

# Honours Project Report

## Using texture synthesis to generate bark textures with variation

---

*By Ryan Mazzolini*

**Supervised by Associate Professor James Gain**

### Mark breakdown

	Category	Min	Max	Chosen
1	Requirement Analysis and Design	0	20	0
2	Theoretical Analysis	0	25	0
3	Experiment Design and Execution	0	20	10
4	System Development and Implementation	0	15	15
5	Results, Findings and Conclusion	10	20	20
6	Aim Formulation and Background Work	10	15	15
7	Quality of Report Writing and Presentation	10	10	10
8	Adherence to Project Proposal and Quality of Deliverables	10	10	10
9	Overall General Project Evaluation	0	10	0
Total marks				80

Department of Computer Science  
University of Cape Town  
*2012*

## **Abstract**

The demand for 3D assets has grown considerably due to the availability and popularity of 3D applications and productions. Low resolution assets are needed for real-time rendering applications, such as computer games, while simulations and animated films require models at a higher resolution. This has led to a desire for procedural generation techniques that can quickly and inexpensively create 3D content. In particular, there has been a growth in the demand for procedural tree generation techniques, due to the many rendered environments that require model of trees. In a previous project, a system was developed that generates 3D models of trees from skeletons drawn by users in a sketch interface. The project presented in this report builds on that system by adding three features: surface subdivision, procedural leaf generation and bark texture synthesis. The focus of this report is the use of texture synthesis to generate seamless bark textures, exhibiting variation, from a sample image. This is achieved through the implementation of an example-based synthesis technique, commonly known as the discreet solver. The quality of the textures produced by this system were scrutinized in an experimental study. The results of this study indicate that the discreet solver has potential to fill all of the requirements, however future work is needed.

## **Acknowledgements**

We would like to thank Triggerfish animation studios for their support and for hosting us during our internship with them. During our time with Triggerfish we gained invaluable insight and experience into the field of 3D animation and computer graphics. We are grateful for the time and patience provided by all the staff during our short stay with them.

Secondly we would like to thank our supervisor Associate Professor James Gain. Prof Gain was instrumental in organising and overseeing the project through the planning and development. This also included the organisation of the internships with Triggerfish.

lastly we would like to thank all of the staff and students who have given up their time to participate in the evaluation of our research.

## Table of Contents

Abstract.....	2
Acknowledgements.....	2
Table of Contents.....	3
1. Introduction .....	5
1.1. Acknowledgements: Triggerfish.....	<b>Error! Bookmark not defined.</b>
1.2. Thesis Outline.....	<b>Error! Bookmark not defined.</b>
2. Background Chapter .....	7
2.1. Abstract.....	<b>Error! Bookmark not defined.</b>
2.2. Introduction .....	8
2.2.1. Defining texture synthesis .....	8
2.2.1.1. Textures.....	8
2.2.1.2. Texture synthesis .....	9
2.2.2. The uses of texture synthesis.....	9
2.3. Techniques in texture synthesis.....	10
2.3.1. Procedural Texture Synthesis .....	10
2.3.2. Texture Synthesis from Samples.....	10
2.3.2.1. Pixel-based methods.....	10
2.3.2.2. Patch-based methods .....	12
2.3.2.3. Tiling-based methods.....	13
2.4. Tabulated Comparison of texture synthesis methods.....	14
3. Requirements.....	15
3.1. Accuracy.....	15
3.1.1. Variation.....	15
3.1.1.1. Different Tree Species.....	15
3.1.1.2. The same tree species.....	16
3.1.2. Seamlessness .....	18
3.2. Performance .....	19
4. Foundations .....	19
4.1. Texture synthesis .....	20
4.1.1. Texture Optimization .....	21
4.1.2. The Discreet Solver .....	23

4.1.2.1.	K-coherence Analysis Phase.....	23
4.1.2.2.	K-coherence Synthesis Phase.....	24
4.2.	Surface Texture Synthesis.....	25
4.2.1.	The Discreet Solver .....	25
5.	Implementation .....	26
5.1.	System Overview.....	26
5.1.1.	Integration into the existing GUI.....	27
5.1.2.	Workflow for the texture synthesis .....	27
5.2.	Texture synthesis implementation .....	28
5.2.1.	Textures.....	29
5.2.1.1.	Coherence textures.....	29
5.2.1.2.	The Indexed textures .....	31
5.2.2.	The synthesis method.....	32
5.2.3.	The discreet solver .....	32
6.	Evaluation .....	33
6.1.	Expert User Testing.....	34
6.1.1.	Design.....	34
6.1.2.	Procedure.....	34
6.1.3.	Results.....	34
6.2.	Final User Testing.....	35
6.2.1.	Design.....	35
6.2.2.	Procedure.....	38
6.2.3.	Results.....	39
6.3.	Performance Testing.....	40
6.3.1.	Design.....	40
6.3.2.	Procedure.....	40
6.3.3.	Results.....	41
7.	Conclusion.....	43
8.	References .....	45
Appendix A.....		48
Appendix B.....		49

## **Chapter 1. Introduction**

The advances in technology, specifically in information technology, have changed the way people interact with each other and the world around them. This is clearly evident in the creation and distribution of multi-media. Multi-media, such as video and computer games, have become ubiquitous and interactive through the aid of computer graphics. The rapid development and changes within the field of three-dimensional (3D) computer graphics and animation have created an expectation for highly realistic 3D models and environments. Creating these detailed and accurate 3D assets manually, however, is expensive and laborious. Models of trees are particularly expensive to create due to the quantity and variation of trees needed in environments (e.g., a forest of trees). This has led to the development of techniques to generate tree models procedurally.

Recently programs have been developed with the intention of procedural tree generation. Two examples are the commercial product SpeedTree [1] and the open-source project TreeDraw [2]. A procedurally created asset is defined as an asset which is created through a process of steps. The asset is created once all of the steps have been completed. The aim of this project, otherwise known as Yggdrasil, is to enhance the procedurally generated trees developed through TreeDraw.

### **1.1. Research question**

Research questions have been created to measure the success of project Yggdrasil. The success for each hypothesis is determined through performance and user testing. For the entire project to be successful each of these research areas need to solve their specific requirements. The requirements for each section are stated for each research question and are discussed further in the Yggdrasil overview in Section 1.3.

#### **1.1.1. Can texture synthesis add variation to generated bark textures in procedural generated tree models?**

A technique for synthesising bark textures needs to be found that can generate bark textures for each model produced. These textures need to be tileable without noticeable seams. The bark texture synthesised should also resemble a sample image or structure, while still being unique. Development for the synthesis method should be iterative to allow regular testing.

#### **1.1.2. Can multiple realistic leaf textures be generated from a sketch-based interface?**

A main requirement for the new system is to be able to create tree models with the inclusion of leaves. The leaves produced should be recognisably similar to a user generated sketch of a leaf outline.

