Procedurally Aided Level Design

by

Dariaus Stewart, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2010

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Dariaus Stewart

September 14, 2010

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Dariaus Stewart

September 14, 2010

Acknowledgments

I would like to thanks my supervisor Mads Haar and course director John Dingliana for their help and suggestions with this project. I would also like to thank my friends and peers in the Interactive Entertainment Technology course for a truly memorable experience.

Dariaus Stewart

University of Dublin, Trinity College
September 2010

Procedurally Aided Level Design

Dariaus Stewart
University of Dublin, Trinity College, 2010

Supervisor: Dr. Mads Haar

Recent research in procedural content generation has demonstrated ways of creating a vast amount varying geometry. From multiple types of variant terrain to vast cityscapes have been explored. Many theorised that the methods employed in such research would become adopted by the games industry to solved their growing development costs for producing game content. But this has not been the case apart from a few exceptions. The purpose of this research is to explore the task of applying procedurally generated content intended for use within games. It is probable that content produced procedurally must meet other criteria that would allow it to be more pliable.

This thesis will first analyse the multiple applications of procedural generation techniques and looks at their usefulness within game development. The fundamental theory within this report, is that procedurally generated content in most cases, would be unsuitable for use in games. The reasons for this are numerous but the foremost one is the lack of control and flexibility available when procedural methods are employed. This report advocates that content for games should be authored by designers on

an abstract level and then procedurally enhanced to create detailed physical models. This is to allow the author to explicitly state the inclusion of certain desired aesthetic features.

To demonstrate this theory a unique approach to developing suitable game content using various procedural content generation techniques is presented along with a example implementation. Specifically the generation 3D game levels will be done to highlight the merit of this method. This novel approach demonstrates how procedural methods, if applied tactfully, can generate quality content that would be usable in a variety of game genres. The output will conform to the structure provided by the author but also introduce small random elements to aid replayability.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Goal

This project and dissertation focuses on the development of a unique system which can author content to be used as 3D levels within video games. The system should be flexible enough to generate a wide variety in content so that it can be reusable for multiple types of game genres.

To achieve this, two objectives much be reached. The first will be to design and create an intuitive interface that can not only be utilized by developers but also by none technical personnel such as consumers. This will require user input to be kept at a minimum but still offer enough options to guide the procedure to generate desirable content. Using the interface, users will describe the basic layout of the level they want to generate by controlling the placement and shape of landforms with the inclusion of other aesthetic features.

The second objective will be to implement a procedural system which can enhance the outlined basic level structure into the full geometry of a level using procedural techniques. The system should be robust and produce varied terrain types. These terrain types may not always produce realistic results. Limiting the system capabilities in such a way to produce only realistic outputs would also limit users creativity. Games can be incredibly creative media forms so it is important for the system to be flexible in this area.

While achieving these objectives, this research will demonstrate the strengths and

weaknesses of procedural content when they are applied a highly interactive form of media such as video games.

1.2 Motivation

While the procedural generation of multiple types of geometry has been extensively research, applying them to a specific purpose such as game development remains relatively unexplored. Also past research only focuses on one type of geometry, however for this system to be successfully used in games development, it will have to be flexible enough to incorporates multiple types of geometry simultaneously.

A system like the one purposed has multiple applications. If it can't produce content of a high enough quality to be used in games, it could be a way of rapidly prototyping levels. This is important in games as levels need to tested excessively to ensure their pacing is correct. It could also be a light weight tool suitable for consumers to use as consumer generated content has proved to be a successful method of expanding game content.

With the specific purpose of being using in games, it could incorporate other things into the procedure that could aid other areas game development. For example it could generate a navigation mesh along with the geometry that could aid path-finding for computer controlled agents. Utilizing the compression capabilities inherent in procedural techniques, the data size of the levels would be significantly smaller than those created using traditional tools. This would help ease the growing demands on storage and facilitate faster downloads of downloadable content which is now quite common. The need for optimisations is an important thing within the games industry and some procedural techniques can be easily adapted into level-of-detail algorithms.

This research will also give insight into whether procedural techniques are a viable alternative for creating game content. Advantages and disadvantages of such a comprehensive system will be investigated and if the approach is suited to certain content more than others.

1.3 Document Outline

The remainder of this thesis is structured as follows:

Chapter 2 provides a brief background to some of the topics related to this research

Chapter 3 contains a review of research done in various areas that are relevant to the topic of this thesis. It includes a description of multiple types of procedural generation methods for terrain, ecosystems and urban environments.

Chapter 4 introduces a novel approach to level authoring that is aided and extended by some procedural methods described in the previous chapter.

Chapter 5 describes the development of a system that implements the approach presented in the previous chapter. Individual components are detailed and the reasons for their exact implementations are specified.

Chapter 6 Evaluates the system developed on the basis on its original goals. The effectiveness of elements integrated into the approach are also investigated.

Chapter 7 Concludes this report and discusses the overall results of the project and details findings discovered during its production. Future work is also suggested.

Chapter 2

Background

2.1 Procedural Techniques

"Procedural techniques are code segments or algorithms that specify some characteristic of a computer-generated model or effect. For example, a procedural texture for a marble surface does not use a scanned-in image to define the color values. Instead, it uses algorithms and mathematical functions to determine the color." [25]

Procedural techniques were originally used to generate textures and became an active area of research on their own when they were first used to create realistic 3D textures in [20]. Procedural techniques have since been exploited as vital tools for creating realistic graphics in applications ranging from movie special effects to computer games. They have most successfully been applied to the animation and simulation of natural phenomena such as fog, fire, water, and atmospheric patterns [8].

There is also an inherent compression capability when using procedural approaches. Rather than explicitly specifying and storing all the complex details of the content being generated, they are abstracted into functions or algorithms that form a procedure. Storage savings are gained as the details are no longer explicitly specified but implicit in the procedure. The game .kkrieger [27] impressively demonstrated this fact by creating a game which only uses 97,280 bytes of storage, whereas if it were made using conventional techniques it is speculated the it would require around 200-300 MB.

2.2 Level geometry

The level geometry in today's games is usually stored as a fixed model. Although this is wasteful it is currently acceptable due to the large storage capacities available. It is admissible to acknowledge this will not always be the case. In the previous decade games have progressed beyond the capabilities of numerous storage devices as CD's and DVD have been superseded by Blu-ray [2]. The hard disk drive capabilities of the current generation of game consoles are also being used extensively for installing additional game content. Utilizing the compression aspect of procedures, procedural approaches to content generation could be used to ease these demands on storage.

The predominant use of procedural methods lately has been the generation of various forms of geometry. There has been numerous research papers on the procedural generation of terrain, cityscapes, dungeons and interiors. Also real-time and offline generation of teleological and ontogenetic content has both been explored. These techniques have yet to be accepted into mainstream game development as they possess a number unreliable traits. Giving more control to an algorithm to produce the content, means less control is given to the author. This severely limits the creativity of the author to whatever the procedural techniques allow. Algorithms that employ stochastic elements contain potential risks for developers as they can produce content that maybe unfit for consumer consumption. Also the ability of capturing emotion within the output is something which has remained unexplored.

Despite these weaknesses, procedural geometry has been used on occasion very successfully within games. For example SpeedTree has been utilized extensively for the supply of vegetation to multiple games. The 5,000 square miles of terrain within the game Fuel [7] is entirely procedurally generated at runtime, as it would impossible to store it as a fixed model. A similar but ultimately more extreme approach to geometry is being used in Infinite [4]. Entire galaxies are created procedurally where the user is capable of seamlessly traveling from planet to planet uninterrupted by load times. This is possible because the procedural terrain can have the natural ability to be reproduced at desired resolutions. This allows an adaptive Level-of-detail to be used to ensure player immersion is not lost with the interruptions of loading screens.

2.3 Structuring level design

Traditionally the structure of levels within games is entirely linear. This is approach has many advantages. It allows the developers to dictate the players experience throughout the level with a high degree of control. By guiding them along a set path they can trigger dramatic scripted events and ensure that the player is in the best place to witness them. It is simple to design and simple to use as players know instantly the direction they're meant to traveling. Because of its simplicity this method is usually deployed within fast paced games where the challenge of navigating levels is secondary to other challenges within the game. The one main drawback of this approach is that it greatly reduces the replayability of the game. Because of the linearity, the challenge of the game becomes a memory exercise. Games have tried branching paths and storylines to positive effects but they are time consuming to construct. The best approach for achieving non-linearity, is to create open-world designs or randomly generated structures.

