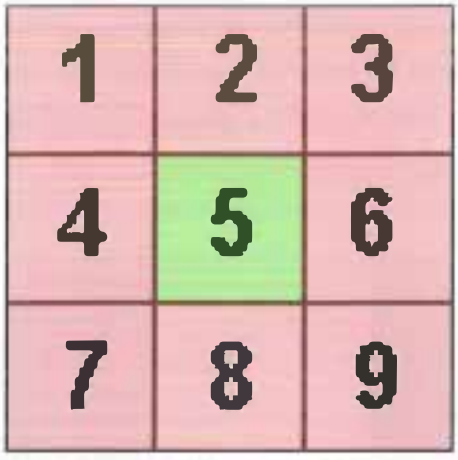
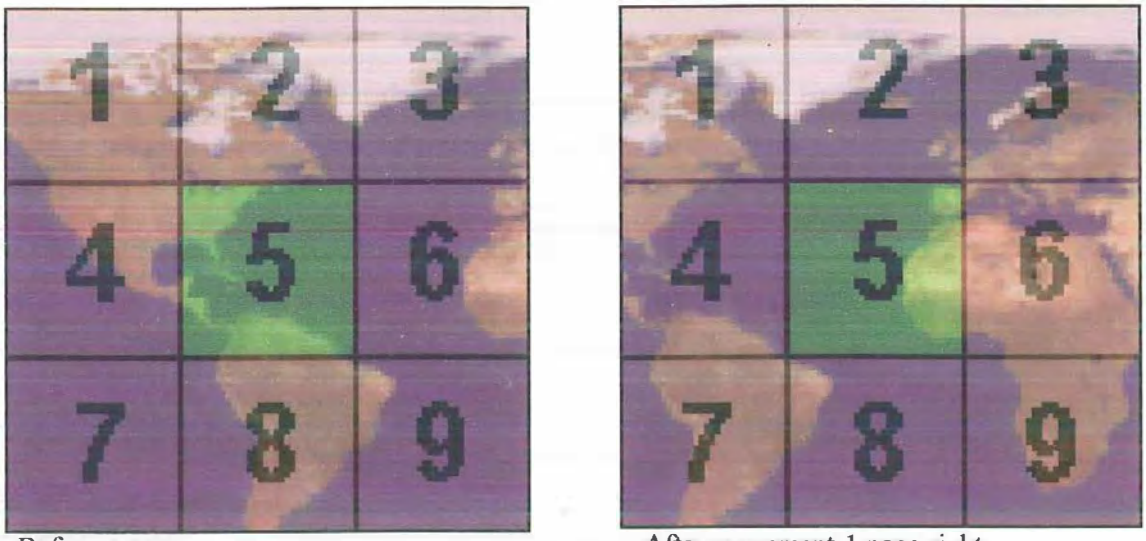


Figure 5.11: A basic 3 by 3 submap.

In the submap, shown in Figure 5.11, the user is situated on page 5. If the user in this submap was to move one page right, the following steps must be taken to ensure the user remains in the centre of the submap:

- pages 1,4,7 are discarded.
- pages 2,5,8 are moved to pages 1,4,7
- pages 3,6,9 are moved to pages 2,5,8
- pages 3,6,9 are created from new map data.

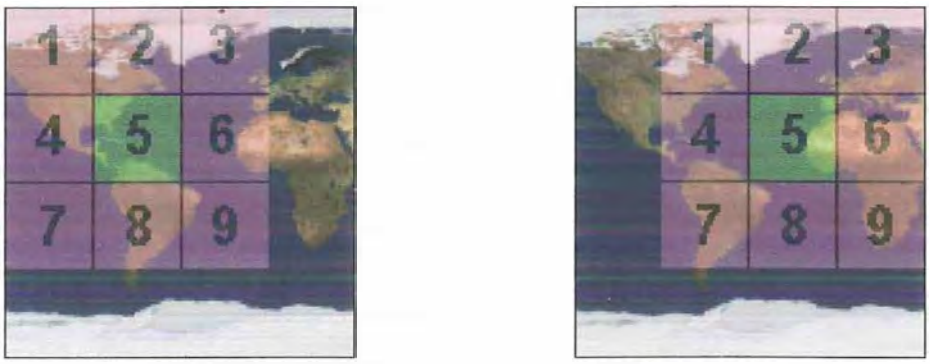
The visual results of this operation as applied to map data for the planet Earth are shown in Figure 5.12.



Before move: After movement 1 page right

Figure 5.12: Simple page management

In the previous example the user's position has remained in the middle of the submap. Since the user in this example always remains in the same position on the submap, it is the position of the submap on the overall map that changes to allow for user movement across the map. The altering position of the submap is shown in Figure 5.13.



Position A Position B

Figure 5.13: Submap Movement in Simple page management

5.2.2.2 Current Techniques for Implementing Simple Page Management

The output submap from the page management routine is what will be visible to the user. The submap will be used as a source of data for the CLOD algorithm. Unfortunately the CLOD engine interferes greatly with implementation of the page management algorithm.

Currently there are two algorithms for terrain page management. Unfortunately neither of these two page management techniques are suitable for this work.

The first algorithm for terrain page management involves

- Implementing a two dimensional array of pointers to pages.
- Creating terrain pages with memory allocation.
- Removing pages by releasing memory.
- Using pointer swapping to shift pages in a submap.

The second algorithm for terrain page management involves:

- Maintaining a submap as a grid of height points.
- Defining a page as a sub-portion of the submap grid.
- Using memory movement routines to relocate memory within the submap in order to shift pages.
- Overwriting memory that is redundant when new pages are created.

On inspection it can be seen that the first type of page management is more efficient because the use of pointers in this technique provides for a highly efficient mechanism to swap pages. The problem with this page management technique is not the efficiency of the technique itself but the fact that the terrain generation algorithm and the CLOD viewing algorithm have problems working with multiple separated pages. Both these algorithms have to constantly figure out on which page the data they are working on exists. Since both the algorithms access a lot of data it becomes highly inefficient to resolve a page memory address before each access.

The second technique of page management produces a single block of memory that is easily accessed by the terrain generation and CLOD viewing algorithm. The problem with this technique is that page swapping is very expensive, due mainly to having to move large portions of memory. In fact, for very large blocks of memory typically used in terrain simulation, the total time of page swapping in this algorithm can be measured in seconds.

This effectively results in the simulation stopping for a few seconds every time page swapping is performed.

Overall the first algorithm, though it is expensive, is least costly in terms of computational efficiency. Since the most efficient form of page management commonly used is possibly not suitable for this work, the author has devised a new approach to terrain page management. This approach is quicker but is also a lot more complicated. The author has termed this technique the offset spherical approach. The main advantage of this approach is that a page in the submap never needs to be moved to a different position in the submap, rather the submap moves around the page.

5.3 The Offset Spherical Approach to Page Management

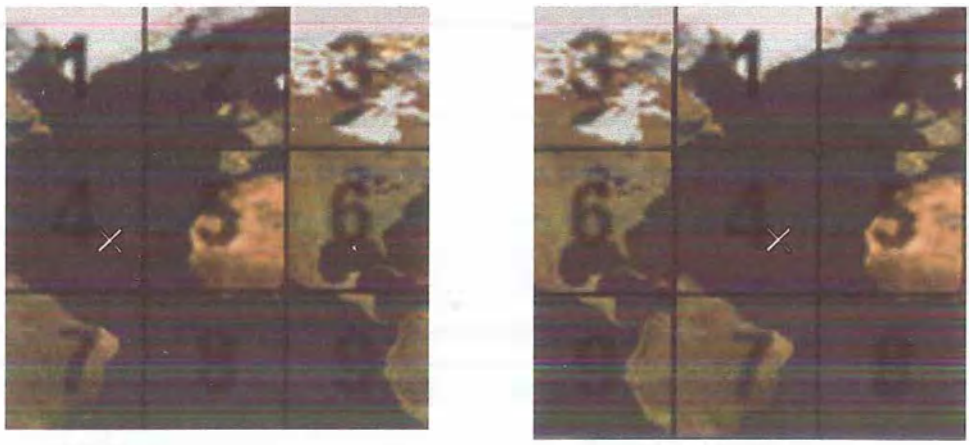
This offset spherical approach is a new method developed by the author for page management. The approach took several months to implement and refine and is a highly original and efficient data structure. With this method the user moves across the submap and does not stay centred in the submap. When the user reaches the edge of the submap the view wraps round to the other side of the map. Figure 5.14 shows an example for a user moving two pages right. In this example the square with a cross is the page which the user is over and the circled pages are pages newly loaded into the submap.

User at initial position *After movement 1 page right.* *After moving another page right.*

Figure 5.14: The offset spherical page management technique

The offset spherical approach to page management is fundamentally different to existing page management techniques. In this technique a user no longer remains in the middle of the submap. A user travelling in a straight line will move though different pages in the submap. When a user encounters the edge of the submap they will reappear on the other side of the

submap, this is shown in the last two steps in Figure 5.14. The next fundamental difference between the offset spherical approach to page management and existing page management techniques concerns how the terrain is stored in the pages. Figure 5.15 shows a close up of the third map in Figure 5.3.1 and it also shows how the map is interpreted as terrain.



a) Submap *b) Rendered terrain*

Figure 5.15: Rendering the offset spherical page management submap

In Figure 5.15 the user's position on the sub map is indicated by the cross. The cross always remains close to the centre of the map displayed on screen, but its position on the submap is arbitrary. Pages 3, 6 and 9 exist on the right of the submap stored in the simulator but are interpreted as existing on the left of the visual landscape. By using the submap in this way the user's view from page 4 actually wraps around the submap.

The reason the offset spherical approach to page management is superior to other traditional methods is because fewer pages are processed when the submap is updated. In Figure 5.14 when a user moves one page right three new pages overwrite three old pages. In traditional page management techniques the same process involves creating three new pages and moving six existing pages.

The terrain generation routines and CLOD viewing routines can be easily modified to wrap around the new submap type. The page swapping in this offset spherical page management technique is highly efficient since no pages have to be moved. It is the efficient page management and simple integration with the other components of a terrain simulation that make this approach to terrain management highly desirable.