#### **1.1.3. Can the branching structure be more realistically modelled by subdivision surfaces?**

The branching structure of a tree model should be smooth and not contain any unrealistic features. This is most noticeably seen where the branches diverge, specifically at the joins between branches.

## **1.2. Project TreeDraw**

Project TreeDraw provides a method of producing procedurally generated trees with variation. A sketch interface is used to capture the structure of a tree through user generated sketches. An L-System is then created from this sketch to generate 2D trees with a similar structure to the sketch. This L-system is then interpreted and converted from a 2D structure of a tree into a 3D tree model.

### **1.2.1. Sketch Interface**

The sketch interface for TreeDraw acts as the front end of the system and the graphical user interface(GUI). The sketch interface interprets a series of mouse movements and converts them into information pertaining to a 2D model of a tree structure. From the tree structure obtained, trees with a similar structure can be created. The sketch interface is specifically designed to be simple and easy to use, this aids the speed using the program and creating tree models.

### **1.2.2. L-Systems**

TreeDraw uses L-systems to take the sketch made, using the sketch interface, to develop similar 2D tree models. Lindenmayer systems or L-systems are similar to formal grammars, consisting of an initial object and a set of productions [3]. In a formal grammar parts of the initial object are changed recursively by the productions. L-systems differ to formal grammars by applying productions in parallel and simultaneously changing the whole initial object. This difference has made L-systems predominantly used for modelling cell divisions in molecular organisms, where many divisions may occur at the same time. L-systems are therefore, naturally suited for the procedural generation of plant and tree models.

### **1.2.3. 2D-3D model converter**

The model converter is designed to derive a 3D model from the L-system in a realistic time (roughly faster than thirty seconds). This is achieved by directly transforming the 2D sketch into three-dimensions. The branches are then rotated uniformly around the main trunk, attempting to maximise the distance between each branch. This portrays realistic branching behaviour, as trees naturally space their branches out in order to capture the maximum amount of sunlight. The resulting model is then rendered and displayed to the user for analysis.

## **1.3. Yggdrasil Overview**

Project Yggdrasil aims to provide enhancements to TreeDraw in order to increase the realism of generated trees. This includes the introduction of new generation features, more variation and the removal of visual artefacts. The three main areas of development include: the branching structure, bark textures and the addition of leaves.

### **1.3.1. Branches**

The existing system generates the 3D tree model with the use of cylinders. Generalised cylinders are placed together, intersecting at joins of branches. This can be extended by creating a single mesh from all of the individual cylinders and smoothing the join areas between branches. Both of these can be solved through the use of subdivision surfaces.

### 1.3.2. Leaves

TreeDraw did not have any mechanism to create leaves for the generated trees, so adding leaves is an obvious area to improve the system. The new process for creating leaves resembles the generation of trees models, by allowing the user to sketch the shape of a leaf to be created. The leaf is then created through the process of venation [4]. Nodes are placed within the outline of leaf shape, close to the stem, and veins are grown toward them. Once veins have been connected to these nodes, more veins are grown toward nodes that are placed further away from the stem. This process continues until the vein system has been created over the entire surface of the leaf. Finally the leaf is coloured according to the vein system and placed on the tips of the branches.

### 1.3.3. Bark Textures

The current system uses a single tiling texture for each tree model generated. These textures have to be seamless and manually added for each tree being generated. This often requires textures to be sourced from image libraries, which have images that are already made to be tiling and seamless. Another option is to use an external program to make an image seamless, which can often introduce unwanted or unnatural results depending on the type program and image used. Environments requiring many different trees of the same species can also add further complications. This requires textures that are similar in structure, while not being exactly the same. For example noticeable repeating patterns can be found easily in a forest of trees, each using the same bark texture.

For these reasons a feature for creating bark is added to the system. This feature uses an example-based texture synthesis technique to generate seamless textures similar to a sample image. This technique, commonly known as the discrete solver, fulfills all of the criteria for generating bark textures (as discussed in chapter 3). It is capable of developing multiple

## 1.4. Thesis Outline

The structure for the remainder of this thesis is as follows:

- Chapter 2 provides background information on texture synthesis, reviewing different techniques.
- Chapter 3 motivates the set of requirements for the texture synthesis of bark.
- Chapter 4 investigates the chosen synthesis technique including the history of its development.
- Chapter 5 explores the specifics of the implementation for selected technique.
- Chapter 6 provides the design, procedure and results for the various rounds of user testing run over the lifespan of the project.
- Finally chapter 7 provides a conclusion and future work for the thesis.

## Chapter 2. Background Chapter

Texture synthesis is a method of computationally creating synthetic textures. It is important for applications in computer graphics, computer vision and image processing. It is a large field containing a variety of solutions and techniques. This makes finding a synthesis technique to fit a desired application

a daunting task. This chapter aims to explore existing techniques for texture synthesis and its applications, highlighting strengths and weaknesses.

## **2.1. Introduction**

Texture synthesis is a large field with many competing techniques and a variety of applications. This makes it an important and active field of study. In this chapter we define what texture synthesis is, investigate the applications of texture synthesis and explore existing techniques [5]. Texture synthesis is used in many different areas and for different applications. It is predominantly important in computer science applications like computer graphics, computer vision and image processing. The uses, however, include many applications outside of computer science.

### **2.1.1. Defining texture synthesis**

#### ***Textures***

A texture in computer graphics is an image that is wrapped around a three dimensional model through texture mapping thereby providing surface properties for the model. Texture mapping is a technique that provides a way of representing surface details without modifying a models geometry or material properties. The mapped images can describe many details, including colour, reflection, transparency and displacements [5]. However, in practice a texture is most often restricted to a colour image with repeating structure such as fabric, as illustrated in Figure 1. In this thesis we will focus on this narrower definition of a texture.

Textures can be created manually by texture artists. However, this is labour intensive, taking long periods of time. The larger the texture is the more time it takes to make a realistic and consistent looking texture. Some types of textures such as photo-realistic textures are simply too complicated to create by hand. This is why the field of automated texture generation has been investigated so thoroughly.