Open-world maps add another dimension to gameplay where the player can tactfully decide how to proceed through a level and approach its problems. Because of this, this method is usually deployed in games that promote strategic gameplay. This approach requires careful planing on behalf of the developers because if it is not implemented correctly it can ultimately have a negative effect on the game. This can happen when developers spend too much time creating this extra content they don't have time or the means to make this extra space as interesting or interactive as a smaller more compact space.

So far only dungeon based levels and expansive open world terrain have been explored procedurally within games. The dungeon based attempts have been criticized for being repetitive and dull. This is likely due to the generation algorithm being restricted to containing only a limited number of prefabricated objects. Because of the vast amount of content that can be produced by procedural methods, especially when it comes to terrain, there is an expectancy for it to be used in open world applications. Detailed levels can be produced however giving all of the content a context or a purpose has been a problem and therefore may only suit a limited number of genres.

2.4 Gameplay

Gameplay can be describe by either emergent or progressive. Emergence is found in games that have developed numerous robust and logical systems that leads to player-unique solutions to situations in the game that developers didn't anticipate. Types of games where emergence is normally found are usually strategical inclined. Progressive is more common and is found in linear games, where the player behaves according to how the designers expected by using the game mechanics the way they were intended. Usually the levels include a number of highly scripted events to provide the best user experience.

Recently there has been experiments using procedural content in an attempt to fabricate the players experience. The content generated ranges from simple quests to the quantity of enemy combatants. Results suggest the generation of entire quests to be too ambitious and yielded negative response. However the generation of smaller challenges such as the amount and type of enemies that engage the player in particular areas was far more successful. This is because the repetition of smaller challenges is acceptable whereas in the repetition of quests the player does not get the feeling that progress has been made, which should be part of the reward.

Another advantage of generating the smaller challenges procedurally, is that there is the possibility of scaling difficult to adapt to the players skill. These approaches are an attempt to remove the predictability present in linear games.

2.5 Storytelling

Narrative experiences within games has almost solely been expressed in a linear fashion. It was suspected that this linearity was reducing the replayability of campaigns within games. Attempts at creating stories with branching elements have been used but only in rare occasions. This is probably because of the difficulty in creating intricate storylines. This is has become a realisation within the games industry and so other strategies have been used increase replayability without creating multiple branching storylines. These include dynamic gameplay as mentioned briefly in the previous section and the option to playing through campaigns cooperatively with other players. It has also been problem for procedural generators to achieve content that is in keeping with the accompanying

narratives.

2.6 Interface

One of the overlooked aspects of designing a comprehensive procedural generation system is defining a user interface that allow users to intuitively manipulate the user-accessible parameters of the procedure, so that they can have better control over the outcome. This is vital because generally developers have ideas for content, usually in the form of concept art, and want certain aspects present in the eventual outcome. This is where most procedural systems fail to produce good content, as developers lack desirable control to structure the outcome.

The interfaces for managing procedural data are often either custom implementations for a specific project and often do not exist outside of the code itself, requiring designers using the system to have programming knowledge. Procedural methods may produce a lot of content quickly. However the quality of the content is usually defined by how well developed the procedure generating it is. Therefore the output of under-developed procedural methods are more than often uninteresting both in terms of gameplay and visuals, and produce repetitive results.

Procedural interfaces must minimize programmer involvement and be flexible enough to effectively control the outcome of generated content. This is necessary as content may have to comply to certain criteria to be usable. Another reason to incorporate a friendly interface, is that consumer produced content is recently becoming a popular method of adding more content to games, therefore it is not unlikely that the end users to the system may be consumers that are unfamiliar to technical development tools.

Chapter 3

State of the Art

3.1 Generation Systems

3.1.1 Procedural Vs. Manual Modeling

For modeling content for games there are two opposing approaches. There is the traditional method of manual modeling and the considerably less used procedural approach.

Software that specialises in manual modeling such as Autodesk's 3ds Max or Maya, focus on providing a comprehensive toolset that allow users to model almost anything their imaginations can visualise. This is undeniably the greatest asset for this type of modeling but ultimately the root of all its negative aspects. The tools although comprehensive can take a lot of effort to learn and are incredibly intimating for new users due to their clustered and complex interfaces. Such tools have been used within the games industry for a majority of the previous two decades and have since grown to become quite expansive, encompassing a variety tools that aid animation and rendering as well as modeling. However they are still laborious and repetitious in use and require specialized 3D modeling skills to be utilized to their full potential. Currently these trade offs are considered worth taking but it is becoming somewhat of a concern within the games industry as it is becoming incredibly expensive to produce content in this way. This is due to the time it consumes produce the content in the vivid detail that is expected by today's consumers. Using this approach it is possible to define every point within terrain, and the position and properties of every object within the landscape. This gives designers absolute control but depending on the size and complexity of the

intended content, it can require an excessive amount of time to create and memory to store.

Procedural approaches are on the opposite side of the spectrum where they are less developed and therefore less used within the games industry. Their strengths are the speed and quantity in which they can produce content. Also they have small memory footprint as they are only stored as code. The largest issue impeding procedural methods from being used in games is the lack of control they offer users. The outcome is entirely limited to how well developed the procedure in use is. Also procedures typically only produce one type of content, be it flora, terrain or buildings. However when they are combined successfully they can produce some incredibly compelling content and can save the designers from a lot of tedious work specifying details.

Manual modeling tools have been developed to the a point where the only restriction on the output, is the creativity of the designer. However their increasing costs are making procedural methods a promising approach but are ultimately too underdeveloped to become an alternative. To further the use of procedural methods, the procedures used will have to become incredibly dynamic and flexible, removing their limitations or at least making them more transparent.

It is possible to facilitate the strengths of both by using a combination where the designers are allowed the freedom to specify aspects of large portions of the content an allow procedural methods enhance these into a detailed model saving the designer from any tedious and repetitive tasks. Parametrization of the procedural functions would allow designers to accurately specify details within the generated content and allow more creative freedom. It is this type of approach that is used within this project.

3.2 Procedural Geometry

3.2.1 Procedural Terrain

To date there is a lot of research on the procedural generation of terrain because due to is complexity and vastness it would be incredibly laborious to create manually. Also some simple techniques that generate terrain can produce realistic results. There are two types of approaches for generating terrain, teleological and ontogenetic.

The teleological approaches focus on creating realistic results through the simulation

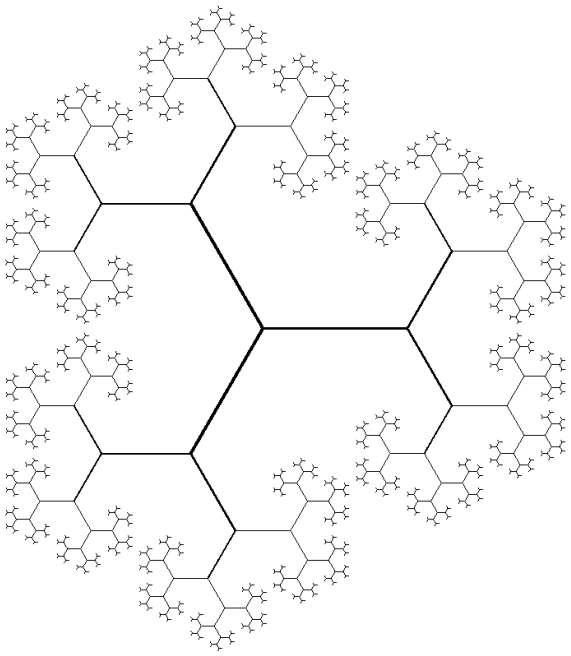

Figure 3.1: Simple Fractal Image

of real-world physical phenomena. As to be expected, these approaches achieve a high level realism, however they are incredibly computationally expensive to perform. Contrary to this approach is ontogenetic modeling which does not rely on physically accurate calculations to produce its outcome but is in fact based on no more than visible morphological features. This approach is generally preferred because it is quicker to generate as multiple iteration may have to be preformed before a desired outcome is achieved.

A very common approach to generating ontogenetic terrain, is through using fractal geometry. Fractals are "a geometrically complex object, the complexity of which arises through the repetition of a given form over a range of scales."[25]. Fractals posses dialated symmetry where the object is invariant under change of scale. For example, a small part of a cloud looks like a larger part, but only qualitatively, not exactly. An example of a fractal can be seen in figure 3.1 and example fractal terrain in figure 3.2. Key algorithms for generating fractal terrain are demonstrated in [13, 26].