5.4 Issues for Creating Consistent Landscapes

This section explains firstly how quasi white noise (see, Knuth (1997)) is used to create terrain pages identically each time they are revisited. Then a technique for the very rough approximation of a pre-defined terrain is discussed. Lastly, a technique is discussed to allow the edge of a terrain submap to line up with pages that are outside the submap

5.4.1 Quasi White Noise Synthesis for Terrain Seeding.

It is necessary for terrain pages to be created identically each time they are created if the user is going to see the same landscape each time he/she revisits an area of a landscape. The midpoint displacement algorithm, in section 5.1.2.2, began with four random variables, one in each corner of the page. These four variables seed the landscape for our map. If we use the same seeds for each page every time they are generated, our pages will look the same each time we visit them.

We could store a bump-map (as greyscale bitmap), so that each pixel corresponds to a corner point on the page. The result of this is that the landscape is an enhanced version of the bump map. This technique is effective, but it doesn't work for large terrains, such as the Earth. If each page represents 100 metres and the diameter of the Earth is 40,212 km the resulting bump map is 400,000 by 200,000 pixels. This example will require about 80Gb of storage. Since storing even one planet's data on a conventional hard disk is not practical, we need to fall back to non-storage techniques.

We can use procedural white noise as a non-stored source of seed heights for our terrain pages. White noise is the noise observed on a television set receiving a static signal, see Kuo (1996) for more details. To simulate white noise we define a random number generator that takes an X and a Y coordinate and returns a normally distributed value.

If we wish pages to appear the same each time they are visited by the user we must create a quasi-random result in our white noise. A quasi-random result means that the function supplying random noise will always return the same result if given the same x, y inputs. For more information on quasi-random number generators see Ward (1991) and Knuth (1997).

Here is a simple sample function, derived from Ward (1991) that provides quasi white noise.

```
constant maxY := 255

real whiteNoise2d(integer x, integer y)
      begin
      integer n := x + (y* maxY)
      n := (n<<13) ^ n;

      noise := real((n*(n*n*15731+789221)+1376312589)) /
      1474836480.0
      end
```

Figure 5.16 shows the resultant noise generated by the quasi white noise function:

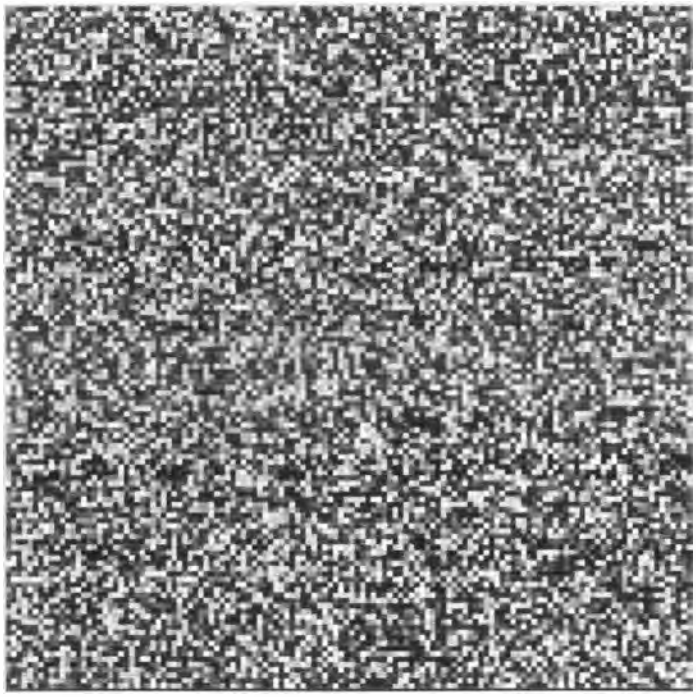


Figure 5.16: White noise generated by a quasi white noise algorithm

5.4.1.1 Planet Creating and Definable Terrain

In the same way that white noise can be used to create seed values for terrain pages, a small bitmap can be altered for this purpose. Revisiting the Earth example in 5.4.1 we required a 400,000 by 200,000 pixel bump-map. If, in this example we start with a 400 by 200 pixel height-map and have one pixel represent 1 million pages instead of one page we have an acceptable image size. By averaging the appropriate pixel from our bump-map with the output from our 2D quasi white noise function we have a definable planet layout. This alteration is useful for game designers wishing to design the layout of their own planets.

5.4.2 Supply-Demand Networks and Dirty Pages

There remains a problem with the offset spherical page management algorithm discussed so far. This problem occurs when two pages are constructed next to each other and thus need to share a common boundary with no seams. This common boundary presents a problem with the paging algorithm as presented so far, because pages on the edge of the submap must be able to tessellate with pages that don't yet exist. To solve this problem this implementation constructs a consumption-demand network. This network has two rules:

- If two pages are constructed next to each other, the right most page is responsible for adapting itself to suit the boundary of the left most page.
- If two pages are constructed above each other, the bottom most page is responsible for adapting itself to suit the boundary of the top most page.

The result is a collection of pages suited to tessellate with each other, with the exception of the top most row and the left most column. These two rows are created without being adapted to the pages around them. They are not visually correct but supply correct boundaries for other pages. Those pages that are not suitable for display are known as dirty pages. Any edge of a page that adapts itself to suit the boundary of another page is known as the demand edge, the other edges used for other pages to adapt themselves to are known as supply edges. When linked up, these pages create a supply-demand network as shown in Figure 5.17.

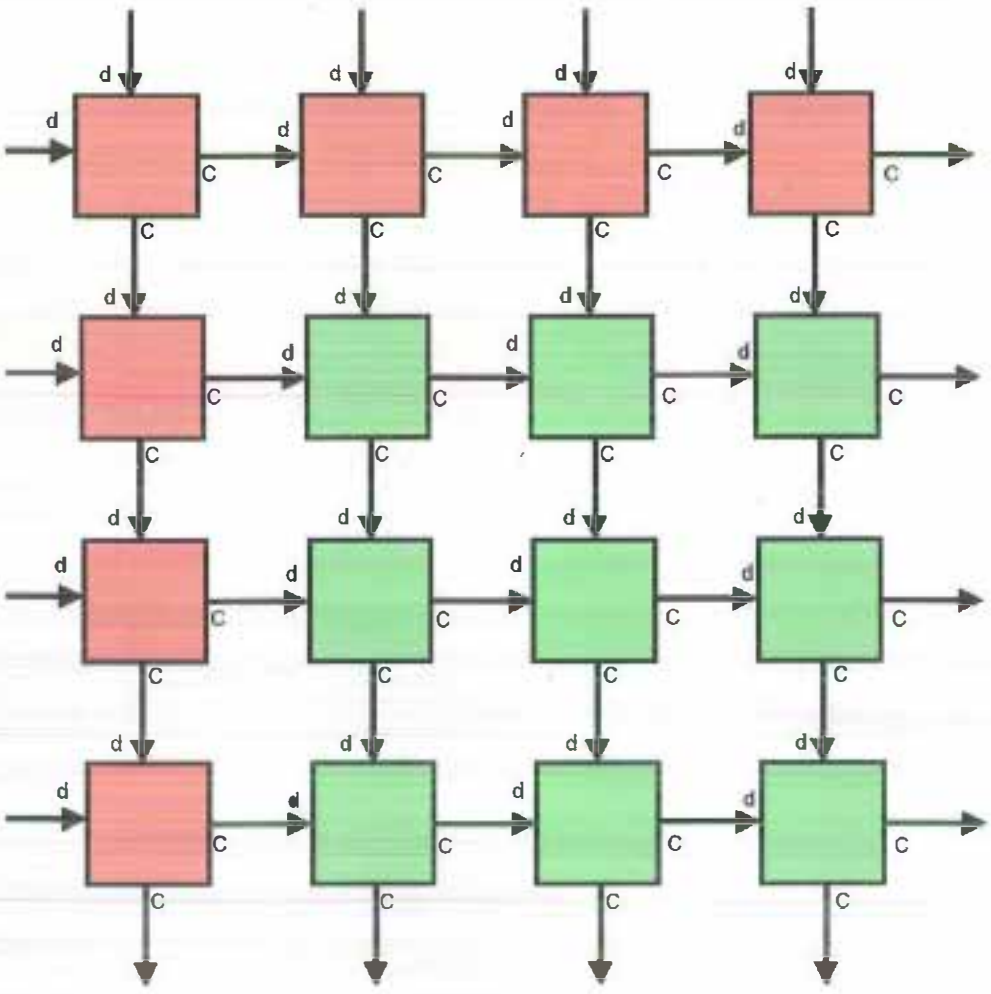


Figure 5.17: A supply demand network

5.5 Level of Detail Control for Terrain Visualisation

The terrain that exists in the submap needs to be optimised for real-time viewing by reducing the amount of triangles present, without degrading image quality. A major advance in geographic visualisation systems in the last few years is the Continuous Level of Detail mesh (CLOD). These meshes are constructed prior to visualisation and provide an optimal real-time rendering of terrain information on modern hardware. To show that the work presented by this implementation is viable for modern visualisation expectations a method for constructing CLOD meshes is integrated into this work.

5.5.1 Overview of CLOD Algorithms

There are currently two major CLOD algorithms used for terrain meshes. The first algorithm is the View Dependent Progressive Mesh (VDPM). The other more commonly used algorithm is the adaptive quad-tree refinement technique.

The VDPM algorithm is concerned with data in a Triangular Irregular Network (TIN). Hoppe (2000) presents an excellent discussion on this technique. However the Adaptive quad-tree refinement algorithm is directly applicable to height field data, making it more suitable for this work. The CLOD algorithm that this work has incorporated was originally presented by Rottger & Heidrich & Slusallek & Seidel (1998).

The goal of a CLOD algorithm is to simplify the landscape mesh in appropriate places so as to reduce the number of triangles used while maintaining the quality of the scene as much as possible. The major task of any CLOD algorithm is to select which areas of a terrain are going to be optimised and how much optimisation is going to be applied to those parts of the terrain.

5.5.2 Examination of Popping Artefacts

CLOD engines that perform the operations discussed so far are typically plagued by the problem of "popping", that is, a terrain artefact that was previously invisible suddenly appears as the user approaches it. The "popping" problem is caused by a terrain element being over-simplified. When detail is increased the CLOD algorithm properly generates the oversimplified element. This artefact is most evident in CLOD engines that use only view distance optimisations.

The main solution to the "popping" problem is the use of geomorphs. Geomorphs essentially allow morphing of a problematic terrain element into a scene. Hoppe (2000) discuses the theory and usage of geomorphs.