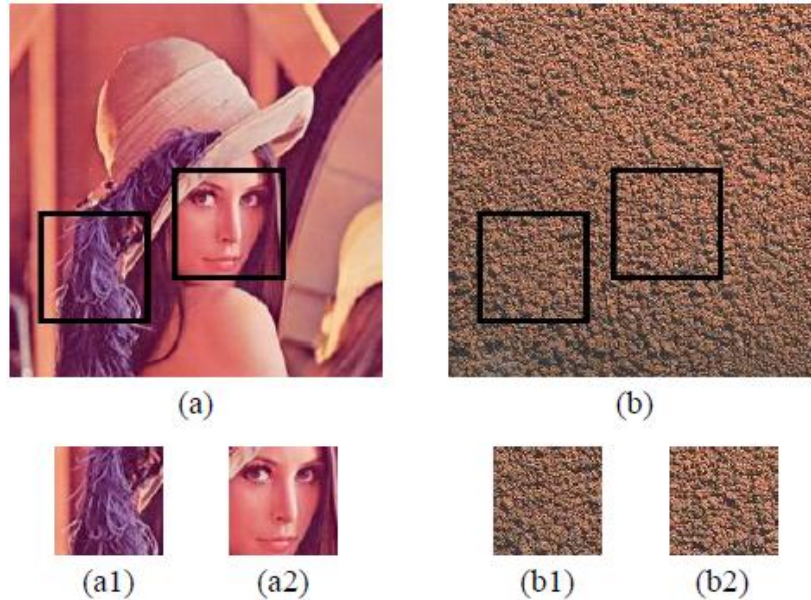


Figure 1. How textures differ from images (adapted from Wei and Levoy [6]). (a) is a general image while (b) is a texture. A movable window with two different positions is drawn as black squares in (a) and (b), with the corresponding contents shown below. Different regions of a texture are always perceived to be similar (b1,b2), which is not the case for a general image (a1,a2). In addition, each pixel in (b) is only related to a small set of neighboring pixels. These two characteristics are called stationarity and locality, respectively.

### *Texture synthesis*

Texture synthesis is the creation and placement of textures through computation and procedural generation. Creating textures that are perceived by humans as realistic is a very hard problem to solve. Textures need to be consistent, for example by not containing visual artefacts such as straight lines that are skewed. Textures also need to be perceived as naturally random and not just as reoccurring sections. While synthesising realistic looking textures might be possible, having them synthesised in a feasible time period is also essential. Users do not want to have to wait for textures to be synthesised for an impractical amount of time. Some applications even require textures to be synthesised in real-time, which puts significant constraints on the complexity of the process.

#### **2.1.2. The uses of texture synthesis**

Texture synthesis is important for many applications that are reliant on computer-generated environments. These include animated films, games and the construction or designing of environments. The environments often need objects that are wrapped in seamless textures which appear realistic. How realistic these textures are is often reliant upon how randomised the texture looks and if there are obvious visual artefacts or patterns. However, this does vary by texture type as some need to be more randomised (e.g., natural looking textures like bark) and some need to be more regular (e.g., material textures like cotton). Texture synthesis is also important for areas outside of computer-generated environments, such as, in medicine and other scientific fields that apply computer graphics, computer vision and image processing.

## 2.2. Techniques in texture synthesis

In the area of texture synthesis various techniques exist for the creation and placement of textures. These include texture placement, procedurally-generated textures and example-based texture synthesis.

### 2.2.1. Procedural Texture Synthesis

A procedurally-generated texture is a texture generated using an algorithm. Similarly procedural texture synthesis is a method of creating synthesised textures via computation and procedural means. The algorithms used in the creation of procedural textures are usually based on fractal noise or turbulence functions. Examples of these techniques are the Perlin [7] and Worley [8] three-dimensional noise techniques which generate random colours. These are particularly useful in modelling natural "randomness" such as for waves or marble. Procedural texture synthesis is limited by its reliance on the programmer. Each texture requires a programmer to write and test the procedural code until the resulting texture looks correct [9]. For these reasons this report will not focus on procedural texture synthesis, but rather on texture synthesis from samples.

### 2.2.2. Texture Synthesis from Samples

As we have seen from section 2.2.1 procedural texture synthesis techniques have a strong reliance on the programmer creating code for each class of desired texture. This is where example-based texture synthesis comes into its own. Instead of procedurally generating a texture, a small sample image is used to synthesize larger texture areas in the same style. Texture synthesis using example-based methods is divided into three categories; Pixel-based methods, Patch-based methods and Tile-based methods [10].

#### *Pixel-based methods*

Pixel-based methods of texture synthesis create the synthesized image pixel by pixel. Each generated pixel is based on its neighbourhood of surrounding pixels. The example image is searched to find the closest match to the neighbourhood of a particular pixel. The resulting area is then used to formulate the pixel. The search will then continue for the next pixel. To make the synthetic image original, the methods will often be based on a noise image in the synthesis process. The main advantage of pixel-based methods is the ability to control textures at the pixel level. These methods are, however, very sensitive to the size of the window in producing adequate results and also do not cope well with strong features (eg., cracks). [10]

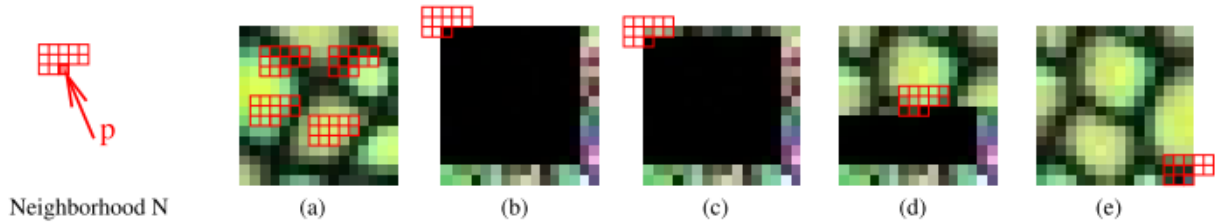


Figure 2. The algorithm by Wei and Levoy [6]. (a) is the input texture and (b)-(e) show different synthesis stages of the output image. Pixels in the output image are assigned in a raster scan order. The value of each output pixel  $p$  is determined by comparing its spatial neighborhood  $N(p)$  with all neighborhoods in the input texture. The input pixel with the most similar neighborhood will be assigned to the corresponding output pixel. Neighborhoods crossing the output image boundaries (shown in (b), (c) and (e)) are handled toroidally. Although the output image starts as a random noise, only the last few rows and columns of the noise are actually used. For clarity, we present the unused noise pixels as black. (b) synthesizing the first pixel, (c) synthesizing the first pixel of the second row, (d) synthesizing the middle pixel, (e) synthesizing the last pixel.

Efros and Leung [11] develop a system that uses non-parametric sampling to grow a texture out from a centre pixel in an image. Neighbourhoods of similar pixels are found that match the square window of pixels already grown around the pixel being analyzed. These neighbourhoods are found to be similar using a Markov Random Field (MRF) model that measures the probability brightness values of a pixel given the brightness values of its neighbourhood. Similar neighbourhoods are chosen at random and are used to create each new pixel. The technique employed by Efros and Leung is a general approach, but is computationally expensive and often grows portions of the texture that do not reflect the structure of the sample image. Other neighbourhood variations are used in methods such as the proposed solution by Wei and Levoy [6] and Turk [9]. Wei and Levoy used a raster scan line method and a smaller neighbourhood of recently generated pixels, this is shown in Figure 2. They also speed up Efros and Leung's method by using tree-structured vector quantization. This creates a binary-tree-structured codebook which can be used for nearest-point queries. Turk's method synthesises directly onto a model by sweeping over a generated vector field that is used for indexing instead of the two dimensional indexing usually used. This is illustrated in Figure 3. By directly synthesizing on a model, boundary seams and artefacts created from wrapping an image around the model are reduced. Turk used both the square neighbourhoods used by Efros and Leung, and a half square neighbourhood. These neighbourhoods are used because multiple sweeps at different resolutions over the model are need for best results.