Simulating various forms of erosion upon a base terrain model is a common approach to adding an extra layer of realism and detail to terrain. Erosion techniques have been researched by for almost two decades with Musgrave et al[16] describing one of

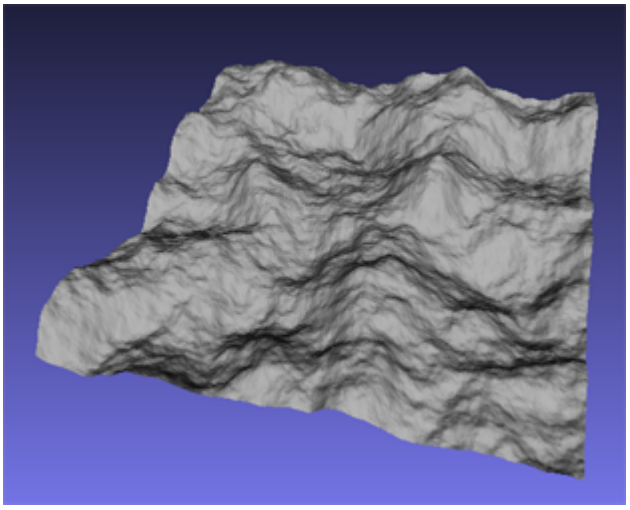

Figure 3.2: Sample Terrain Generated from Fractals

the first algorithms for visually simulating terrain erosion. Typically erosion dissolves material from steep slopes and transports it downhill where it is then deposited at lower inclinations. The result tends to make steep slopes even steeper, and flatten out low-altitude terrain where the transported material is deposited. This is exactly the effect achieved by Chiba et al [6] which simulates the forces caused by running water on the terrain surface. The water is approximated by particles and a simple collision detection algorithm is used to simulate the erosion. These approaches are however quite expensive to calculate and therefore unlikely to be used in games or any other media in which terrain may be generated at runtime. For such cases, there are less physically accurate methods that achieve acceptable results. For example the research in [18], applies certain optimisations to pre-existing algorithms for thermal and hydraulic erosion, to achieve suitable eroded terrain within acceptable times.

Although the methods described in this section produce realistic terrain, in most cases such results would be unsuitable for use within games for anything more than a scenic backdrop. This is because they lack fundamental aspects, such as a navigable paths, which are necessary for typical games in which players must traverse the terrain in some manner. [18] attempted to ease this problem by generating terrain with more flat areas, however this approach lacks the ability to add specificity to this feature which would enable designers the control to produce more usable terrain. It is clear that more input form the designer is required to produce usable content but for this to

become a viable approach there must also be a generation process that can take this input and produce suitable terrain.

Generated terrain may be represented in many formats once generated. Most common of these is as a height-map which is a two-dimensional grids of elevation values. Most of the methods in this section work exclusively with height-maps. This is mainly because of their simplicity and ease of use.

3.2.2 Sketched Terrain

This is a relatively new approach to generating terrain and certain features held within. The idea is that the user provides a generative system with sketched based inputs such as splines, which are in turn interpreted by a procedure that will produce some sort of 3D output.

One of the more successful implementations of such a system can be seen in [9]. Here users can control the placement and shape of landforms without sacrificing realism. This system enables users to draw a silhouette, spine and bounding curves of hills and mountains and may embed other landforms such as river or canyons. With a sketching interface users can interactively create or modify landscapes that include varied and complex landforms. An example of the input and output of this system can be seen in figure 3.3. This method appears to be a very natural and intuitive approach and provides superior control over the output compared to the parameter manipulation of the methods in the previous section.

A more recent exploration of this approach can be seen in [23] where not only terrain is produced but also vegetation, urban districts and roads. This research comes closer than any previous research towards a framework that may be suitable for developing usable content for games. This is because it does not focus on one aspect of the content, such as the terrain, but incorporates other aspects which usually co-exist with the terrain. It also combines these aspects in a user friendly manner enabling non-specialist users the ability to create interesting content quickly. It has two main interaction modes: landscape mode, where users define ecotopes including elevation ranges and terrain roughness; and feature mode, in which elements like rivers, roads, and cities are placed on the above landscape. The procedural sketching facilities described here provide designers with the productivity gain of procedural methods, while still allowing

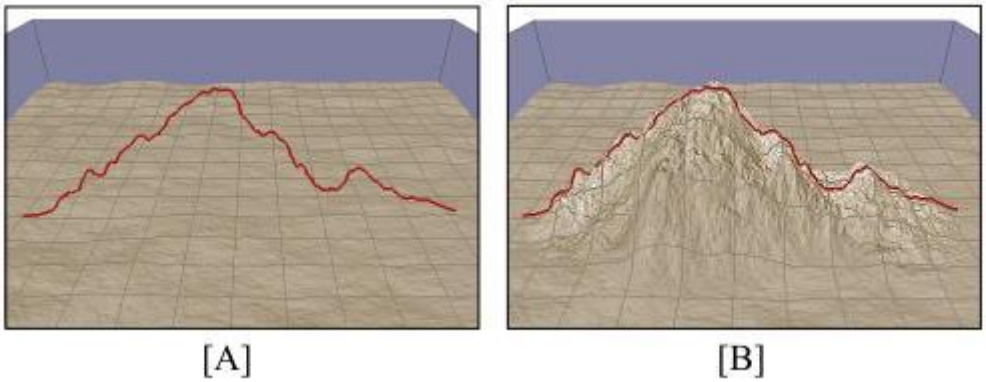

Figure 3.3: Terrain Sketching
[A] the user sketches a silhouette curve. [B] A matching landscape is created by surface deformation with noise propagation.

for fine user control. Although this research describes a framework which incorporates multiple procedural techniques it does not specify which ones it uses or which ones work best with this framework.

3.2.3 Ecosystems

Generating ecosystems is usually the first step after generating terrain to add detail to the environment. This involves two things: the generation of the various plant life that will be contained within the terrain; and the distribution and placement of this plant life. The complexity of natural scenes makes them not only difficult to render, but also to specify. The challenge stems from the visual complexity and diversity of possible modeled scenes. Fortunately this is one area procedural methods are considerably well equipped for as it would be extremely tedious to model and place every plant form within a scene.

The distributions of plant life is a complex phenomena which is the result of a staggering amount of variables which include soil chemistry, local climate, botanical history and competition with other plants within the area. Calculating each of these factors would likely result in an accurate, however this is unnecessary as plausible results can easily be achieve through rather simple methods. In figure 3.4 you can see the distribution of trees based on an algorithm defined in [10], were the slope and

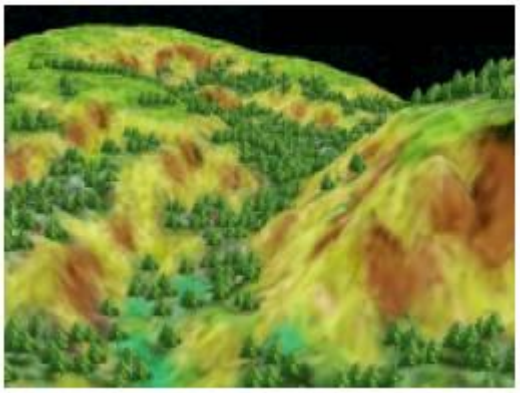

Figure 3.4: Plant Distribution based on Slope and Altitude

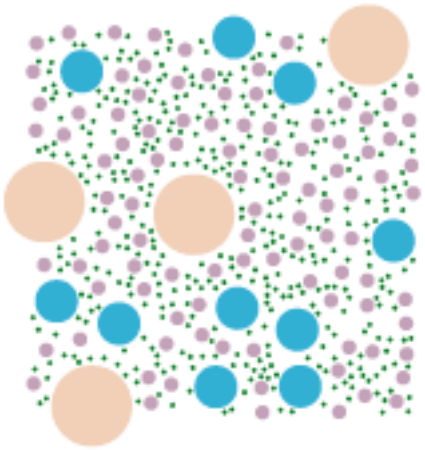

Figure 3.5: Plant Distribution using Size and Proximity Constraints

relative height of the terrain determined the placement of the trees. To simulate the competitiveness between plant life, found where larger plants dominate the resources in its area, proximity constraints are simply placed to prevent other plant life being placed too close. This method was purposed in [12] and an example can be seen in figure 3.5. There are also some other natural entities, such as stones, which can use similar distribution algorithms as plants.