5.5.3 Non Degradive Terrain Mesh Simplification

There are two approaches used to identify areas in a terrain mesh that can be simplified:

- Distance from camera optimisation
- Relief/ hill top areas optimisation

5.5.3.1 The "Distance From Camera" Optimisation.

Terrain aspects at a certain distance from the camera are not visible, simply because the on-screen size of these aspects is less than a pixel. Obviously triangles close to the user's position must be rendered in as much detail as possible in order to look good. The detail reduction is focused on terrain aspects distant from the user, so that the missing detail takes less than a pixel when rendered to the screen.

5.5.3.2 The "Relief Dependent" Optimisation.

Relief dependent optimisation focuses on the fact that smooth areas of land can be drawn with fewer triangles than bumpy areas of land. The coarseness of the land is known as the "relief". Figure 5.18 shows in two dimensions how more lines are required for greater roughness.

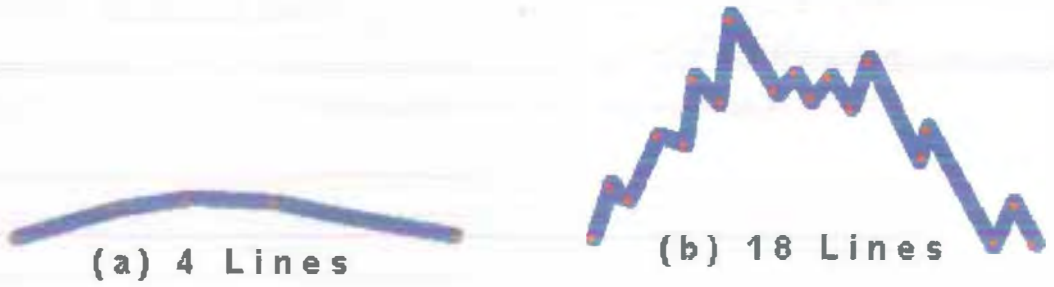


Figure 5.18 Lines used to model terrain in two dimensions

Also of note to this optimisation are hilltops. It is possible that hilltops (crests) may, depending on a user's point of view, be silhouetted against the sky. This situation makes the outline of the polygons very noticeable. Therefore we may dedicate more triangles to the creation of hilltops in order to reduce this effect.

5.5.5 The Quad-Tree Algorithm

The algorithm used by this work to reduce detail (the CLOD algorithm) is known as adaptive quad-tree refinement. This recursive algorithm utilises a data structure that stores a square that is optionally made up of four other squares which in turn are optionally made up of four other squares. An example of the quad-tree structure is shown in Figure 5.19:

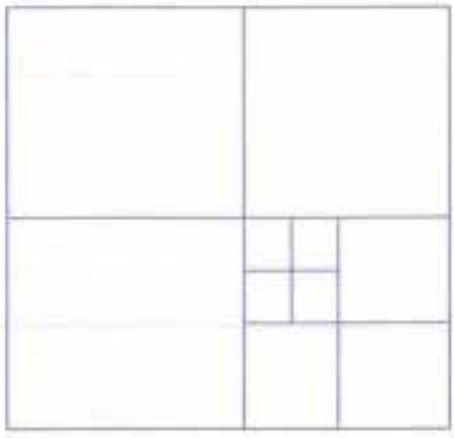


Figure 5.19: A simple quad-tree structure

This allows for squares to be positioned in greater or lower density at different parts of the structure. For example, squares can be positioned around a user's position. The corners and centre of each square correspond to a value in the height field grid. It is with this mechanism that smaller squares in the quad-tree represent areas of greater detail.

Figure 5.20 shows an example of increasing detail around the user's point of view.

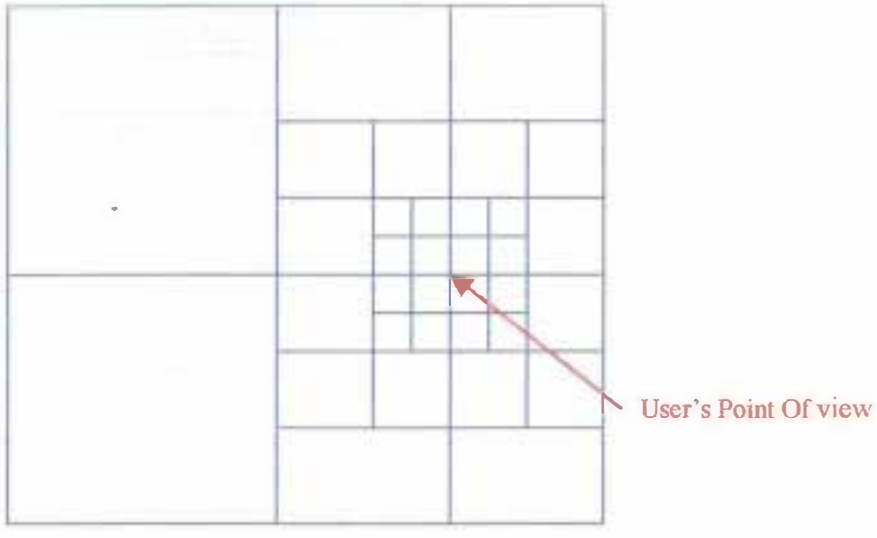


Figure 5.20: A quad-tree structure adapted to a user's position

The corners and centres of each of the final (leaf node) squares in the structure correspond to a point on the terrain, which is rendered to the screen using triangle fans, as shown in Figure 5.21.

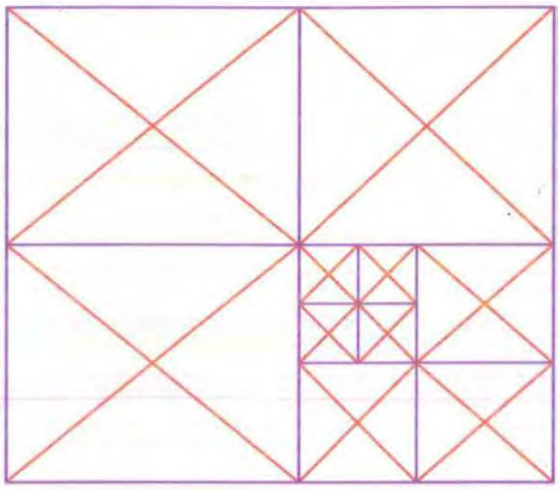


Figure 5.21: A CLOD mesh created by triangle fans

The level of detail mesh shown in Figure 5.21 is now complete. The problem with this model is that the corners of the triangles do not line up, which allows for a problem common to this algorithm known as tearing. Figure 5.22 shows where the tearing problem can occur in a quad-tree structure.

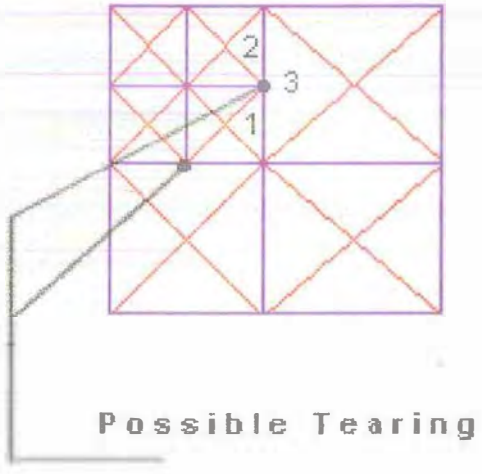


Figure 5.22: Tearing points in a CLOD mesh

The tearing problem can occur on the two points indicated. To highlight how the tearing problem occurs we examine how the right most point will be effected.

- First we define line "A" as the line along triangle 3 adjoining triangles 1 and 2.
- If the elevation of the right most indicated point is such that does not lie along line "A", then there will be a gap in the mesh.

This gap, when viewed by the user, is known as a tear. To remove the occurrence of tearing we have two options:

- Make sure possible tearing points are forced to lie on the appropriate lines.
- Subdivide the squares into more triangles, so suspect points will lie upon triangle junctions.

The first option presented above results in fewer triangles, and less detail but requires minimal computational overhead. However the OpenGL specification and many hardware boards do not guarantee that triangles joining along a line of another triangle will accurately render without tearing, no matter how accurately the points are lined up. This implementation problem makes the second option more desirable. Wright (1999) suggests the use of a mesh of triangles that join points on the square using as few triangles as possible. This technique however involves using structures other than triangle fans.

Rottger & Heidrich & Slusallek & Seidel (1998) present an algorithm that primarily utilises triangle fans. This algorithm was taken into consideration and an adapted for use in the implementation of this work. The CLOD algorithm that is used in the implementation of this work is optimised for quickest mesh construction. However the algorithm used here also has a slight rendering expense due to more triangles.

Figure 5.23 is an example of how the algorithm implemented in this work constructs a mesh so as to avoid the tearing problem.

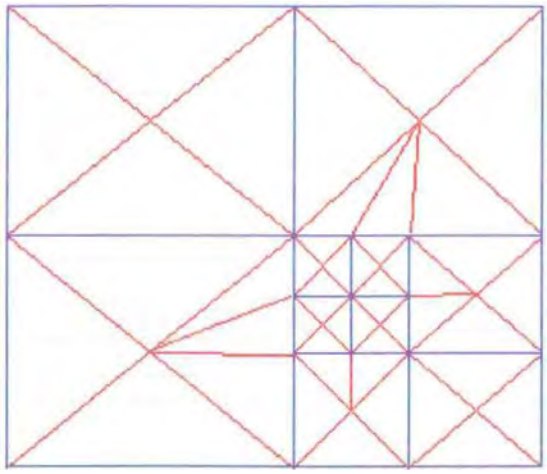


Figure 5.23: A well formed CLOD mesh created by triangle fans

The simple triangle fan algorithm is used to join all points. The simpleness of this algorithm reduces mesh generation time, although rendering time is more expensive. This novel approach to a CLOD algorithm is intended to reflect the increased speed of rendering hardware as compared to CPU speeds.

5.5.5.3 Results of the Quad-Tree Algorithm

Figure 5.24 shows two images that were generated by the CLOD algorithm described in this work. The left image shows the position of the quad-tree squares. It is important to note that every square is always touching a square either of the same size, twice its own size or half its own size. The right image demonstrates detail reduction accomplished by the CLOD algorithm. This image is also an example of a level of detail terrain visible in real-time.