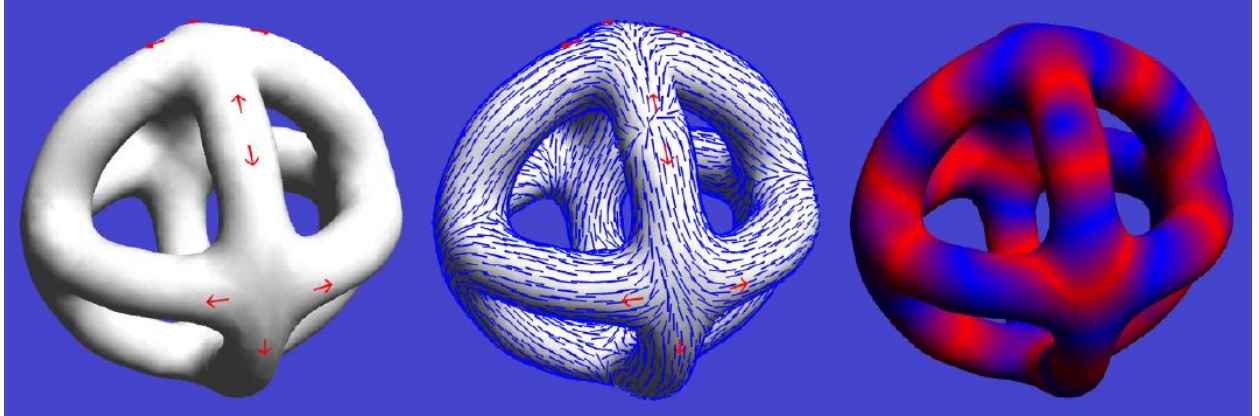
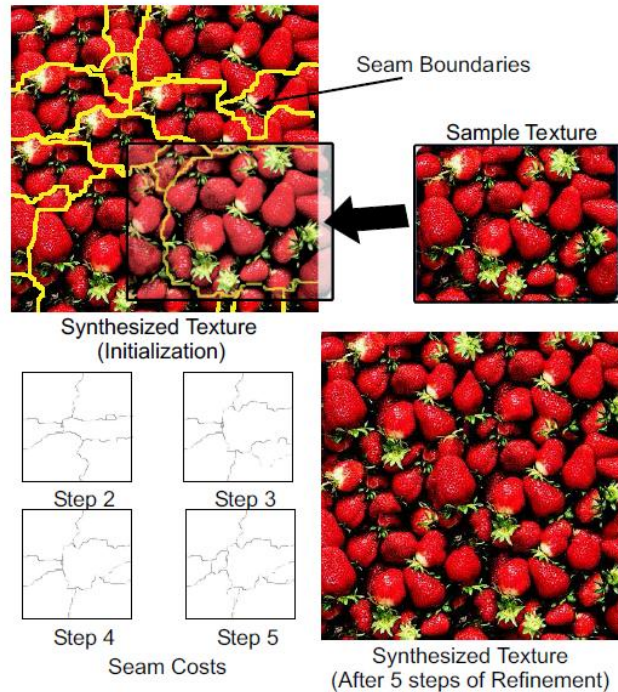


Figure 3. The vector field algorithm by Turk [9]. User's orientation input (left), the interpolated orientation field (middle), and the sweep values shown using colour cycling (right).

For increased quality Wei and Levoy's as well as Turk's methods use a multi-resolution technique. This technique is a multi-resolution pyramid based on a hierarchically statistical method. Another method introduced by Ashikhmin [6] uses only the pixels around the targeted pixel for its neighbourhood to reduce the blur effect found in Wei and Levoy's method. This technique was also much faster than Wei and Levoy's even without the tree-structured vector quantization. Ashikhmin's method produces good results for natural samples but can also contain obvious artefacts from some samples.

### ***Patch-based methods***

With patch-based algorithms a texture is generated patch by patch. These patches are chosen through the use of cutting paths. These cutting paths are created in the attempt to minimise errors due to overlapping[10]. A method used by Efros and Freeman quilts together random overlapping blocks according to a minimum cost path across the overlapped area. This obtains good results at low computational cost. Another method by Liang et al. [12] uses patches sampled according to a Markov Random Field density function. This is fast and real time, however, errors and artefacts are created regularly.



**Figure 4.** This figure illustrates the graph-cuts approach by Kwatra et al.[13] showing the process of synthesizing a larger texture from an example input texture. Once the texture is initialized, new patch locations are found iteratively to refine the texture. Note the irregular patches and seams.

Kwatra et al.[13] introduced a graph-cut approach that computes seams of patches as shown in Figure 4. This technique is useful for partially structured textures and can create good quality images from by the patches. A problem with the above patch-based technique is that the resulting textures are all regular. Liu, Lin and Hays [14] put forward their deformation field solution where they view near-regular textures as statistical departures from a regular texture along different dimensions. While this is a novel approach their method can only create near-regular textures. A Directional Empirical Mode Decomposition-based texture synthesis algorithm has been proposed by Y. Zhang et al. [15], which decomposes the sample texture into a series of Intrinsic Mode Functions (IMF) images in the inherent direction of the texture image. This technique manages to maintain the features of the sample image while synthesizing at a high level.

Patch-based methods often manage to preserve the global structure of the sample texture much better than pixel-based techniques. However, they sometimes produce local artifacts on the overlapping regions and do not keep the local consistency as well as pixel-based methods. The search for good patches and cutting paths is not trivial and will often require iterative improvement [10].

### ***Tiling-based methods***

In tiling-based methods tiles are pre-computed and placed seamlessly together to create a texture. This makes synthesizing a large texture very fast[10]. However tile based methods often fall victim to similar artefacts as patch-based methods.

The first of these techniques is the Wang Tiles method [16], which uses a set of tiles to synthesize on the fly. This was proven to be highly efficient for real-time applications with memory constraints. This technique, however, suffers from artefacts created from the corner, joint and sampling problems. Recently,  $\omega$ -tiles having been introduced by Ng et al. [17] with a higher quality of pre-computed tiles and smaller tile sets than Wang tiles. This has been shown to produce tiles of reasonable quality while only using 16 tiles as opposed to 64 tiles generated for Wang Tiles. This means that it is even more efficient than Wang tiles with respect to computation and memory. A method introduced by Zhang and Kim [10] aims at reducing the artefacts produced by Wang Tiles. They remove the key artefacts using graph cut and sampling techniques. The tile set is reduced from that of Wang Tiles, however, the method requires greater pre-computation in developing the tiles.