L-systems or Lindenmayer systems have been central to plant geometry modeling since they were first used together in [21]. The central concept of L-systems is that of rewriting, which is a technique for defining complex objects by successively replacing

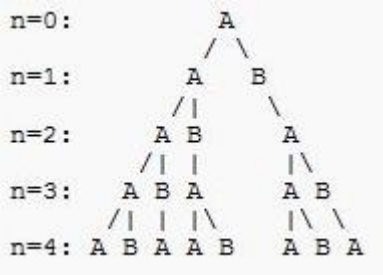

Figure 3.6: L-System Example

Figure 3.7: Plant Generated form L-System

parts of a simple initial object using a set of rewriting rules or productions. For an example using strings, say we have the initial string "A", where we use the production rules (A $\rightarrow$ AB) and (B $\rightarrow$ A). In figure 3.6 we can see the results of these production rules after n iterations. This simple concept is expanded upon to create complex plant structures such as the one in figure 3.7.

3.2.4 Urban Modeling

The modeling of such environments is usually undertaken in two steps. The first to generate a suitable road networks and the second is to populate the remaining space with various buildings.

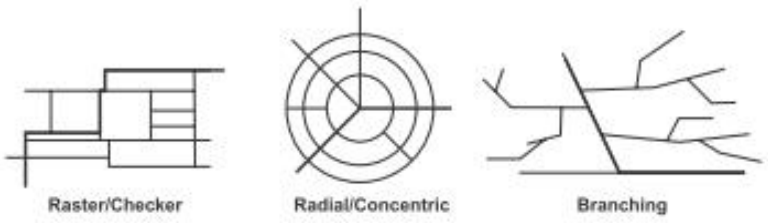

Figure 3.8: Frequent Road Patterns

[19] introduces an efficient way to procedurally model complete cities. Natural looking road networks are first constructed using context-sensitive L-systems. Using a small set of statistical and geographical input data the generator can produce realistic results. This input data, represented in various maps for population density and land gradients, provide the context-sensitive L-system with vital information. Using the input data roads can be attracted to areas of high population density or avoid mountainous areas. Another important goal during road generation is the compliance with the dominant patterns in that area. These patterns are demonstrated in figure 3.8.

Once the road networks have been defined the remaining space is usually subdivided into lots which will eventually contain buildings or parks. [15]describes a novel shape grammar approach for the procedural modeling the exterior of buildings with high visual quality and geometric detail. The context sensitive shape rules allow the user to specify hierarchical shape descriptions therefore incorporating a high level of control which is usually absent in procedural methods. This method can be used to model a wide variety of building geometry, ranging from skyscrapers to sub-urban homes provided the users can provide accurate design rules. A case study was carried out were the ground plans and figures of selected building types from the excavation site at Pompeii were used to abstract 190 design rules. These were then used to model the complete city including the streets and placement of trees.

Such work could be used in a multitude of games. Its flexibility and control could easily provide game designers with the means of creating infinite worlds should they wish to do so.

3.3 Level Representation

During the production of games, the levels present within them often go through many representations before they are produced in the full 3D or 2D graphical outputs the consumers will ultimately view them as. Commonly the first representations are sketches, concept art and then whiteboards. After these, production of the final level begins and is usually constructed in a manner that aids its final form.

For example, with 2D games that use a tile-based layout, designers often start out with an empty 2D grid, be it square, hexagonal or triangle. They then choose from a discrete set of tiles which can be considered as building blocks and place them onto a grid structure. Each tile may also posses properties which effect gameplay making this a very efficient way of building interactive content. However the restrictions of the grid make this approach unattractive for many 3D games as its difficult to achieve realistic organic structures such as landscapes.

A geometric/freeform approach is usually used to construct such structures as it imposes very little restrictions on the designer. This is a common approach for nearly all 3D games. The only drawback of this method, is that it is usually takes considerable time constructing level geometry as tools can be incredibly complex requiring experience to use efficiently and get good results.

Graph structures are sometimes used to represent dungeon levels within games. With this approach nodes commonly represent rooms or junctions and edges represent corridors. This abstract approach is interesting as you can provide additional data to the nodes and edges which can effect how they are finally illustrated or in the case of this research, generated. If nodes could represent homogeneous areas such as terrain and urban districts and edges represent roads, rivers or other linear features, this structure could be used to specify large amounts of geometry with minimal input. Because of the 2D restriction, altitude data will have to be supplied to nodes and edges to aid with the generation of a 3D output.

3.4 Gameplay

In this section we examine how the geometry within a level can effect gameplay. This is important because when procedurally generating geometry, there may be things we

can tweak to aid certain gameplay elements. The most common gameplay element associated with procedural content is replayability. This is a direct implication of the randomness that is usually contained within most procedural approaches. The objective with randomly constructed levels is to remove the predictability which is commonly found in static levels upon replay. A common approach to dungeon level generators is to create a set of prefabricated elements, that are consistent with the game's narrative and randomly construct levels from these elements. However, even though the level structure is random, the individual prefabricated elements aren't and eventually become repetitive themselves. The technique of this method is reasonable, but its downfall is the limited number of prefabricated elements which still have to be constructed by hand. Approaches which are capable of creating random levels that are consistent with narratives are being researched, with such works like [17] using the the users actions within the game environment aiding the procedural content generation process.

The difficulty of a level may be significantly influence by the terrain. For example in driving simulation games or any other game were land transportation is a large part of gameplay. Within these games, the ability of the player to successfully navigate the terrain can be hindered if the terrain is difficult to traverse or has an unconventional design that confuses players. Also elements of smaller areas of the level may be strategically placed to promote certain gameplay tactic like the approach in [5]. The placement of these elements could be written into a procedure and then easily reused when generating extra content.

3.4.1 Drama and Immersion

Using procedural methods to create drama and immersion is an unexplored area of research but recently has been used to great effect in the survival horror game Left4Dead [3]. The entity known as "AI Director" within its engine procedural generates enemy populations whenever necessary to either increase or decrease the player's emotional intensity. This enables players to have different experiences during each gaming session and removes most of the predictability found in static games.

The game also employs various environmental and atmospheric effects to add additional drama and immersion[14]. These include using film techniques for adding details

to dark settings and manipulating lighting to enhances moodiness and suspense with the objective of creating dark, scary cinematic environments. Although in this case the atmospheric effects are achieved by shaders, they could easily be incorporated into a procedural level generator. For example generating levels that strategical place or limit the number of light sources.

Infinity: Quest for Earth uses procedural geometry extensively, generating entire galaxies of planets in which users are free to travel to and from seamlessly. This is possible as the planets are procedurally generated at runtime at varying resolutions depending on the users proximity. This is important as the players experience is never interrupted during gameplay by loading screens and therefore maintaining immersion within the game[28].

Chapter 4

Design

The approach taken for this project was to first decide on a suitable abstract level infrastructure. This should be flexible enough to allow a user to easily construct variable 3D environments. Its abstract nature would streamline the toolset for modifications and could be extensible to produce more varied types of 3D environments. The interface is then assembled around this and must enable users to intuitively construct a description of a 3D environment.

The next major component of the project is the procedural generator. Its purpose is to produce a 3D model which conforms to the description outlined by a user through the interface.

4.1 Abstract Level Representation

The structure that the user will create and modify to produce their 3D level will be a 2D graph. Users should find this easy to relate to as it represents a top-down view of the eventual 3D environment. Within the graph, nodes will represent homogeneous areas such as terrain and urban districts. Edges will represent roads, rivers or other linear features. In this research the eventual output is an exterior 3D environment, however it is possible to extend the system to also produce interior environments where edges represent corridors and nodes represent rooms and junctions.

Because of the 2D restriction, altitude data will have to be supplied to nodes and edges to aid with the generation of a 3D output but because the structure is suitably

abstract users can specify large amounts of geometry with minimal input.

4.2 Interface

As the abstract level representation resembles a graph, the primary interface will incorporate graph editing functionality. Secondary interface elements will be contextual depending on the selected component, be it an edge or a node.

4.2.1 Graph Construction

Sufficient graph editing tools must be implemented to allow the user to easily create and modify a graph structure.

This will mean users must have the ability to create nodes and edges where every edge must connect two nodes. Users may also remove edges and node. When a node is removed, any edges incident upon it must also be removed. When users select a node or edge, they must be highlighted and the secondary interface elements must be displayed to allow the user to edit the properties of the selected graph component.