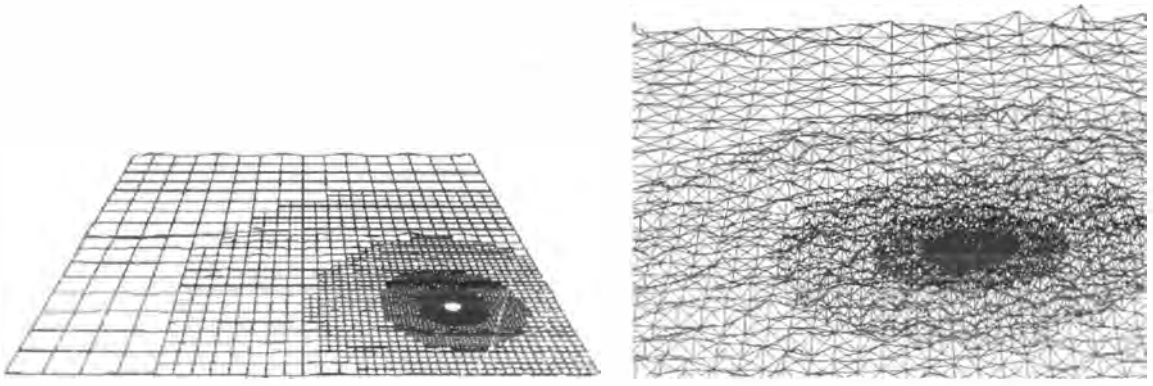


Figure 5.24: Meshes generated by the Quad-Tree based CLOD algorithm

6 METHODOLOGY OF CONTINUOUS PLANT LIFE GENERATION AND VISUALISATION.

In the field of landscape simulation plant life is classified as belonging to a collection of objects called non-terrain elements. The term non-terrain elements refers to all elements of a landscape that are not geographical, eg rocks, roads, plants. The implementation in this work is limited to the placement of plant life non-terrain data. For information concerning storage and usage of more complicated non-terrain elements, such as roads, housing and political boundaries, the author directs interested readers to the very complete works of Ohler (1994).

The placement and creation of plant life is tied to the page management algorithm discussed in the last chapter. The implementation that is provided with this work uses random point generation to place plant life in the terrain. The author directs readers wishing to place plant life in a more scientifically correct manner to Mandelbrot (1977).

Meshes that model plant life can be generated using fractal techniques. In keeping with the level of detail concept discussed in previous chapters, the plant life generation algorithms presented in the implementation can be drawn at different complexity levels depending on how close the user is to the mesh.

This chapter is dedicated to the construction of plant meshes, and how the plant generation algorithms meet the qualitative objectives of:

- Quick execution
- Production of results faithful to the visual appearance of plant life

The primary objectives of plant life generation will also be examined, namely:

- The production of a wide variety of species
- The production of visible variation in different plants of the same species
- The production of detailed plant meshes

6.1 Construction of Plant Meshes

The basis for the implemented tree/plant generation routines follow closely the pioneering work of Lindenmayer (1968). The algorithms presented in this work are focused on replicating what plant structures looks like, that is, plant life appearance, rather than modelling biologically correct plant structure. This difference is subtle, but is used to accelerate the modelling of plants, because it is easier in some cases to utilise the rules of plant appearance, rather than the rules of plant growth.

Because of the focus on plant life appearance, a view of plant topology will be defined for use in this work. It is stressed that these views are biologically based but are not necessarily true to the correct biological topology of plants. This proposed topology is derived from personal research into visual aspects of plant topology combined with existing research on biological topology. The adaptation of the biological topology of plant life for plant life generation is well presented by Reffye & Edelin & Francon & Jaeger & Puech (1988, 151-158) which is a primary source of biological information utilised in this work.

The general structure of any plant follows a recursive model. This model forms the basis for most common plant topology. And the basis of the model is shown in Figure 6.1.

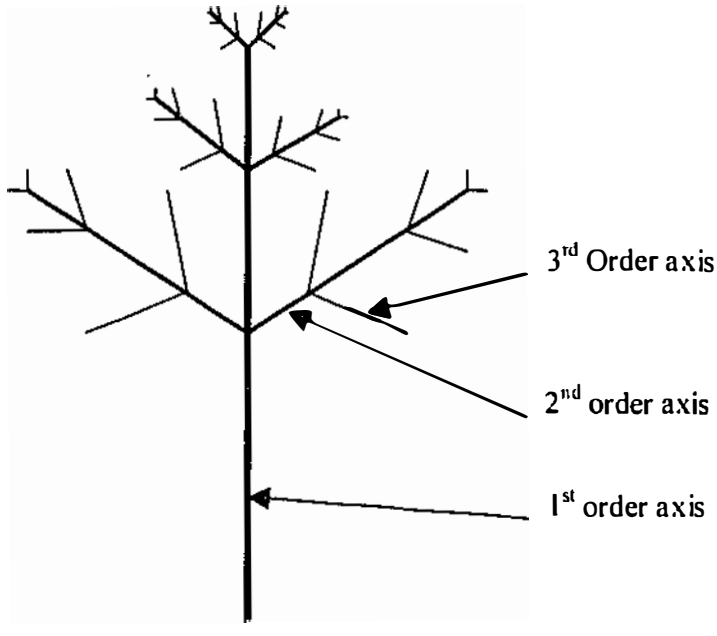


Figure 6.1: The basic recursive model of plant structure

Any tree or plant generally follows these rules.

- A branch is a length of wood containing nodes
- Nodes are spaced at similar intervals along a branch
- A node may spawn one or more branches of equal or higher order.
- A node may spawn a leaf

6.1.1 Quick Execution

It is an essential aspect of this work that the plant generation algorithm is built for real-time generation. Most works on plant generation advise against doing this. A combination of increased processing power and highly optimised code is hoped to achieve the unprecedented goal of real-time plant generation.

Most of the optimisation in this work is based on a simple observation of plant life - plants have more leaves than branches. A plant with only ten branches is likely to have a hundred or more leaves. This results in the recursive algorithms used to generate plants spending most of their time creating leaves. It is because of the exponentially proportional time spent creating leaf nodes that the leaf creation functions are made as simple as possible, thus minimising CPU time used. Conversely the branch algorithms are the ones least called and contribute most to the shape and appearance of the plant and are thus allocated more relative CPU time.

Plant aspects such as rough bark surfaces and the use of textures are not visible from a distance. The plant life can be created with these aspects missing when the plant life is far enough away from the user for these aspects to not be noticeable anyway.

6.2 A Topology for the Production of Detailed and Varied Plant Lives

To produce results faithful to the visual appearance of plant life, data used to formulate individual plant models is derived partly from personal "in the field" research, and from on-line botanical databases. The in the field research was conducted over several months focusing mainly on plant life native to Western Australia. The main on-line database used was "Plants Database" (2000). This database provided mainly information relevant to plant life native to America and Europe.

Central to any plant life topology is branch order. The order of a branch determines the behaviour of the branch. The highest order braches are leaves. The lowest order branch is the trunk of the tree. This is shown in Figure 6.2.

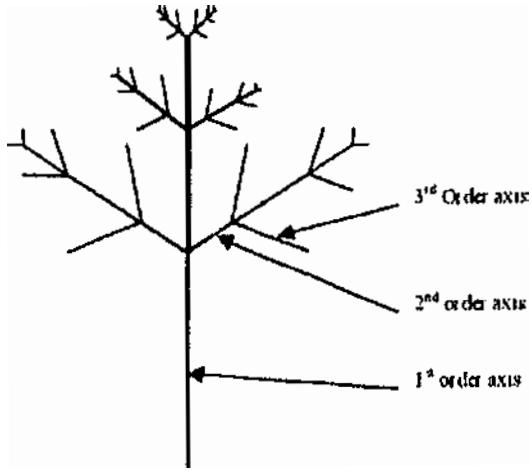


Figure 6 2: Branch order

Each species of plant life has a pre-determined maximum branch order. This makes a configurable branch order very important in producing different species of plants. The implementation provided allows specification of different growth/appearance attributes (parameters) for each branch order. This facilitates creation of complicated plant forms.

There are many parameters used in the generation of plant life. There is a performance trade off in the generation algorithm between the amount of parameters present and execution speed. In order to satisfy the requirements of execution speed and botanical realism the following parameters have been identified as being suitable for inclusion in the plant generation algorithm.

- Ramification
- Length Reduction
- Fertile Area
- Bifurcation
- Continued Bifurcation
- Gnarl
- Phyllotaxy
- Multiple Branch nodes

The above parameter level is complicated enough to model a large variety of plant species. It is unlikely that users would benefit greatly from an increased set of parameters to control plant life generation.

These parameters will now be discussed with reference to how they create realistic plant life representations and how they meet the objectives of detail and variation in plant life.

6.2.1 Ramification

The definition of basic plant topology states that branches can have sub-branches that are of higher or equal order. Ramification is defined as a branch being given a higher order than its parent; ie the sub branch is of a different order to its parent.

There are three types of ramification:

- Rhythmic
- Continuous
- Diffuse

These ramification types are shown in the Figure 6.3.

Rhythmic ***Continuous*** ***Diffuse***

Figure 6.3: Branch order ramification

Definition of terms:

Continuous ramification - each branch is of a higher order than its parent.

Rhythmic ramification - some of the sub branches belonging to a parent are of equal order and the rest are of higher order. There will be a repeating pattern or rhythm to which branches are of lower order.

Diffuse ramification	- There is random function that determines whether any particular sub branch is of equal or lower order

Statistics used in the proposed topology

Ramification type:

Specifics continuous, rhythmic or diffuse ramification. This allows for the production of a variety of plant species.

Lowering of order:

The probability of a child branch being of an order one less than its parent. Child branches that are not of an order one less than its parent will have the same order as its parent. This allows for the production of a variety of plant species. All branches created are tested against this probability. This means that individual plants of the same species will have different orders at different branches, allowing for variation in plants of the same species.

Node survival rate:

The probability of a node on a branch spawning a child branch. This allows for the production of a variety of plant species. All branches created are tested against this probability. This means that individual plants of the same species will have different amounts of branches, allowing for variation in plants of the same species.

6.2.2 Length Reduction

Length reduction refers to the phenomenon by which new sub-branches deriving from a parent branch are generally smaller than their parent. This phenomenon is caused by the sub branches being younger than their parents and not having as much time to grow. Sub-branches being smaller than their parent is not always the case, alterations to this rule are commonly observed when there has been damage to a plant. Length reduction is a "rule of thumb" useful when modelling plants.