### 2.3. Tabulated Comparison of texture synthesis methods

Table 1 is a tabulated comparison of different texture synthesis techniques for comparison. This is extended from Y. Zhang et al. [15] to include Tile-based techniques. Each technique is compared indicating the type of technique it is, if it is a multi-resolution technique, what the quality of the output image is like as well as the speed of the technique. The synthesis quality column compares the quality of the synthesised image from the best results "High" to results containing noticeable artefacts " Good " or large noticeable artefacts " Medium ". The speed of synthesis is quantified into three categories, "Medium" which runs in a few seconds, "Fast" running much faster than "Medium" but is not fast enough to be useably interactive and "Real-time" which runs fast enough for interactive synthesis.

Method	Synthesis method	Multi-scale synthesis	Synthesis quality	Synthesis speed	Synthesis type
Wei and Levoy [6]	Pixel-based	Yes	Good	Medium	General textures
Ashikhmin M[18]	Pixel-based	No	Good	Fast	A wide variety of textures
Liang L, Liu C, Xu Y, et al. [12]	Patch-based	No	Medium	Real-time	Natural textures
Efors AA, Freeman WT [19]	Patch-based	No	Good	Medium	General textures
Kwatra V, Schödl A, Essa I, et al.[13]	Patch-based	No	High	Fast	Partial structural textures
Liu YX, Lin WC, Hays J.[14]	Patch-based	No	Medium	Fast	A wide variety of textures
Lefebvre S, Hoppe H. [20]	Pixel-based	Yes	High	Real-time	Near-regular textures
Kwatra V, Essa I, Bobick A, et al. [21]	Pixel- and patch-based	Yes	High	Fast	A wide variety of textures
Zhang Y et al.[15]	Patch-based	Yes	High	Fast	A wide variety of textures
Cohen et al. [16]	Tile-based	No	Medium	Real-time	A wide variety of textures
Ng et al. [17]	Tile-based	No	Medium	Real-time	A wide variety of

					textures
Zhang and Kim [10]	Tile-based	No	Good	Real-time	A wide variety of textures

**Table 1. Method comparison for texture synthesis techniques as adapted from Y. Zhang et al. [15]. The synthesis quality column compares the quality of the synthesised image from the best techniques. The speed of synthesis is quantified into three categories, "Medium", "Fast" and "Real-time".**

The most highly regarded synthesis techniques for quality of synthesis are the two competing methods by Kwatra V, Essa I, Bobick A, et al. [21] Kwatra V, Schödl A, Essa I, et al.[13]. These methods are the texture optimization and graph-cuts algorithms. The fastest algorithm, which also produces high quality results is the parallel technique by Lefebvre S, Hoppe H. [20].

## Chapter 3. Requirements

This chapter explores the features required to enhance textures in the existing system. The current structure uses a single, predefined texture which is tiled across the surface of the tree model. The use of a single tiled texture introduces seam inaccuracies which can be improved through the use of texture synthesis methods (as reviewed in the background chapter). The new texture synthesis method should:

- Improve the accuracy of the of the texture covering the surface. This can be done by removing seams over the surface and providing variation between trees and over the surface.
- Provide the synthesised texture within an appropriate time so as to integrate smoothly into the existing system.

### 3.1. Accuracy

Precision is very important in the computer graphics industry in order to improve the realism of generated models. Models in 3D applications undergo considerable scrutiny from users viewing and interacting with the application. This can range from watching an animated film to engaging with a 3D environment in computer simulations and games. It can also vary between pre-rendered images to real-time rendering that requires up to 60 frames per second. Rendering is the process of capturing and representing environmental information from a specific point of view in a modelled environment. The information captured is used in the creation of frames or images.

Improving the accuracy in the generation of bark textures can be divided into two main sections: variation and seams. .

#### 3.1.1. Variation

In the pursuit of improving tree bark variation, there are two focus areas: differentiation in the types of tree species through texture variation and the improvement of bark diversity across single tree models.

#### 3.1.2. Different Tree Species

When simulating a natural 3D environment, the inclusion of a range of tree species can be integral in manufacturing authenticity for the viewer. Each type of tree demonstrates unique characteristics

in branch and bark structure, which when emulated, perpetuates the perception of reality. The types of bark can vary from the thin papery scales of the Silver Birch, to the thick ridges of the Oak tree, to the thin plates of the London Plane (as seen in Figure 1). This diversity in the bark provides a challenge when trying to replicate each type accurately. By modelling each bark type individually, highly accurate copies of the original bark structure can be created. As discussed in the previous chapter, this is most commonly achieved through procedural generation as well as very specific example-based texture synthesis methods. However, due to the vast quantity of tree and bark species, a procedurally generated system is unfeasible. This is because creating a robust system for every species is an almost impossible task. Each bark structure provides unique challenges that require specific attention to be modelled correctly. This could be addressed by a system with many different parameters, however, parameters are often slow to use, overwhelming to new users and difficult to manage.



**Figure 5.** This image shows examples of bark from three different tree species; Silver Birch bark (Left)[22], Oak bark (Middle)[23] and London Plane bark (Right)[24].

The new system should ideally be able to model a wide range of bark. To accomplish this a general synthesis technique is required. The technique should not need to be manually modified for each bark type. Typically, the synthesis techniques that are the most general are example-based texture synthesis techniques. By providing a sample of the bark structure to be replicated, the example-based synthesis techniques can automatically analyse the sample for synthesis. This removes the need to manually modify the technique for each bark type.