4.2.2 Secondary Interface Elements

As the level representation is abstract, to be able to further define the level, additional properties must be added to the nodes and edges.

4.2.2.1 Edges

As stated earlier, edges represent linear structures. The user will able to specify what linear structure the selected edge represents as this will have ramifications during the procedural enhancement stage.

The most powerful editorial feature of edges is the ability to define the inclinations of the terrain which lay either side of the edge. Although this may seem simple it provides a great deal control to the user. This enables a user to place vast chasms, steep mountains or flat planes on either side of the edge. As currently the user is only viewing the abstract level representation, it is important provide visual aids to the users when defining properties that have large impacts on the end result such as

the inclinations. Therefore there should be a small graphic to inform the user how the properties will effect the outcome. This will allow the user to control the output with more accuracy.

By altering the edge's "Path Variance" property, the user can specify how much the eventual path will deviate. Supplementary to this is the property "Width Variance". This will control the intensity of the deviance. These are interesting controls as one could use them to influence the difficulty within of a game. This could be true for action games where the player would have to endure a longer path if the path and width variance properties were set appropriately. This could also be true for driving games.

4.2.2.2 Nodes

Nodes have a number of properties available to edit. Most of these relate to altitude as the graph is in 2D. Here the user may specify the maximum and minimum altitude of the surrounding terrain. Also the altitude of any edges which are incident upon the node may be specified. These altitudes can be viewed on small graphic to aid the user when altering them. The user can also specify the size of the area surrounding the node in which these properties effect. This should also represented visually.

It should also be possible to state if the area of land represented by the current node is attached to the main land mass. If the node is defined as detached, it will become a separate land mass i.e. an island. This feature could be used by the user simply for aesthetic reasons but could also be used to add platforming gameplay to the eventual level.

The user can also specify if the node hosts an urban environment. When selecting this property the user should also be able to specify how organised the additional roads within the urban area will be generated.

The user may also specify the vegetation density of the area occupied by the node. This should also effect urban areas. For example, a small density may represent trees planted along the sidewalks whereas a large density could be use to simulate a desolate, possibly abandoned environment which has been overgrown with various plant life.

4.3 Procedural Enhancement

The 3D model must be generated with the intention of it being used within a game. Therefore the generator must enforce conformity's such as ensuring defined paths are navigable and possibly promote replayability by adding random segments. Realistic environments must be achievable but considering games can be very creative, unrealistic results should also be achievable.

However, before the output can be generated any conflicts that may be present within the graph must be resolved. An example of such is intersecting edges which may be solved by creating bridges.

The output of this stage should be a 3D environment. To represent the terrain, a height map will be constructed. This is a very common method for representing terrain as it is very portable. This is due to the fact that it can saved as an image. Other elements within the 3D environment such as vegetation, bridges and urban buildings will be represented by individual 3D models.

4.3.1 Resolving Graph Conflicts

The first step in resolving conflicts will be detecting them. The main conflict which will require resolving is intersecting edges. There should be two possible outcomes to intersecting edges, a crossing or a bridge. Also, edges that are incident upon urban districts should be connected to the end of a suitable road that is generated within the urban zone.

4.3.2 Generating Terrain

The first thing to do when generating terrain is to separate the land masses so that any nodes that are defined as detached generate separate terrain. As stated previously the terrain will be stored as a height map, which is an image where every pixel represents an altitude. The generation process for this height map will entail parsing the pixels of the output and calculating their height values. This will include finding the nearest edge or node, then using its proximity, inclination and altitude to determine height.

4.4 Additional Details

These are the details implied through the non graph interface. These specify details to be contained in or around the nodes and edges. For example, defining a node as an urban area implies there is an internal road network and buildings. These details effectively give meaning to the components of the graph specified by the user.

4.4.1 Vegetation

As shown in the previous chapter, defining vegetation models is an expansive area of research, therefore an external library that generates vegetation models will be used [11]. This open source project uses L-systems to generate tree structures based on defined profiles. These profiles specify the type of tree structure to generate. The project also allows users to create their own profiles by using XML files to define them.

Using this library all that is left to be determined is the positions within the generated map to place the vegetation models. This should be done by generating a density map like the one in figure 3.5. This will be based on the vegetation density value defined within the nodes contained in the graph. The higher this value is, the smaller the proximity between the distributed vegetation, thus producing a denser outcome. Also vegetation will be restricted from being placed on terrain that has a steep slope much like the approach taken in[10].

4.4.2 Urban Road Networks

As seen in the previous chapter, it is a popular method to use L-systems to generate road networks. However because of the graph structure used in the rest of the map, a similar approach should be used for the urban zone. Also any edges that are incident upon the urban node should be treated appropriately. For example, edges that are defined as roads should be connected to a suitable road on the boundaries of the urban zone.

4.4.3 Buildings

The procedural generation of buildings is also a very large and active area of research. And because building models are dependent on the period of game setting it would be unrealistic to incorporate an expansive procedural modeling component for this project. Therefore the buildings generated here will effectively be place holders.

Chapter 5

Implementation

Two main components comprise the system which is purposed in this research. The first of which is the graph editor described in the previous chapter which allows users to construct a level description of outdoor 3D environment. This consists of basic graph editing tools with additional options to supply detail to the nodes or edges. The second component is the procedural generator which evaluates the data provided and produces the 3D out model.

5.1 Interface

As stated in the previous chapter, the abstract representation the user will be using is a graph. An example of which can be seen in figure 5.1. In figure 5.1 there are six nodes, one of which is selected and therefore highlighted in orange while the others remain gray. There are also four edges represented by the blue and yellow lines.

5.1.1 Graph Editor

To the right of the application display window, the graph editing options are displayed. These consist of "Create Node", "Create Edge", "Select" and "Remove" with the currently selected one highlighted in orange.

Adding nodes is as simple as clicking anywhere within the graph editing window. However nodes have an exclusion zone around them, preventing other nodes being added on top of them.

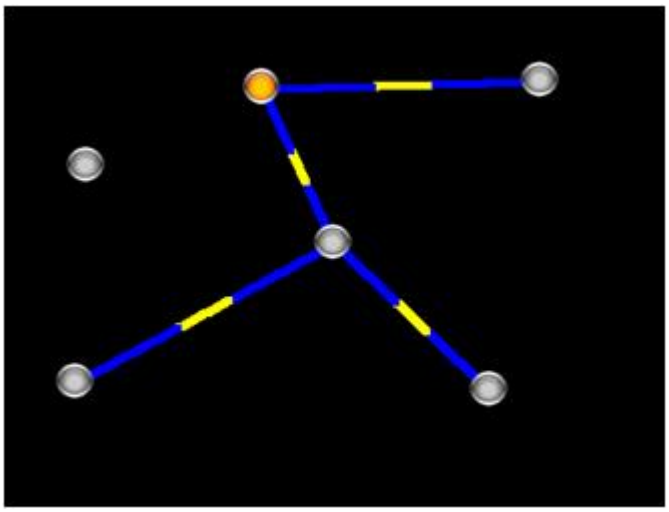

Figure 5.1: Example Graph

To add an edge, the user must first click on a node. Once this is done a temporary edge is added from the selected node to the cursor position. To cancel this the creation of this edge, the user may click the right mouse button. To complete the creation of an edge, the user must click on another node.

To select a node, the user simply clicks on the desired node. It will then be highlighted and the "Node Options" panel will appear below the graph editing window. When a node is selected it may be dragged to a new position and any edges that are incident upon it will be automatically updated.

To select an edge the user must click near the center point of the edge which is highlighted in yellow. This makes selecting edges easier when there are multiple overlapping edges. The select edge will be highlighted and the "Edge Options" panel will be displayed below the graph editing window.

Removing nodes and edges is done similarly to selecting except a second click is necessary to confirm the deletion. When a node is deleted, any edges that were incident upon it will be deleted also as edges must connect two nodes.

5.1.2 Node Options

When a node is selected, the "Node Options" panel will be displayed. Here the user can specify data relating to the selected node.

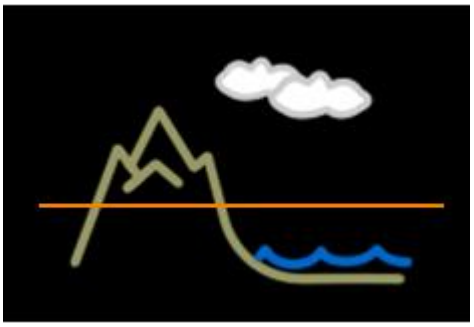

Figure 5.2: Altitude Visual Aid

The user can edit the various altitude options by using the appropriately labels slider. While altering these values, the user may observe a graphical aid next to the sliders indicating the data they are specifying. You can see this graphical aid in figure 5.2 where the altitude being specified is indicated by the horizontal orange line.