Statistics used in the proposed topology

Length reduction:

The average ratio between the length of a parent branch and the length of a child branch relative to the distance between child's base and the parent's base. This allows for the production of a variety of plant species.

Variation:

The amount of variation allowed on the average length reduction of branches from the same species. This allows for variation in plants of the same species.

6.2.3 Fertile Area

A sub branch produced by a parent branch is usually produced at the top of the parent branch. As the branch grows the first sub-branch created remains towards the base of the branch. ie. the closer a branch is to the base of its parent the older it is.

Most forms of plant life have branches with infertile regions that contain no sub branches. This phenomenon for the purposes of this work is categorised into two forms:

Constant Fertile Area

In some cases the fertile area of a plant is designed to only occupy a small part of the branch. Typically the fertile part of this type of branch has a different texture to the rest of the branch.

Percentage Fertile Area

The other type of fertile area most commonly observed is a percentage fertile area. This results in a certain percentage of the branch being fertile. The larger the branch the more the fertile area, but in the same proportion as in smaller branches.

Statistics used in the proposed topology

Fertile Area Type:

Specifies a "constant fertile area" or a "percentage fertile area". This allows for the production of a variety of plant species.

Fertile Area:

In the case of constant fertile area, specifies a distance from the end of the branch that is fertile. In the case of percentage fertile area, specifies the percentage of the branch that is fertile (form the end of the branch). This allows for the production of a variety of plant species

Variation:

The amount of variation allowed in the fertile area form branches in the same species. This allows for variation in plants of the same species.

6.2.4 Bifurcation

Bifurcation was first examined for its mathematical basis by Leonardo Da Vinci in his notebooks (Da Vinci, 1510). Bifurcation is the phenomenon where a branch splits into two, the frangipani in Figure 6.4 is an excellent example of this phenomenon.

Figure 6.4: A Frangipani

Bifurcated branches are all of the same order. The branches formed from bifurcation are not sub-branches (from a biological perspective) and can be considered as being the same branch. Bifurcation is similar to identical human twins in that two branches are formed instead of one.

In Figure 6.4 you will notice that the right most branch has the same length as the sum of the branch that bifurcated left from its base and any one of its children.
This visual aspect occurs because at any division both new branches are just a continuation of the original branch. Thus they have both existed for the same length of time and have received the same nutrient line (growth). Da Vinci noted in his notes that "all the branches of a tree at every stage of its height when put together are equal in thickness to the trunk" (Da Vinci, 1510).

Statistics used in the proposed topology

Chance of Bifurcation:

> The probability of a branch bifurcating. This allows for the production of a variety of plant species. All branches are tested against this probability, this means that individual plants of the same species will have different bifurcations at different branches, allowing for variation in plants of the same species

Balance:

The average dominance of one branch (growth and angle closer to parent) over the other branch produced in the bifurcation. This allows for the production of a variety of plant species

Variation of balance:

The variation in the balance for plant life of the same species. This allows for variation in plants of the same species.

Variation of bifurcation angle:

The variation possible in the angle of bifurcation. Allows deviation of the angle from what is determined by the balance statistic. This allows for variation in plants of the same species.

6.2.5 Continued Bifurcation

Often branches on a plant will appear to divide into 3 or more parts, see Figure 6.5.

Figure 6.5: A Ficus Bay displaying continued bifurcation

On closer inspection we see that what appeared to be a three way split is often a bifurcation of one branch followed by a bifurcation on one of the other branches, see Figure 6.16. Many plants will develop a secondary bifurcation one node after the original bifurcation, similar to how human triplets are made.

Figure 6.16: A Ficus Bay displaying continued bifurcation (detailed view)

Often the secondary bifurcation is at 90° to the original bifurcation, as shown in Figure 6.16. This multiple division is termed continued bifurcation and is not limited to the production of 3 branches but could result in 4,5,6,7,8 or more subdivisions. A useful statistical note is that higher numbers of branch subdivisions become increasingly less probable.

Statistics used in the proposed topology

Chance of continued bifurcation:

The probability of continued branch bifurcation. This allows for the production of a variety of plant species. All bifurcations are tested against this probability, this means that individual plants of the same species will have different continued bifurcations at different branches, allowing for variation in plants of the same species.

Chance of 90degree to last bifurcation:

The probability of continued branch bifurcation being at ninety degrees to the previous bifurcation. Continued bifurcation not at ninety degrees follow normal plant phyllotaxy. This allows for the production of a variety of plant species. All continued bifurcations are tested against this probability, this means that individual plants of the same species will have ninety degree continued bifurcations at different places, allowing for variation in plants of the same species.

6.2.6 Gnarl

Gnarl is the twist of a branch, often either because of traumatic conditions or because of a plant's design, a branch will not grow in a straight direction. Note: often a gnarled branch might produce a protruded knot refereed to as a "gnarl", this protruded feature should not to be confused with the phenomenon being discussed in this section.

For the purposes of this work gnarls will be considered in two parts: the chance of the branch changing direction and the probable degree to which the new direction can change. It has also been noted that sometimes the generation of a large branch from one node will cause a gnarl in the parent branch.

Statistics used in the proposed topology

Gnarl Probability

> The probability of a branch changing direction at any node. This allows for the production of a variety of plant species. This also greatly increases the detail present in plant meshes. All branches are tested against this probability, this means that individual plants of the same species will have different gnarl characteristics, allowing for variation in plants of the same species.

Gnarl Angle

> The average angle at which a branch changes direction when gnarl occurs. This allows for the production of a variety of plant species

Gnarl Angle Variance

> The amount of variation allowed in the gnarl angle in branches in the same species. This allows for variation in plants of the same species.

6.2.7 Phyllotaxy

Phyllotaxy is the position of the buds that create leaves with respect to each other. Phyllotaxy has two common forms: spiralled and distic. The area of phyllotaxy is complex so this work models only spiralled phyllotaxy since it is sufficient for most purposes.

There is considered to be an angle or twist between each node on a branch. The value of this twist is the basic way in which phyllotaxy is modelled in this work, along with some statistics determining the variance from this twist. Figure 6.17 shows examples of phyllotaxy with a twist of 0° and90°.

0° twist

90° twist

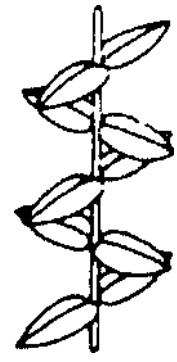


Figure 6.17: Examples of phyllotaxy.

Statistics used in the proposed topology

Phyllotaxy

The angle at which a branch twists at any node. This allows for the production of a variety of plant species.

Chance Distortion:

The chance of there being a change in phyllotaxy for one node. All nodes are tested against this probability, this means that individual plants of the same species will have different phyllotaxy characteristics, allowing for variation in plants of the same species.

Gnarl Angle Variance

The maximum amount of variation allowed in a distortion of phyllotaxy at one node. This allows for variation in plants of the same species.

6.2.8 Multiple Branch Nodes

It is possible for one node to create more than one sub branch.

Typically there are one, two or four sub branches per node. This can vary though, for example clover has 3 leaves at its nodes. Figure 6.18 shows examples of multiple branch nodes.

a) b)

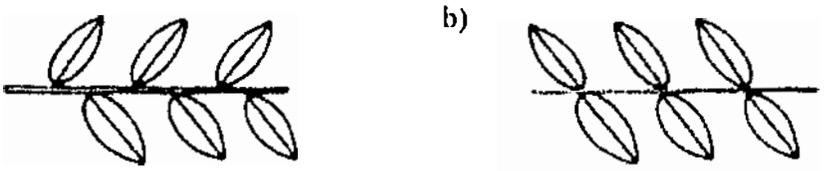


Figure 6.18: a: Branch with one sub-branch per node
b: Branch with two sub-branches per node

Statistics used in the proposed topology

Branches Per Node

The number of branches that will spawn from any node. This allows for the production of a variety of plant species.

6.3 Detail in Plant Life

The goal of producing detailed plant life is achieved by two means. Firstly, the three-dimensional meshes produced by the plant generation algorithm are complex in their shape. This complexity creates detail in the plant meshes. Secondly, using detailed texture maps for bark and leaves completes the detail requirements for realistic plant life.

6.4 Variation of Plant Life

6.4.1 Inter Species Variation

The goal of producing a wide variety of species is achieved by two means. Firstly the paramaterability of the plant generation algorithms allows for the generation of many different categories of plant shapes. Secondly, using different texture maps on the generated meshes completes the visual depiction of different plant species.

6.4.2 Intra Species Variation

The variation of different plants of the same species is achieved by the random placement of branches that follow the rules for the species. Plant meshes can be structured differently but still be in adherence with the statistics of their plant species. This results in plants that appear very different but are still noticeably from the same species.

Figure 6.19 is an example that shows three different plants created in the implementation that all appear to be the same species but exhibit variation.

Figure 6.19 Variation in plant of the same species

7 EXAMINATION OF RESULTS

The examination of results achieved in this work will be taken in two parts:

- Evaluation of functionality
- Comparison with today's technology

The evaluation of functionality is intended to describe how this work meets its objectives. The comparison with current technology is designed to indicate the potential usefulness of this work.

7.1 Evaluation of Functionality

This section measures aspects of the implementation in order to show its capabilities. The aspects measured are a combination of the direct functional goals of this work and the general objectives of any terrain simulator. The primary functional objectives of this work are that the simulator demonstrates:

- A large landscape size
- High landscape detail
- Variation in plant life

The simulator should also achieve the following goals that are common to all landscape simulators:

- Visual realism
- Interactive engine speed

The implementation will be discussed with relation to how it meets these five objectives.

7.1.1 Large Landscape Size

Landscape size is measured in kilometres squared. For the purposes of this evaluation landscapes with $0km^2 - 25km^2$ are considered to have a small landscape size. Simulators with $25km^2 - 100km^2$ are considered medium in size. Anything greater than $100km^2$ can be considered large in size.

7.1.1.1 Results

The terrain available in the simulator covers approximately 4,294,967,296 km^2. This is adequate for simulation of planets similar in size to this earth. Based on this result the simulator is deemed to satisfy the objective of large landscape size.

7.1.2 High Landscape Detail

Landscape detail is composed of the detail available in the terrain and the detail used to model plant life. Terrain detail is concerned with the overall amount of polygons used to render the terrain. The more polygons used the more detailed the terrain will be. Terrain detail is measured as triangles per metre squared, this is measured from the most detailed part of a CLOD mesh. For the purposes of this evaluation the use of one or more triangles per square metre can be considered as developing a highly detailed mesh. The use of between one triangle per square to 0.3 triangles per square metre gives a medium resolution mesh and anything less than 0.3 triangles per square metre can be considered a low resolution mesh.