### **3.1.3. The same tree species**

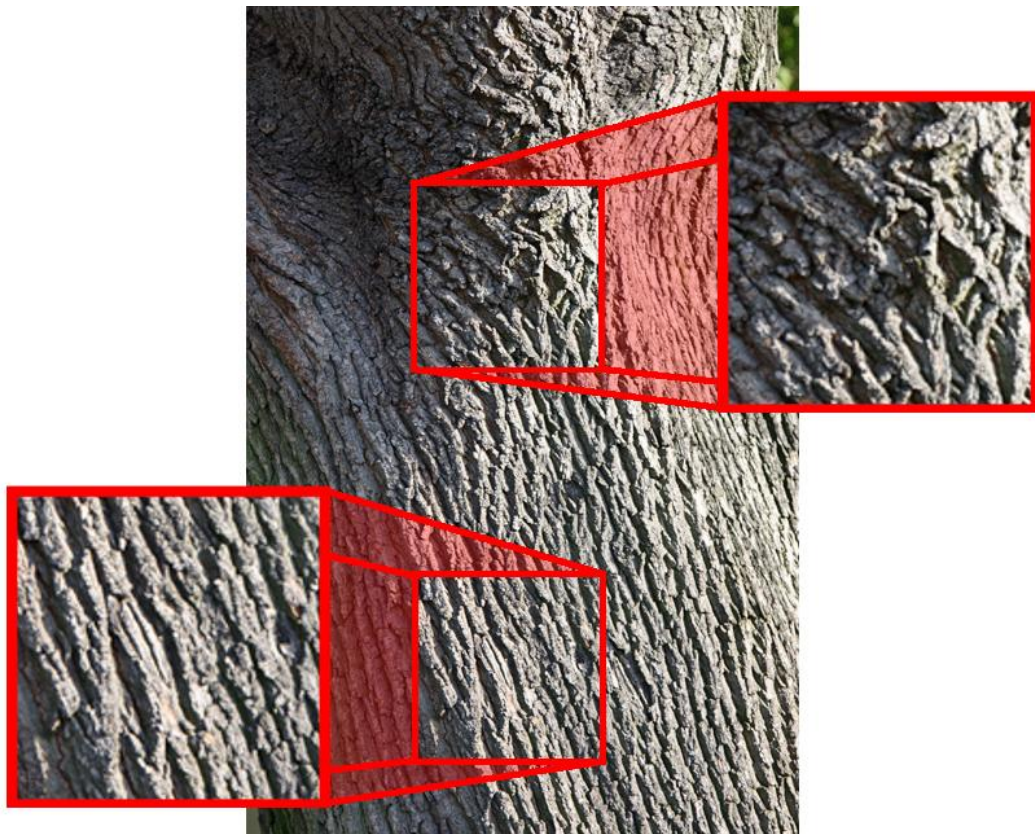
When observing nature it is apparent that each tree, even in a forest filled with the same tree type, is unique. It is easily noticeable that the branching configuration is different from tree to tree, where the structure includes the number, shape and angle of each of the branches. Similarly the bark on the surface of each tree is unique. Each tree's bark develops uniquely according to its growth and environment.

As a tree grows the formation of its bark is affected by many variables, both internal and external [25]. The internal variables include: the age and season of the tree's growth, natural mutations and the tree's genetics. External variables include; the type and quantity of minerals and nutrients



available to the tree, damage caused by weathering, availability of sunlight, interaction with insects and animals as well as the interference of other plant life.

These factors not only make the bark of each tree in a species unique but also make areas on the same tree different. As a tree grows the old layers of bark die and are pushed outward from the centre of the branch or trunk by new layers of bark [25]. The older dead bark dries out and fractures or peels, creating the familiar patterns on the surface of trees. Each area on the tree is unique for these reasons, as each area is constructed and fractures differently. However, at a quick glance different areas on a tree can look very similar or even identical. This is because the global structure, colour and texture are the same, while the local features are different. In Figure 2, two sections of a Sycamore tree are highlighted to show the differences between them. It is noticeable that the colour and texture of the bark is almost identical. The overall direction of the striations are very apparent in the bottom section while the direction in the top section is less defined and seems more random.



**Figure 6. This image highlights the differences between bark in two different sections of the same Sycamore tree [26]. The bottom section looks almost regular with striations in the same direction, from the top left hand corner to the bottom right hand corner. The top image looks more irregular with the overall direction of the striations harder to recognise.**

In starting this analysis of bark formation above, improvements can be made to the existing systems texture mapping. Each tree could have its own generated texture instead of the single texture per model. The single tiled texture can also be replaced by a few textures that tile together or a single

larger texture that covers the tree to reduce tiling. Further improvement can also be made to make the direction of the features of the texture follow the structure of the tree (similar to the striations in Figure2.)

### 3.1.4. Seamlessness

In the previous system, seams are created in two different situations. The first, is due to the tiling approach to texture mapping that is used. A single texture is tiled across the surface of the tree. If this texture was not created to be tileable then noticeable artefacts are created as shown in Figure7 and 8.



Figure 7. Two images are shown to illustrate tiling using a non-tileable image. Left is the sample image and right is the resulting tiled image with seams.



Figure 8. Two images are shown to illustrate tiling using a tileable image. Left is the sample image and right is the resulting tiled image without seams.

The second way that seams are created is at the joins of branches. The textures tiled over the join area do not match between the two branches joined. This is illustrated in Figure 5.

To reduce the introduction of seams, two approaches can be used. The first is to use a synthesis method that directly synthesises a seamless texture such as the method used by Wei and Levoy [6]. The second is to synthesise the texture directly onto the meshing, such as the method used by Turk [9], which removes the effect of seams created over joins.

### 3.2. Performance

The previous system is used to create tree models that can be used in 3D applications. This allows the user to build the tree model according to their specifications before using the model. The emphasis is on the creation of an accurate tree model in a reasonable amount of time. This means that the application should execute as fast as possible without the loss of realism.

The options available for texture synthesis are limited by the choices made in section 3.1 where an example based texture synthesis method was considered preferable. As seen in the previous chapter, texture synthesis methods can vary from real-time to lengthy execution times. The accuracy of these methods also varies greatly depending on the type of method and results required. To increase the accuracy of the existing system, a slow accurate method is preferable to an inaccurate, fast method. This rules out real-time texture synthesis methods such as the method by Lefebvre et. al. [20] due to their inaccuracies. Similarly, pixel-based methods are not desirable due to the loss of global features of the sample image during synthesis. For these reasons it is preferable to use either a patch-based or pixel-patch based hybrid techniques for bark synthesis.

## Chapter 4. Foundations

For this research we have chosen the technique presented by Han et. al. [27], commonly known as the discrete solver [5]. This is a hybrid between pixel and patch based texture synthesis techniques. The discrete solver is an expansion of the texture optimization technique [21]. This synthesizes a new texture at a pixel level unlike other pixel-based approaches because every pixel is considered together by optimizing a quadratic energy equation. The overall energy is determined by mismatches between input and output images. Iteratively solving this function over an image minimises the energy over subsequent iterations for a better quality image.

The discrete solver expands on this technique by synthesising the texture directly over a target mesh and by introducing a more parallelisable technique for solving the energy equation. Both the texture optimization and discrete solver techniques can be applied to flow animation texture synthesis. However, in this work we will focus on synthesising a single texture.

The reasons for choosing the discrete solver for synthesising bark are as follows:

- The technique can handle many different types of textures while producing high quality results. This is important for synthesising bark textures due to their inherent diversity and variation. As

shown in Figure 1. the resulting textures can have artefacts for highly structured images. However, due to the stochastic nature of bark these artefacts will be hard to find.

- The discreet solvers parallelisable nature makes synthesis very fast for both CPU and GPU implementations.
- By synthesising the bark texture directly over the tree model using the method described by Turk [9], visual artefacts created around seams are removed.