To the right of this panel there is another slider labeled “Area”. This indicates the effect of the various properties associated with the selected node. When this property is being altered there is also a visual aid shown on the graph to help the user accurately select their desirable area.

Above this there are two checkboxes labeled “Urban Area” and “Attached”. When “Urban Area” is selected, another checkbox appears allowing the user to specify the type of road network to generate for the urban area. Vegetation density can also be specified on this panel.

5.1.3 Edge Options

When an edge is selected, the “Edge Options” panel will be displayed. Here the user can specify data relating to the selected edge.

The most powerful options available on this panel are inclination sliders. These sliders specify the slope of the terrain which exists either side of the selected edge. To benefit the user, two visual aids are present when editing these options. The first is an added line drawn on the graph next to the selected edge. Depending on which inclination is being edited, the added line will be drawn on the appropriate side. The second visual aid is next to the sliders and can be seen in figure 5.3. where the inclinations are labeled “A” and “B”. This graphic represents a cross section of the

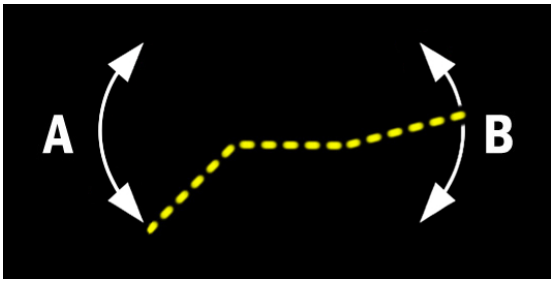

Figure 5.3: Inclination Visual Aid

eventual edge and accurately depicts the slope of the terrain at either side of the edge depending on the values of the inclination sliders.

Within this panel the user can also specify the type of edge. The user can select from "Road", "Path", "Land" or "River". These each have unique effects when the edge is being procedurally enhanced.

Located to the right of this panel is the "Path Variance" and "Width Variance" sliders. As described in the previous chapter, these sliders control the amount of meandering bends present on the edge and the extremity of these.

5.2 Procedural Enhancement

Using the minimal data provided through the various nodes and edges, it is now possible to procedurally enhance these into a fully 3D environment. Before the terrain can be generated, the graph most be prepared.

5.2.1 Graph Enhancements

To enable the user to go back and tweak their graph after viewing the output, a copy of their graph is made before it is enhanced and its conflicts resolved.

The first conflicts resolved are intersecting edges. Once they are detected, they are handled differently depending on their type and altitude. For instance if one edge is a river and the other is a road, two new nodes will be created either side of the river and the road edge will be split into two. Upon rendering the 3D environment, a bridge will be generated over the river and between the two new nodes. If however the two intersecting edges were both roads and the altitude of the two edges at which they

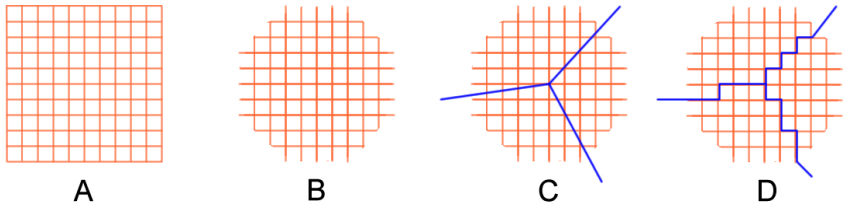

Figure 5.4: Urban Road Generation

crossed is within a small threshold, instead of a bridge, a crossing will generated. This involves creating one new node at the crossing, removing the existing edges and adding four new ones to replace these. The properties assigned to the new nodes and edges during these procedures are derived from the adjacent nodes and removed edges.

Road networks for urban nodes must also be generated. As mentioned in the previous chapter, it may be more practical to used a similar structure to the graph instead of implementing a context sensitive L-system such as the one in [19]. The approach taken in this research is rather simplistic but effective as it generates a structure which is easy to manipulate and use.

The first step is create a sub-graph consisting of a basic grid of edges and nodes as seen in part A of figure 5.4. The grid size should be based of the area property of the urban node. In part B of figure 5.4 any nodes that are beyond the radius of the urban node are removed. This gives the network a slightly more natural feel. In part C of figure 5.4. you can three edges which are incident upon the urban node. These edges are snapped to fit the grid like seen in part D of figure 5.4. And finally any edges that do not lay on the three edges incident, may be selected at random to be removed. This is also to give the output a more natural feel.

There is also an optional process which can be selected from the "Node Options" panel called "Distort". If selected, an additional process is added to road generation procedure. This basically takes every node within the sub-graph representing the urban zone and displaces it by a random amount along the X and Y axis. This is to mimic less organised road networks.

One of the benefits of using a sub-graph structure within urban zone is that you can also apply the procedures for creating bridges if a river has been placed running through the area.

It is at this point that various other details for the urban environment are defined.

As a detailed approach to procedural placement and modeling of buildings had already been defined in [19, 15] a simplistic approach was implemented so that focus could be placed on other features. Therefore only one building is placed in each block of the urban zone.

Positions for parked vehicles were also generate alongside the roads. This simple feature could be expanded to aid the generation of levels for multiple types of games. For example many action games, feature a cover system[1]. We could alter the positions of parked vehicle to ensure the player always has another position of cover to move to. Other objects that could be added to urban areas include traffic lights, bins, lamposts or any other objects that which follow distinct patterns. For example lampost can be places periodical along sidewalks while traffic lights can be placed at every road intersection.

As mentioned earlier, vegetation can also be placed within urban environments. It was planned that this could range from a few trees placed neatly along the side walks to a completely overgrown scenario. However the later proved to to be very unrealistic because of the limited set of structures that could be generated using the L-trees library. If it were possible to create vegetation structures that grew alongside buildings and walls maybe the result would look more realistic.

In non-urban areas, edges must be replaced by fully defined paths before terrain generation can begin. This is done in a few steps. First the edge is split up into a number of segments. The number of segments is determined by the "Path Variance" property of the edge. The higher the value of this property the more segments generated. Each segment has a random amount added or subtracted from it. This prevents the edge being evenly segmented which would yield an unnatural result. The next step generates one point for each segment and randomly places it to a side edge. The distance from these points to the edge is determined be "Width Variance" property of the edge. The higher the value of this property the greater the deviance from edge. If we were to join up these positions now with a straight line we would simply get a zig-zag pattern. To ensure a smooth and natural output Catmull-Rom interpolation is used.

[1] http://en.wikipedia.org/wiki/Cover_system

5.2.2 Terrain Generation

The terrain generated will be represented as a height map. Height maps are usually stored as textures and are a very common way to represent terrain and therefore will aid integration into other possible components of game engines. For example, most physics engine are capable of taking height maps and generating appropriate collision meshes.

Within the three colour channels of the height map image i.e. RGB, each will hold unique data to aid the visualisation of the eventual 3D terrain. The red channel will hold the altitude data. The green channel will hold property data i.e. river, road, terrain, etc. Incorporating these types of properties into the output will aid the terrain rendering process as they could be used to specify specific textures to be used. The blue channel will hold a value that will determine how much the altitude provided in the red channel can change. This is important when considering post-procedures that alter the surface of terrain to add detail and realism. For example users may want to apply erosion procedures such as those defined in [16], but they may want certain areas of the terrain such as roads or urban zones, to remain unaffected by the procedures. For these areas to remain unaffected, the value within the blue channel should be zero. This value increases as we move further away from roads and urban zones. This is to gradually introduce the effect to the area. This implementation doesn't apply any erosion procedures but does subject the terrain to ridged multifractal noise. This adds visual details to the terrain without compromising the structural integrity of the defined paths.

This value is also limited to the difference between the maximum and minimum defined altitudes of the area. This is to allow the user more control over the surrounding terrain. For example if the user wished to create flat planes such as the ones often found in dessert areas, they would specify the maximum and minimum altitude to be of similar altitude. The user can specify mountainous terrain by defining a large difference in the minimum and maximum terrain altitude.