Biological detail is hard to measure, since there is little data to analyse other than the appearance of the plant life on the screen. For the purposes of this work the visual appearance of the plant will be measured by recording the smallest features visible when the plant is displayed close to the user's position. For the purposes of this evaluation plant life will be evaluated as follows:

- If individual leaves in the plant are not discernible the plant is considered to be low resolution.
- If individual leaves in the plant are discernible the plant is considered to be medium resolution.
- If details on individual leaves in the plant are discernible the plant is considered to be high resolution.

7.1.2.1 Results

The implementation applies about 1.8 triangles per virtual square metre in a terrain mesh. This high density is sufficient for realistic terrain representation. The terrain produced in the simulation is shown in Figure 7.1.

Figure 7.1: Detailed terrain scene from implemented simulator.

This work allows individual leaf aspects to be displayed when a tree is close to the user's position. Therefore the plant life is considered to be high resolution.

As a high amount of visual detail has been achieved this approach is suitable for automatic generation of highly detailed landscapes.

7.1.3 Variation in Plant Life

Variation in plant life is directly measured by the amount of individual plants available in the simulation. For the purpose of this simulation less than ten plants is considered to be low variation. Ten to thirty forms of plant life is considered to be a medium amount of variation. More than thirty forms of plant life is considered to be a large variation in plant life.

7.1.3.1 Results

This work allows the loading of one hundred possible plant species. Each plant species can be represented by approximately 16,000 possible individual plant meshes. This allows for 1,600,000 possible plants in the simulation. This is considered in this evaluation to be a large amount of plant life variation.

7.1.4 Visual Realism

Visual realism is a measurement of how well the landscape in the simulation models a real life landscape. This is almost impossible to put into meaningful units of measurement so the work is assessed by the effectiveness of the features provided to achieve visual realism.

For the purposes of this evaluation a landscape simulator is considered to have low visual realism if a user has difficulties recognising what is being displayed on the screen. A moderate degree of visual realism is achieved when the user instinctively knows what it is that is being displayed on the screen. It is not common for real-time simulators to display a level of realism that makes the simulator difficult to distinguish from a photograph or movie. In the event that this was achieved it would be considered a high degree of visual realism.

7.1.4.1 Results

A major visual realism feature present in the implementation provided here is a generic tree specification language. This language, implemented as a file format, enables specification of plant life topology as described in chapter 6. Another visual realism feature provided is the mid point terrain generation algorithm used to create realistic terrain.

With the presence of these features and the results they produce, this implementation produces images that are readily recognisable as landscape images. Thus the moderate degree of visual realism expected in landscape simulators was achieved.

7.1.5 Interactive Engine Speed

Engine speed is measured by the number of frames that are rendered by the engine per second. A simulator needs to achieve 25 frames per second to create the illusion of a moving image. However, for ease of use with immediate response it should be capable of rendering 30 frames per second.

7.1.5.1 Results

The implementation of this thesis was tested on a modern personal computer and found to have an average frame rate of 45 fps.

The personal computer used for this test was constructed as follows:

- 800Mhz thunderbird CPU
- Geforce 2 GTS video processor
- 512mb system ram
- 32mb video memory

7.2 Comparison with Todays Technology

For the purposes of comparing this work with other works in the field two commercial programs were selected. The two programs and the implementation provided in this thesis where run on the same test machine and the results were compared.

"Tread Marks" (McNally & McNally, 2000) is the first software chosen for comparison. This tank simulator is recent and features a typical landscape quality present in todays hardware.

"Draken" (Denman & Patmore & Ebling, 1999) is the second simulator chosen for comparison. This simulator has large environment sizes and tries to maintain reasonable detail levels. This software is often noted for the quality of its graphics. "Drakan excels graphically" Smith (2000).

7.2.1 Measures used for Comparison

To accurately compare these three pieces of software it is necessary to formalise what aspects of the software will be measured. Furthermore to compare these software titles accurately, units of measurement must be established.

The aspects of software tested measure the size, detail, variation, execution speed and visual realism of the various software titles. The following aspects of the software will be compared in this study:

- Execution speed
- Viewable distance
- Polygon density
- Map size
- Plant life detail
- Plant life variation

Execution speed is a measurement of how many frames are drawn per second on average during a program's execution. This aspect will be measured in frames per second (FPS).

Viewable distance is a measurement of the virtual distance in front of the user where the object clipping takes place. Any object beyond the clipping distance will not be visible in the simulation. This aspect will be measured in metres

Polygon density is a measurement of how many polygons are used per square metre to represent an unsimplified area of terrain. This aspect will be measured in polygons per metre squared.

Map size is a measurement of the size of a simulators virtual environment that is explorable by the user. This aspect will be measured in km^2.

Plant life detail is measured as either being low, medium or high. The measurement given is determined as follows:

- If individual leaves in the plant are not discernible the plant is considered to be low resolution.
- If individual leaves in the plant are discernible the plant is considered to be medium resolution.
- If details on individual leaves in the plant are discernible the plant is considered to be high resolution.

Plant Life Variation is a measurement of how many individual plants are possible in the simulation. This aspect is a whole number derived by multiplying the number of species present by the number of variations possible for each species.

7.2.2 Set up of the Test-Bed

For accuracy of the performance measurements all software titles were run on the same computer. Figure 7.2 is taken to be a view of the integral components of a graphics system.

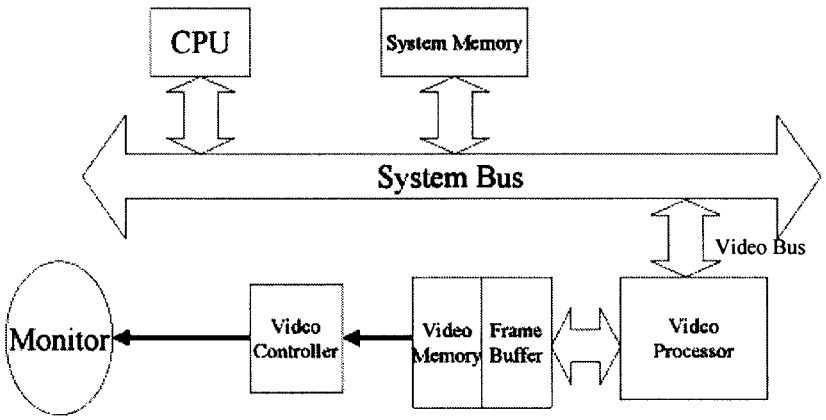


Figure 7.2: Integral Components of a graphics system

The specifications of the test systems integral components are as follows:

CPU	- 800Mhz AMD thunderbird
System Memory	- 512Mb 133Mhz
System Bus	- 133Mhz
Video Bus	- AGP 4x (Fast Write Enabled)
Video Processor	- Geforce 2 GTS 220Mhz
Video Memory	- 32mb 364Mhz DDR
Video Controller	- Asus based VC
Monitor	- Philips 109p (800x600 at 120Hz)

7.2.2.1 Software Set up

In setting up the tests necessary to compare the software, the configuration of the software titles is a key factor in forming a reasonable comparison of the titles.

All software was configured to run at maximum detail levels. However in no case was full screen antialiasing enabled because the performance penalty would be too large and introduce too much unnecessary bias into the results. All software was set up to use 32 bits per pixel and an 800x600 screen mode.

7.2.3 Results of Comparison

7.2.3.1 Evaluation of This Work

Execution Speed

This implementation achieves an average frame rate of 30 frames per second. This is adequate for the purposes of real-time simulation and will improve when the simulator is run on faster hardware.

Viewable Distance

The viewable distance in this simulator is 300 metres. This is a semi restricted visual distance in a landscape simulator.

Polygon Density

This work uses approximately 1.8 triangles per metre squared when rendering a terrain mesh.

Map Size

The size of the overall explorable landscape in the implementation is about 65 thousand by 65 thousand kilometres, or 4,200,000,000 km^2.

Plant Life Detail

The work presented here provides for highly detailed plant meshes. If desired the details of individual leaves are visible in the simulator.

Plant Life Variation

This simulation can handle 100 species of plants. Each species can be viewed as about 16,000 individual meshes. This allows for 1,600,000 possible plants in the simulation. Extending the limit of 100 plants is a trivial task for developers wishing to use more species.

7.2.3.2 Evaluation of "Tread Marks"

Execution Speed

Tread Marks achieves an average frame rate of 45 frames per second. This is adequate for the purposes of real-time simulation.

Viewable Distance

The viewable distance in Tread Marks is 1000 metres. This is a very large visual distance in a landscape simulator.

Polygon Density

Tread Marks uses approximately 2 triangles per metre squared when rendering a terrain mesh. This is an extremely high resolution allowing for very detailed terrain to be displayed.

Map Size

The size of the over all explorable landscape in Tread Marks is about $1km^2$. This is a very small landscape, which in this simulator tessellates so as the landscape is seen to repeat during travel.

Plant Life Detail

This software provides for moderately detailed plant structures. It is possible for a user to distinguish individual leaves on a plant.

Plant Life Variation

This simulator contains only seven possible trees. All plant life present is always going to be one of these trees.

7.2.3.3 Evaluation of "Draken"

Execution Speed

Draken achieves an average frame rate of 60 frames per second. This is more than adequate for the purposes of real-time simulation.

Viewable Distance

The viewable distance Draken is 300 meters. This is a semi restricted visual distance in a landscape simulator.

Polygon Density

This software uses approximately 0.5 triangles per metre squared when rendering a terrain mesh. This is not sufficient to form highly detailed terrain. Because of this jagged areas of the terrain often appear unrealistic.

Map Size

Draken has a larger map size than most simulators using its level of detail. The map size available in this simulator is $1000km^2$. This is achieved by using 10 sets of maps that are approximately 10km by 10km. This software uses 124mb of data to store this information.

Plant Life Detail

Draken provides for plant structures with low detail. The user sees a tree that is more cartoon like in appearance than realistic.

Plant Life Variation

This simulator contains 25 possible plants. This allows for jungle scene to not appear highly repetitive, but the user does become familiar with what every plant looks like.

7.2.4 Comparison of Results

The results show that the solution implemented in this work does not achieve the same frame rate as the competing software. With all the extra overheads used in this implementation this was expected. However the results clearly show a large increase in environment size and the amount of available plant life.