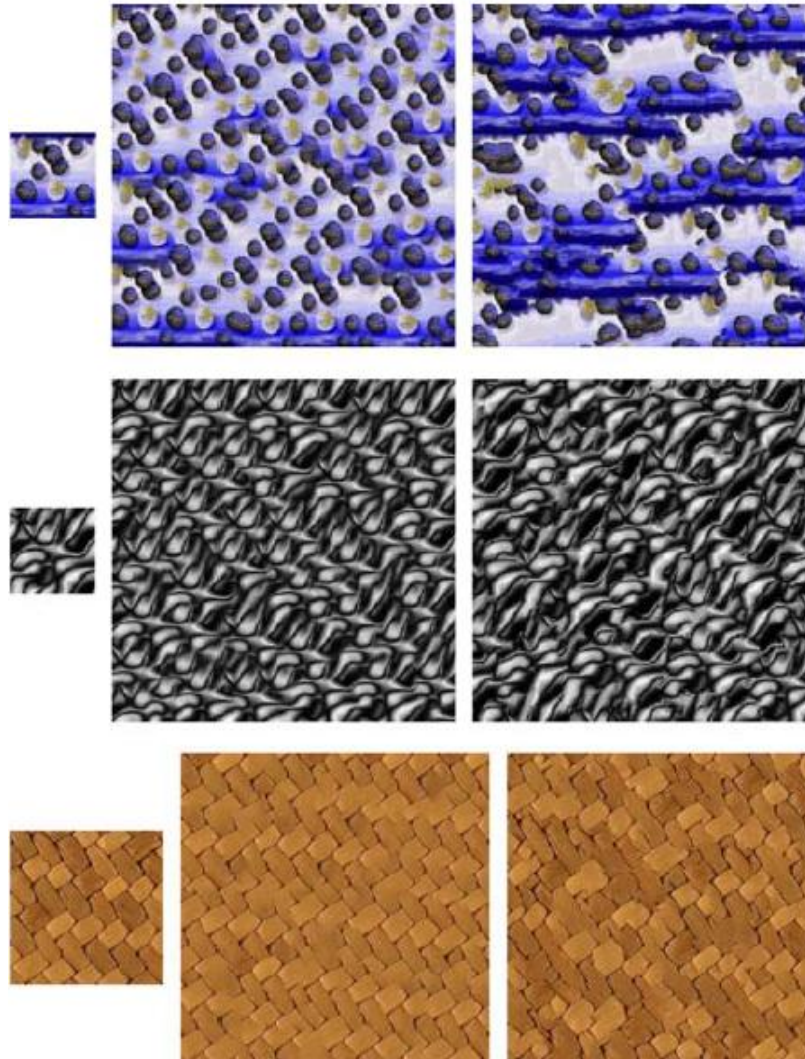


Figure 9. This figure shows a comparison of the discreet solver and texture optimization techniques for 2D images. On the left is the example texture, in the middle is the result from the discreet solver and the result from the texture optimization is on the right.

#### 4.1. Texture synthesis

As previously stated the texture optimization technique is a hybrid pixel and patched-based technique. This method has lengthily execution times, however, it is considered to be a very good general-purpose synthesis technique.

### 4.1.1. Texture Optimization

The texture optimization technique is algorithmically similar to Expectation Maximization (EM) algorithm [28]. EM is used for optimization when the desired variables, as well as the parameters of the energy function, are unknown [21]. The EM method iteratively alternates between solving the desired variables and the unknown parameters. In this case the desired variable is the texture being synthesised and the parameters are the set of input neighbourhoods. The texture optimization algorithm can be divided into two steps, the E-step and the M-step as shown in Algorithm 1.

The E-step consists of the estimation of the energy of an image  $X$ , where  $X$  is the texture to be synthesised. The E-step then tries to pull the values of the synthesised texture closer to the sample texture. The M-step minimizes the energy by finding the neighbourhood in the sample texture that is closest to each neighbourhood in the synthesised texture. The information gathered from the M-step is then used for the estimation of the E-step. So from these two steps an iterative algorithm can be created that minimizes the energy for the target texture providing better results after each iteration.

$$E_t(x; \{z_p\}) = \sum_{p \in X^*} |x_p - z_p|^2 \quad (1)$$

This represents the total estimated energy  $E_t$  for  $x$  over the subset of  $X$ ,  $X^*$ . This is a least squares solver that sums the error between the pixel neighbourhoods  $z_p$  and  $x_p$ . [21]

In the E-step the energy of the image  $x$  is  $X$ 's vectorised form. Similarly  $Z$  is the example texture and  $z$  is its vectorised form. The vectorised form  $x$  is found by concatenating the intensity values of each pixel in  $X$ . The estimation of  $x$  is achieved by minimising the energy equation seen in (1). This equation is a least squares solver that sums the error found between pixel neighbourhoods  $z_p$  and  $x_p$  for each pixel  $p$  in  $X^*$ , where  $z_p$  is the most similar neighbourhood from the input texture to  $x_p$ . The set  $X^*$  corresponds to a subset of  $X$  where each pixel  $p$  is spaced apart by a distance of  $w/4$  pixels, where  $w$  is the width of the neighbourhood. The spacing is intended to avoid too close an

overlap of pixel neighbourhoods producing costly redundant calculations.