The edges within the users graph will be the largest factor in the generation of the terrain. Their meandering paths will be carved into the terrain making them the most prominent features. This is justified as these are the areas the players will most likely see the most. The main influences on the terrain generation are the altitude of the

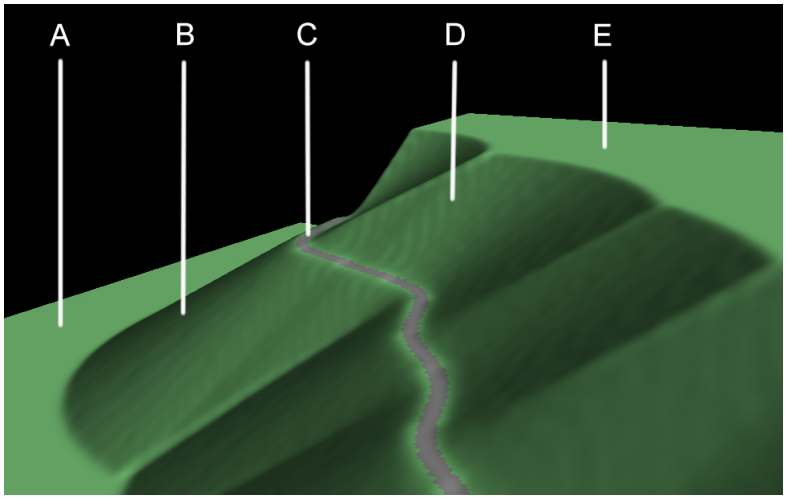

Figure 5.5: Terrain Generation

path and its inclinations. To calculate the altitude for a given pixel within the height map, we must first find the closet point on the path of the closet edge. The altitude of this point is then calculated by using the "Path Altitude" properties of the nodes at either end of the edge. If a pixel is within the radius of a path, it is assigned the "Path Altitude" value. This is labeled "C" in figure 5.5. To either side of a path, its inclinations are used to define the altitude of the terrain. This can be seen under labels "B" and "D" in figure 5.5. The terrain will continued to be sloped according to the inclinations until either the maximum or minimum altitude is reach. These areas are labeled "A" and "E" in figure 5.5. The same terrain with noise applied to it can be seen in figure 5.6.

5.2.3 Vegetation Distribution

This is represented by a density map like the one in figure 3.5 which is determined through the "Vegetation Density" value defined within the nodes contained in the graph. The higher this value is, there is an increased possibility of larger trees and the proximity between the distributed vegetation is reduced, thus producing a denser outcome. Once the density map is generated, it is compared to the users graph. Areas in which urban zones, rivers and roads exist are removed form the density map.

During the initialisation of vertices's for terrain visualisation, the positions within

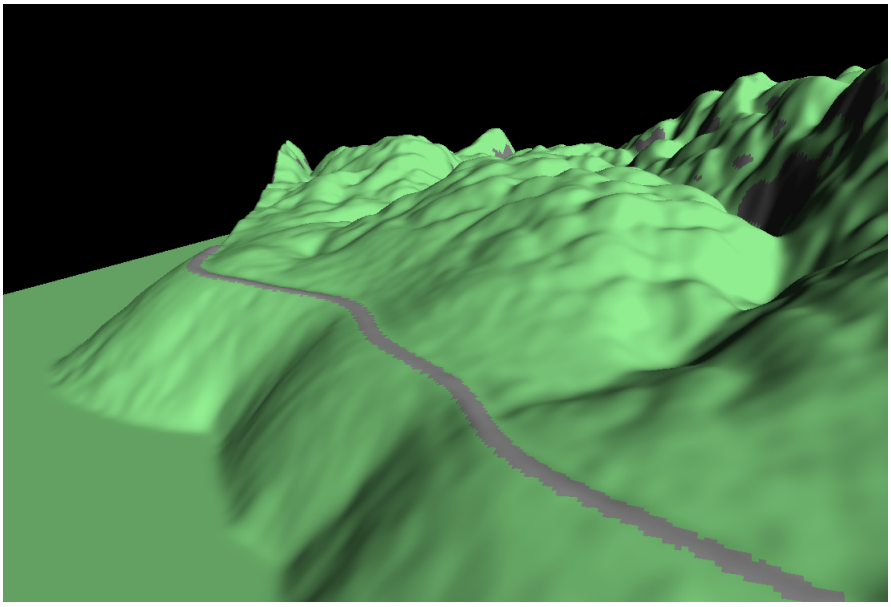

Figure 5.6: Terrain with Noise Applied

the density map are compared again. This time it's to the slope of the terrain. The slope can easily be determined by calculating the dot product of the normal of the nearest vertex and the unit Y vector (0,1,0). If the slope is greater than a defined threshold, vegetation is removed from this area. There is also an altitude threshold where vegetation can only be place above a minimum altitude, which may represent sea level, and below a maximum.

5.3 Random sub-graphs

Thus far the user can exercise a lot control over the resulting output. Unfortunately the generation process does not incorporate much randomness into the outcome. Therefore if this approach was used to create game levels, they would not promote replayability. This would be a missed opportunity as graphs are very flexible structures and there are multiple ways to incorporate random additions to them.

Graph grammar is one such way of expanding graphs. A graph grammar is defined as a tuple (A, P) where A is a nonempty initial graph and P is a set of graph grammar productions [22]. Much like the productions within L-system, graph grammar productions take either a single node or a graph and replace it with graph. Graph grammars

usually fall into two categories based on how productions are applied. These are algebraic node replacement systems and algorithmic node replacement systems. Both approaches are described in [24].

5.3.1 Replacement Procedure

In this research we will use an algorithmic node replacement systems to generate random subgraphs within the user defined graph. We will effectively parse the user's graph and replace certain nodes with small randomly generated subgraphs. Nodes that are eligible for replacement must not be urban nodes and have at least two edges incident which are either roads or paths. The properties of the nodes and edges contained within the subgraphs will be determined from the properties of the node being replaced and the edges incident upon it. This allows for the overall consistency and structure of users graph to remain intact. Therefore if the users graph was designed to accommodate a specific narrative, this will not be compromised.

The procedure in this research first creates a subgraph consisting of two nodes and an edge connecting them. Edges incident upon the node being replace will be redirect to one of these two nodes. The edge connecting these two new nodes is then subdivided into a number of edges. The number of edges is determined from the "Area" property of the node being replaced. With each subdivision an extra node is added to the subgraph to connect the new edges.

The nodes within this current linear structure will now each be put through a random graph grammar production. Available graph productions are shown in figure 5.7. These productions are not exact as the smaller nodes may be randomly placed on the opposing side of the dashed edge. It is possible to define further more complex productions or procedurally create the whole graph using a graph grammar similar to the one described in [1].

The final step to applying these subgraphs is to check for conflicts and resolve them.

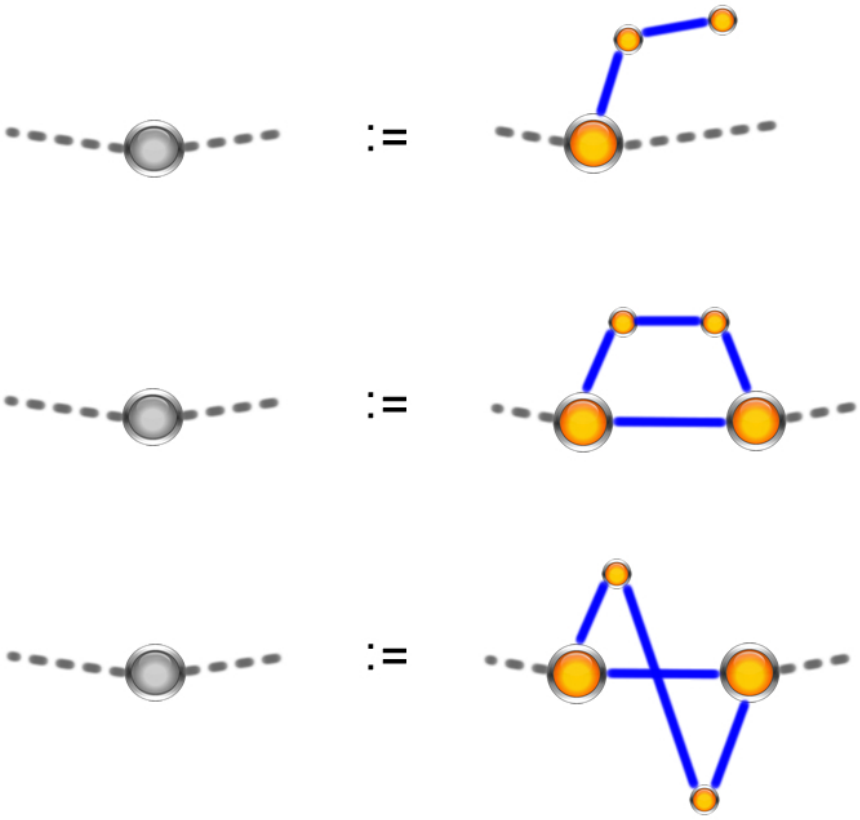

Figure 5.7: Graph Grammar Productions

Chapter 6

Evaluation

This research proposes and implements a procedurally aided approach for producing game content, specifically 3D exterior environments. Based on high-level user input in the form of a graph, this work in its current state provides automated terrain generation using a number of procedural methods. This resulting in an extremely quick way to create vast 3D exterior environments. While always keeping in mind ways to aid game development, the project incorporates a number of features to aid the rendering, navigation and further modification of terrain. To ensure the content generated can be used within games, paths which are key to player progression through the level remain unobstructed throughout the various processes.