Figure 7.3 shows the different attributes of the compared terrain simulators. On these statistics the simulators are competitive in their results.

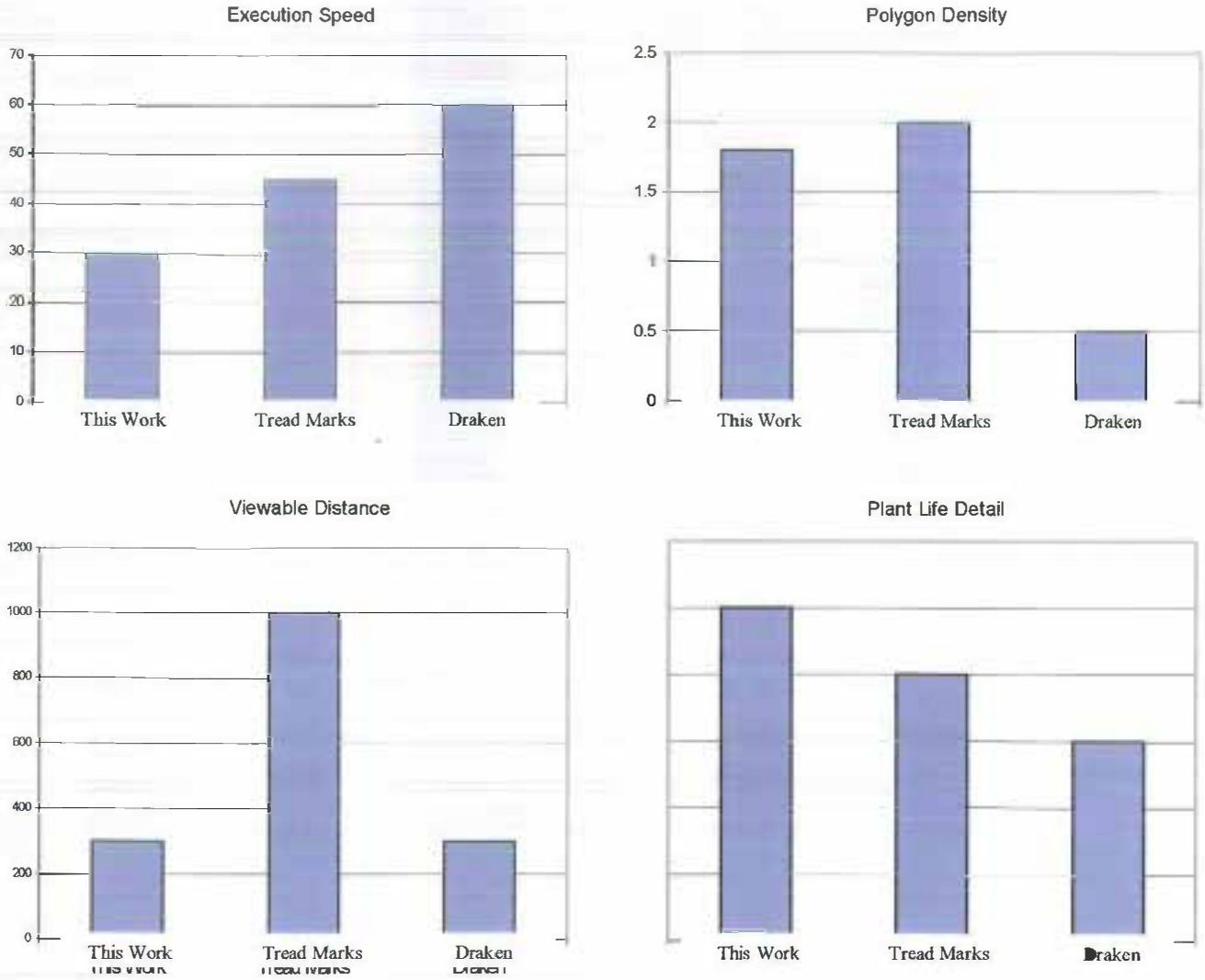


Figure 7.3: Comparison of results

Figure 7.4 shows where this work excels over the other works in terms of map size. Note this graph is scaled logarithmically. This was necessary because the results given by this work show a million fold increase in map size. Without a logarithmic graph the other works results would not be displayable.

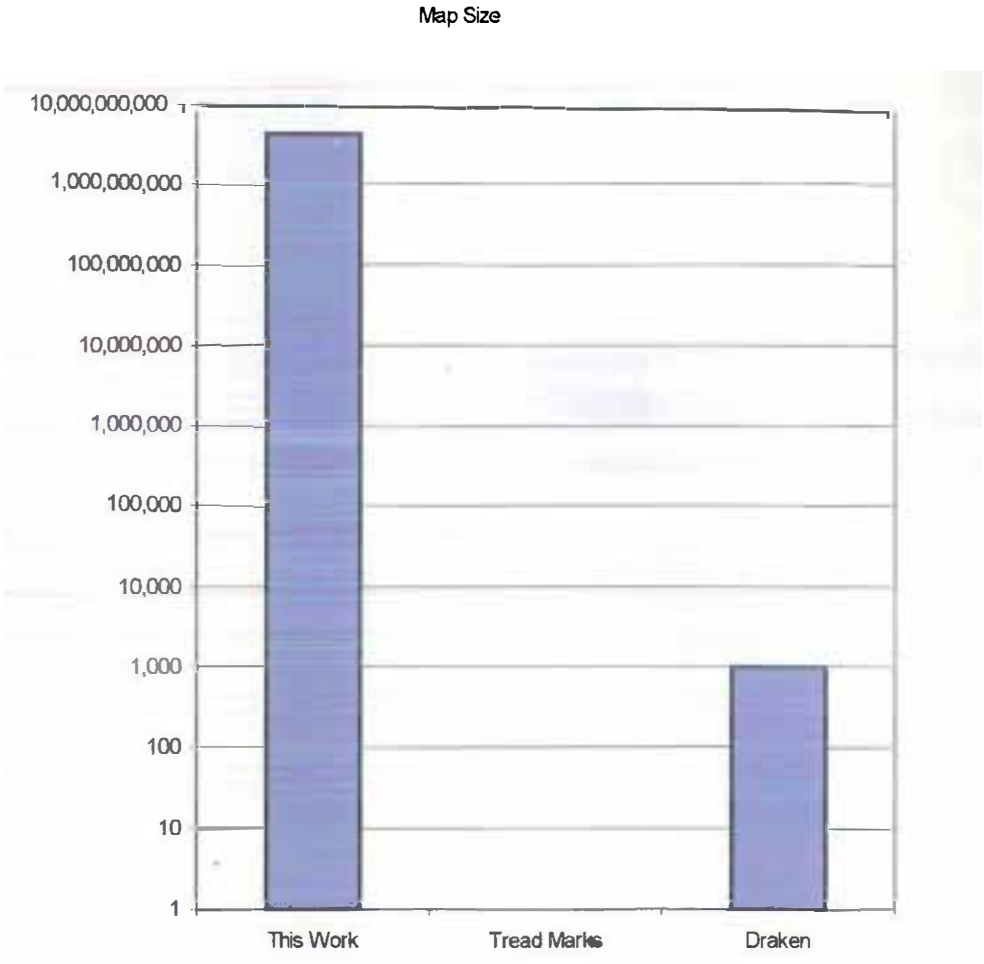


Figure 7.4: Comparison of results

Figure 7.5 shows the plant life variation results of this work. As expected this work has incorporated noticeably more individual plant lives than the other works. Note this graph is scaled logarithmically. This was necessary because the results recorded for this work show a one hundred thousand fold increase in the amount of individual plant life available.

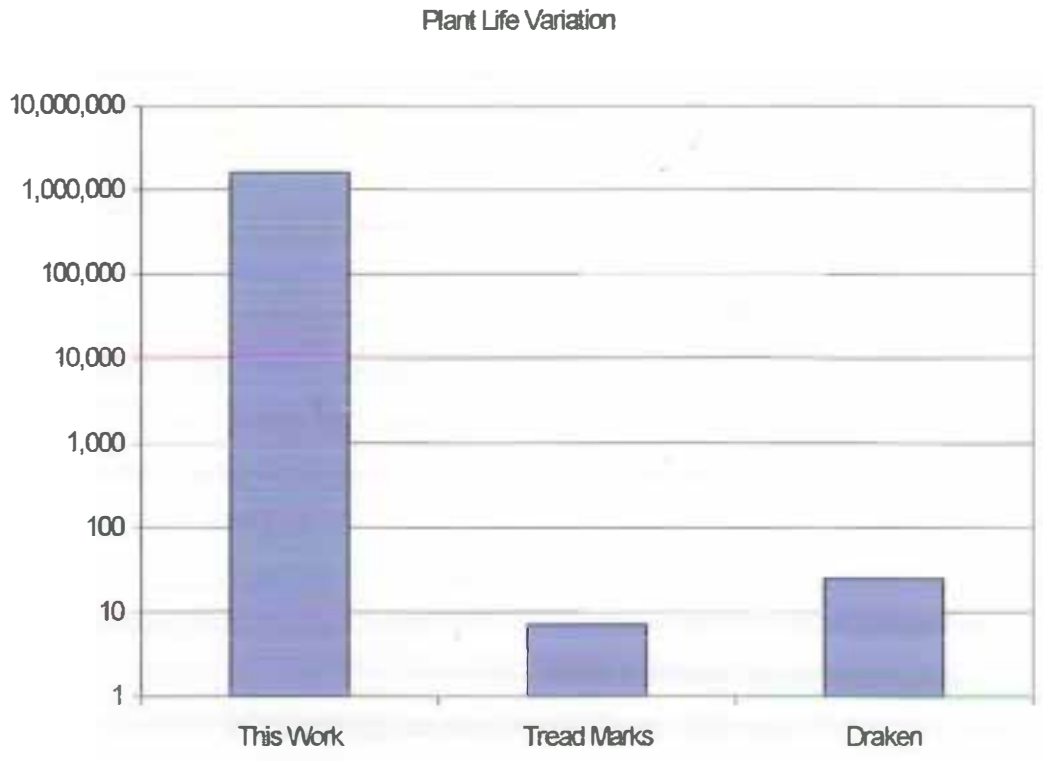


Figure 7.5: Comparison of results

8. CONCLUSIONS

8.1 Conclusions

The problem discussed in section four concerning storage space limitations in landscapes simulators has been solved. The techniques of procedurally generating all landscape elements during a program execution may be slower than alternative techniques but offer an effective solution for providing larger and more sophisticated landscapes than is currently possible using non-generative techniques.