Central to the success of this approach was the adaption of a suitable abstract interface which would allows users created and design descriptions of 3D environments. This interface had to be flexible enough for it to be used for multiple types of environments and as it has been used for interiors in the past, here it is applied it exterior environments which contain natural elements. Users can exercise sufficient control through the minimal interface which is simple to use and easy to understand. The can allow non-specialist personnel like consumers to effectively use the system.

As graphs are very malleable structures, they can be easily modified or extended. To enable this system to promote replayability, which has become increasingly important aspect to games, random subgraphs were introduced. Although there are a limited number of graph grammar productions the concept of adding random areas which

remain consistent with the surrounding designed structure, is very useful.

Although the procedural enhancement system fulfilled its purpose of integrating multiple procedural techniques into a coherent and useful output, its design does not promote expansion. For example, as demonstrated in Chapter 3, there are multiple procedures available for multiple content types. As games are very creative forms of media, users should not be restricted to using one process. The ability to select individual procedures for specific purposes would be ideal. However this may be unlikely as for the various procedures to be integrated into one system and compatible with others, their outputs and parameters would have to be standardised. This would also be necessary if a coherent and consistent interface was to be used.

However, the system implemented successfully integrates an number of procedural techniques which would be sufficient to successfully use the system to quickly prototype levels. This is a very desirable quality as it would enable faster testing and modifying iteration cycles during development.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This research presented a novel approach to the creation of 3D virtual worlds that integrates a variety of procedural techniques while incorporating elements to aid game development into the procedures. Through a simplified interface, users can define and control a minimal data set. The data can then be procedurally amplified to provide a fast and intuitive way for both experts and non-specialist designers to create vast virtual 3D environments. By allowing users to interact with an abstraction of the eventual output the design process is streamlined.

As the area of game development is a hugely creative area, their development tools must be extremely flexible and be capable of producing not just realistic results. It is for this reason procedural tools won't be used mainstream for quite some time but this research represents, if only on a small scale, that procedural methods can be flexible and a high level of control can be exercised over them. This allows users to get good results in a fraction of the time it might take to manually model the same environment. Therefore it is probable that we will see an ever increasing presence of procedural methods within content generation.

7.2 Future Work

Although this research was relatively successful there exist a number possible improvements that would make it more productive and feature rich.

With the introduction of subgraphs it could be possible for the user to specify huge amounts of geometry and still get incredibly fine detail. For example a graph hierarchy could exist where the highest tier represents a continent, but as the users specifies lower tiers of subgraphs the area represented is reduced to a point where the nodes and edges could represent rooms and corridors within a building.

If the framework was modified and several optimisations were done, such as implementing the geometry generation algorithm on the GPU via geometry shaders, it could be possible to view the output in real-time while editing the graph.

The obvious extension to the system would be to incorporate more generation methods such as erosion, increasing the variance of achievable outputs. However, when the algorithms are integrated into the interface, finding adequate parameters to standardise each method would be tedious.

7.3 Closing Thoughts

As for procedural methods becoming an alternative to manual modeling, this still remains unlikely. However this research brings it closer to a possibility. The advantages of procedural modeling are obvious in the speed and quantity they can produce results. And with this research a significant level of control is added. Unfortunately it still comes down to the desires of the users. If they require complete control over their 3D environments to ensure the success of there game, they have no option but to use manual modeling.

Also when creating complex procedures to generate content, one must weigh to complexity to the procedure to the amount of content that will ultimately use the produce. Although procedural methods may produce results quickly, time is still required to design and implement them well.

Because of the increasing demands in virtual world modeling, it is essential to further develop both forms of modeling and hopefully bridge the gap between them. In

doing so we could provide designers with the productivity gain of procedural methods, while still allowing them to exercise a fine level control. This research achieves this on a small scale but represents clear step towards making procedural techniques suitable for a variety of applications within game development.

Appendix

...

Bibliography

[1] Adams, D., and Mendler, S. M. Automatic generation of dungeons for computer games, 2002.

[2] Blythe, D. Directx futures. In *MS Meltdown* (2005), 3D Application Research Group ATI Research, Inc.

[3] Booth, M. Replayable cooperative game design: Left 4 dead. In *Game Developer's Conference* (2009).

[4] Brebion, F. Infinity: Quest for earth, 2010.

[5] Bungie. Environment design. Pressentation, 2008.

[6] Chiba, N., Muraoka, K., and Fujita, K. An erosion model based on velocity fields for the visual simulation of mountain scenery. *Journal of Visualization and Computer Animation 9* (1998), 185–194.

[7] Codemasters. Fuel, 2009.

[8] Deusen, O., S. Ebert, D., Fedkiw, R., Musgrave, F., Prusinkiewicz, P., Roble, D., Stam, J., and Tessendorf, J. The elements of nature: Interactive and realistic techniques. In *Siggraph* (2004).

[9] Gain, J., Marais, P., and Strasser, W. Terrain sketching. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), ACM, pp. 31–38.

[10] Hammes, J. Modeling of ecosystems as a data source for real-time terrain rendering. In *DEM '01: Proceedings of the First International Symposium on Digital Earth Moving* (2001), Springer-Verlag, pp. 98–111.

[11] Klasker, A. Xna procedural ltrees. Open Source Project, 2009.

[12] Lane, B., and Prusinkiewicz, P. Generating spatial distributions for multilevel models of plant communities. In *In: Proceedings of Graphics Interface* (2002), pp. 69–80.

[13] Miller, G. S. P. The definition and rendering of terrain maps. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), ACM, pp. 39–48.

[14] Mitchell, J. Connecting visuals to gameplay at valve. In *Montreal International Game Summit* (2008).

[15] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. Procedural modeling of buildings. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (2006), ACM, pp. 614–623.

[16] Musgrave, F. K., Kolb, C. E., and Mace, R. S. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (1989), ACM, pp. 41–50.

[17] Nitsche, M., Ashmore, C., Hankinson, W., Fitzpatrick, R., Kelly, J., and Margenau, K. Designing procedural game spaces: A case study. Tech. rep., Georgia Institute of Technology, 2006.

[18] Olsen, J. Realtime procedural terrain generation. Tech. rep., University of Southern Denmark, 2004.

[19] Parish, Y. I. H., and Müller, P. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 301–308.

[20] Perlin, K. An image synthesizer. In *Siggraph* (1985), vol. 19.

[21] Prusinkiewicz, P., and Lindenmayer, A. *The algorithmic beauty of plants.* Springer-Verlag New York, Inc., 1990.

[22] Rekers, J., and Schürr, A. A parsing algorithm for context-sensitive graph grammars. Tech. rep., 1995.

[23] R.M. Smelik, T. Tutenel, K.J. de Kraker, and R. Bidarra. Interactive creation of virtual worlds using procedural sketching. In *Eurographics* (2010).

[24] Rozenberg, G., Ed. *Handbook of graph grammars and computing by graph transformation: volume I. foundations*. World Scientific Publishing Co., Inc., 1997.

[25] S. Ebert, D., Musgrave, F., Peachey, D., Perlin, K., and Worley, S. *Texturing and Modeling A Procedural Approach*. Morgan Kaufmann, 2003.

[26] Smith, A. R. Plants, fractals, and formal languages. *SIGGRAPH Comput. Graph. 18*, 3 (1984), 1–10.

[27] .theprodukkt. .kkrieger. Breakpoint 96kb game winner, 2004.

[28] Wilkinson, O. Flavien Brebion. Online Interview, 2009.