As has been shown CPU technology is currently able to run simulations of the form presented in this work. It has also been established that conventional landscape simulators utilising modern storage devices cannot achieve the same levels of size, detail and variation presented here. It is these two facts that validate the techniques presented here as being viable for modern landscape simulators.

8.2 Summary of Contributions

1. Developed a pioneering approach to remove the size and detail limitations imposed on the terrain present in landscape simulators.

A unique way of handling complex graphical detail and large environment size in graphical simulations was explored. This field has more potential than can be shown in this one work alone.

2. Developed a pioneering approach to remove the detail and variation limitations imposed on the plant life present in landscape simulators.

This allows for a more realistic simulation of a natural environment.

3. Found a practical use for real-time generation of plant and terrain meshes.

8.3 Future Research

The author advises readers wishing to continue the work presented in this paper of two possible research directions:

- The integration of techniques to model roads, road layout, city planning and generation of architectural structures.
- The integration of procedurally generated textures. This research direction would allow every patch of ground to look unique and all trees will seem to have their own bark.

The first research direction would allow cities to be explored in the same way the landscape in this simulation was explored. The second research direction would allow for every patch of ground to look unique and for all trees to have their own bark.

9. REFERENCES

Abelson, H., & diSessa, A. A. (1981). *Turtle Geometry The Computer as a Medium for Exploring Mathematics*. London: MIT Press.

Barnsley, M. F., Devaney, R. L., Mandelbrot, B. B., Peitgen, H. o., Saupe, D., & Voss, R. f. (1988). *The Science of Fractal Images*. New York: Springer-Verlag.

Barnsley, M. F., Jacquin, A., Malassent, F., Reuter, L., & Sloan, A. D. (1988). Harnessin Chaos for Image Synthesis. *Siggraph, 22*(4), 131-140.

Barrett, A. N., & Mackay, A. L. (1987). *Spatial Structure and the Microcomputer*. Houndmills: Macmillan Education Ltd.

Berg, M. d., Kreveld, M. v., Overmars, M., & Schwarzkopf, O. (1997). *Computational Geometry Algorithms and Applications*. Berlin: Springer-Verlag.

Bitters, B. (2000). *Terrian Data*. Available: http://bbq.ncgia.ucsb.edu/education/curricula/cctp/units/unit06/06.html.

Bourke, P. (1997). *Frequency Synthesis of Landscapes*, [on-line]. Available: http://www.swin.edu.au/astronomy/pbourkefreqland/ [1997.

Bourke, P. (1998). *2 Dimensional FFT*, [on-line]. Available: http://www.swin.edu.au/astronomy/pbourke/analysis/ff2d/.

Bracewell, R. N. (1986). *The Fourier Transform and its Applications* (2 ed.). New York: McGraw-Hill Inc.

Brinkmann, R. (1999). *The Art and Science of Digital Compositing*. San Diego: Academic Press.

Brunes, T. (1967). *The Secrets of Ancient Geometry* (Vol. 2). Copenhagen: International Science Publishers.

Carmack, J., Carmack, A., Romero, J., & Hall, T. (1996). Quake (Version 1) [computer software]: id software.

Da Vinci, L. (1510)., [unpublished notes].

Denman, S., Patmore, A., & Ebling, T. (1999). Drakan, Order of the Flame (Version 1) [computer software]: Surreal Software

Eberly, D. H. (2001). *3D Game Engine Design.* San Diego: Academic Press.

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., & Worley, S. (1998). *Texturing and Modeling, A procedural approach.* San Diego: Academic Press.

Foley, J. D., Dam, A. v., Feiner, S. K., & Hughes, J. (1990). *Computer Graphics principles and practice* (2nd ed.). Massachusetts: Addison-Wesley.

Foser, R. (1996). *OpenGL Programming for Windows 95 and Windows NT* Massachusetts: Addison-Wesley Developers Press.

Gleick, J. (1987). *Chaos making a new science.* New York: Viking Penguin.

Hearn, D., & Baker, M. p. (1994). *Computer Graphics* (2 ed.). New Jersey: Prentice Hall. Inc.

Hill, F. S. (2001). *Computer Graphics using Open GL [sic]* (second ed.). London: Prentice Hall.

Hoppe, H. (2000). *Smooth View-Dependent Level-of-Detail Control and it's application to Terrain Rendering*, [on-line]. Microsoft Research. Available: http://www.research.microsoft.com/~hoppe/.

Kelly, A. D., Malin, M. C., & Nielson, G. M. (1988). Terrain Simulation Using a Model of Stream Erosion. *Siggraph, 22*(4), 263-268.

Kenneth, J. I., Grant, C. W., Max, N. L., & Hatfield, L. (1988). *Computer Graphics: Image Synthesis.* Washington: Computer Society Press.

Knuth, D. E. (1997). *Seminumerical Algorithms* (3rd ed. Vol. 2). Massachusetts: Addison-Wesley.

Kuo, H.-H. (1996). *White Noise Distribution Theory*. Boca Raton: CRC Press.

Lindenmayer, A. (1968). Mathematical Models for Cellular Interactions in Development. *Journal of Theoretical Biology., I & 2.*

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., & Turner, G. (1996). Real-time, continues level of detail rendering of height fields. In Computer Graphics. *Proceedings of Siggraph '96*, 109-118.

Magnenat-Thalmann, N., & Thalmann, D. (1985). *Computer Generated Images, The state of the art.* Paper presented at the Graphics Interface, New York.

Mandelbrot, B. B. (1977). *The Fractal Geometry of Nature* (Rev ed. ed.). New York: W.H. Freeman and Company.

Martz, P. (2000). *Generating Random Fractal Terrain*, [on-line}. Gameprogrammer.com. Available: wysiwyg://22/http://www.gameprogrammer.com/fractal.html.

Mauro, J., & McDougall, R. (2000). *Solaris Internals : Core Kernel Architecture*. New Jersey: Prentice Hall.

McNally, S., & McNally, J. (2000). "Tread Marks" (Version 1.0.1) [computer software}: Longbow Digital Arts.

Moller, T., & Haines, E. (1999). *Real-Time Rendering*. Massachusetts: A K Peters.

NIMA. (2000). *National Imagery and Mapping Agency*, [on-line]. Available: http://www.nima.mil/ [2000.

Ohler, T. B. G. (1994). *On the Intergration of Non-Geometric Aspects into Access Structures for Geographic Information Systems*. Unpublished dissertion, Swiss Federal Institute of technology.

Perlin, K. (1985). An Image Synthesizer. *Proceedings of SIGGRAPH '85*, 287-295.

Plants Database(2000)., [Data Base]. Natural Resources Conservation Service. Available: http://plants.usda.gov/.

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. t. (1986). *Numerical Recipes the art of Scientific Computing*. Cambridge: Cambridge University Press.

Prusinkiewicz, P., Lindenmayer, A., & Hanan, J. (1988). Developmental Models of Herbaceous Plants for Computer Imagery Purposes. *Siggraph, 22*(4), 141-150.

Refle, P. d., Edelin, C., Francon, J., Jaeger, M., & Pucch, C. (1988). Plant Models Faithful to Botanical Structure and Development. *Siggraph, 22*(4), 151-158.

Root, M., & Boer, J. (1999). *DirectXComplete*. New York: McGraw-Hill.

Rottger, S., Heidrich, W., Slusallek, P., & Seidel, H.-P. (1998). Real-Time Generation of Continuous Levels of Detail for Height Fields.

Savchenko, S. (2000). *3D Graphics Programming Games and Beyond*. Indianapolis: Sams Publishing.

Smith, R. (2000, August). *"Drakan: Order of the Flame"* [Review of the computer software Drakan, Order of the Flame]. PC Gamer California: Imagine Media

Ulrich, T. (2000). Continuous LOD Terrain Meshing Using Adaptive Quadtrees. *Gamasutra*(228).

Ward, G. (1991). *A recursive Implementation of the Perlin Noise Function*. GraphicGems(Vol. 2) , 396–401. New York: Ap Professional

Watkins, C. D., & Sharp, L. (1992). *Programming in 3 Dimensions*. San Mateo: M&T Publishing, Inc.

Watt, A. (2000). *3D Computer Graphics* (3 ed.). Essex: Addison-Wesley.

Watt, A., & Watt, M. (1992). *Advanced Animation and Rendering Techniques*. New York: Addison-Wesley.

Wernecke, J. (1994). *The Inventor Mentor Programming Object-Oriented 3D graphics with Open Inventor* (2nd ed.). Massachusetts: Addison-Wesley.

Williams, R. (1979). *The Geometrical Foundation of Natural Structure. A source book of design*. New York: Dover Publications, INC.

Woo, M., Neider, J., Davis, T., & Shreiner, D. (2000). *OpenGL Programming Guide* (Third ed.). Massachusetts: Addison-Wesley.

Wright, R. S., Jr., & Sweet, M. (1999). *OpenGL Super Bible* (2nd ed.). Indianapolis: Waite Group Press.

GLOSSARY

Artefact	Undesirable effect unintentionally present in a computer rendered scene.
Bifurcate, Bifurcation	To split in two, as in a lake becoming two lakes, typically involving a "Y" junction.
Billboard	A flat textured rectangle used to create the illusion of a 3D object.
CLOD (mesh)	A mesh that can have its detail changed at any part its surface.
Level Designer	A person who design's an environment for a computer game
LOD (mesh)	A mesh constructed with different levels of detail in different parts of the mesh.
Mesh	A collection of joined triangles that represent an object in three dimensions.
Popping	A sudden noticeable object that appears as a scene's detail increases. Regarded as an undesirable artefact.
Quad-Tree	A data structure that recursively subdivides a 2 dimensional area into quadrants.
Tearing	An artefact produced when three triangles join in a "T" Junction.
Terraform	To shape a terrain. Specifically to alter a terrain/environment to resemble the earths natural terrain/environment.
Terrain Mesh	A mesh used to define a terrain, typically with no overlapping segments
Topology	The concept behind the organisation of a structure, this word has no plural form recognised in the English language.
Triangle Count	The number of triangles used to define a mesh or scene
Triangle Fan	A series of triangles sharing common edges and one common corner.

INDEX

Q

R

S

T

V

W