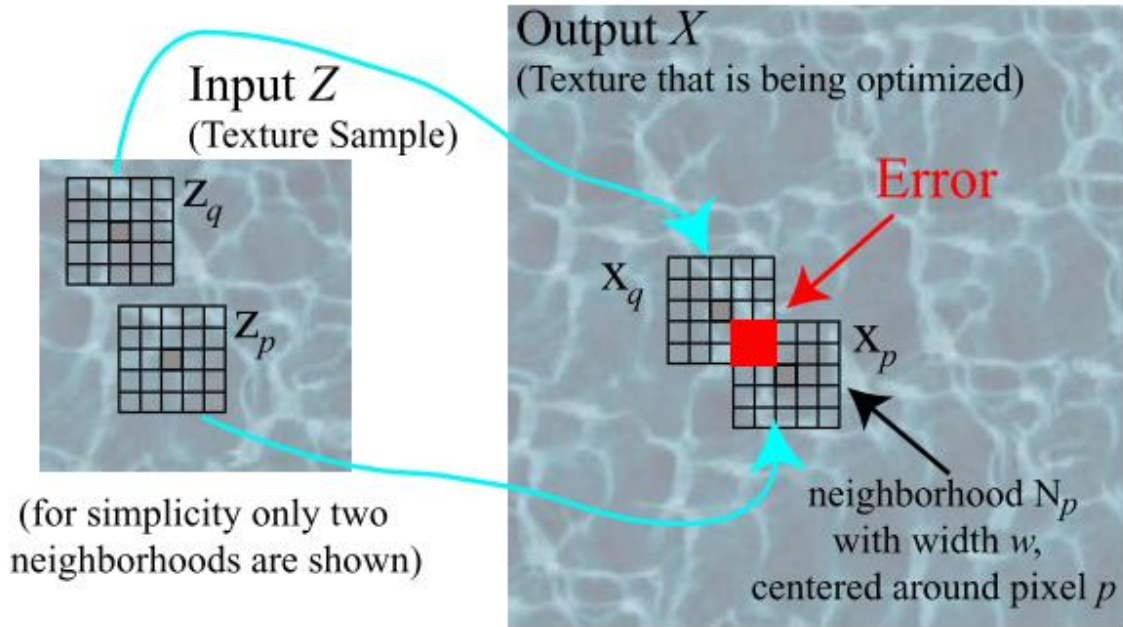


Figure 10. The energy of neighbourhood  $x_p$  centered around pixel  $p$  is given by its distance to the closest input neighbourhood  $z_p$ . When two neighbourhoods  $x_p$  and  $x_q$  overlap, then any mismatch between  $z_p$  and  $z_q$  will lead to accumulation of error in the overlapping region, shown in red [21].

The energy of Equation 1. is minimised by bringing each neighbourhood  $x_p$  as close to  $z_p$  as possible. If two neighbourhoods  $x_p$  and  $x_q$ , surrounding pixels  $p$  and  $q$ , overlap then the intensity values from the two neighbourhoods are averaged to provide a single intensity value for the overlapping pixels.

The M-step minimizes Equation 1 with respect to  $\{z_p\}$  by keeping  $x$  fixed at the new values found in the E-step. For each neighbourhood  $x_p$  the nearest neighbourhood  $z_p$  needs to be found. The similarity is determined by the distance squared between  $z_p$  and  $x_p$ . This is accelerated through the use of a tree structure and k-means clustering, where  $k = 4$  [21]. K-means clustering is a method of analysing clusters of data, by partitioning a number of observations into,  $k$ , clusters.

### Texture Optimisation

---

$$z_p^0 \leftarrow \text{random neighbourhood in } Z \forall p \in X^*$$

for iteration  $n = 0 : N$

$$x^{n+1} \leftarrow \operatorname{argmin}_x E_t(x; \{z_n^p\}) // E - \text{step}$$

$$z_p^{n+1} \leftarrow \text{nearest neighbour of } x_p^n + 1 \text{ in } Z \forall p \in X^* // M - \text{step}$$

if  $z_p^{n+1} = z_p^n \forall p \in X^*$  then

$$x \leftarrow x^{n+1}$$

```
        break
    end if
end for
```

Algorithm 1. The final iterative algorithm for texture synthesis using the E and M steps. [21]

---

This can be achieved through a set number of iterations or when the set  $\{z_p\}$  no longer changes, as seen in Algorithm 1. The algorithm can also be extended to synthesise at multiple resolutions and neighbourhood sizes. Better results can be made by first synthesising at low resolutions followed by a higher resolution synthesis from a scaled-up version of the resulting texture. In addition first synthesising with large neighbourhood sizes captures large features in the input texture, followed by refinement from smaller neighbourhood sizes capturing more local features.

#### 4.1.2. The Discreet Solver

The discreet solver iteratively synthesises the texture in similar manner to the texture optimization technique. The EM algorithm used by texture optimization is used as a base to build the discreet solver. The discreet solver extends the texture optimization for texture synthesis through the use of k-coherence. Coherence is introduced [18] to use the knowledge that clusters of pixels in the input tend to remain together in the output image. K-coherence, introduced by Tong et. al.[29], aims to provide a technique that improves the speed of synthesis while maintaining a high quality texture. K-coherence achieves this by keeping record of similar pixels to each pixel. Each similarity set is recorded for each individual pixel and is of finite size, k. K-coherence allows synthesis in linear time, according to the size of the image, in comparison to a tree search, which is of  $O(\log N)$  complexity. K-coherence is divided into two phases, analysis and synthesis.

##### *K-coherence Analysis Phase*

In the analysis phase a similarity set is built for each pixel in the sample image. The similarity set is composed of other pixels in the sample image with similar pixel neighbourhoods to the chosen pixel. This similarity set size is controllable and usually user defined a number in the range of [2,11]. The chosen size determines the quality and speed of the image produced. A small size will be much faster however noticeable breaks and artefacts in the synthesised images are created. A higher number improves the quality of the synthesis at the cost of the speed of synthesis. A size of 11 is generally considered to be the highest quality image with greater sizes producing results that are considered virtually identical. In the synthesis phase a pixel is copied from the input texture into the output texture. The pixels location in the sample texture is also stored to easily access its similarity set. For each pixel being synthesised a candidate set is created. The candidate set is created by adding the similarity set based on each pixel in the neighbourhood surrounding the pixel to be synthesised. A pixel from the input texture offset from the neighbourhood pixel's position contributes its similarity set to the candidate set. This offset is the opposite to the offset of the neighbourhood pixels position from the pixel being

synthesised. The candidate set is then used to find the best pixel for synthesis according to the closest neighbourhood in the candidate set to the neighbourhood of the pixel being synthesised.

### *K-coherence Synthesis Phase*

The analysis phase is introduced as a pre-process to the discrete solver to analyse the input texture for coherence. K-coherence is also incorporated into both the E and M steps of the texture optimization algorithm. Specifically tree structure with k-means clustering used in the M-step is replaced with a k-coherence search. In addition a new approach for the E-step is created to replace the least square solver. This forms a new algorithm that is considered to be a discrete optimization. as shown in Algorithm 2. To compute  $x^{n+1}$  in the new E-step each value  $x(p)$  is determined independently by finding a value in the candidate set  $\{c(p)\}$ . A candidate is chosen from the k-coherence candidate set that minimises the energy function. The candidate pixel and its input texture position are copied directly to  $x^{n+1}$ . The independence between synthesising each pixel makes this ideal for parallelisation. The new algorithm also removes blurring that would occur due to the blending between overlapping pixel neighbourhoods.



```
z_p^0 ← random neighbourhood in Z ∀ p ∈ X*
for iteration n = 0 : N
  x^{n+1} ← argmin_{x,x} E_t(x; {z_n^p}) //E-step
  z_p^{n+1} ← argmin_x |x_p - z|^2 //M-step
  if z_p^{n+1} = z_p^n Z ∀ p ∈ X* then
    x ← x^{n+1}
  break
end if
end for
```

Algorithm 2. The final discreet solver algorithm for texture synthesis using k-coherence.

---

## 4.2. Surface Texture Synthesis

There are two main methods for placing a texture over the surface of a mesh. The first is through texture wrapping where a flat texture is wrapped around the mesh. This technique however suffers from texture seams and distortion. Texture seams are created where the edges of the texture meet. This can cause visual artefacts when the edges do not match across the seam. Usually this is combated by making a texture seamless. Distortion is also created due to stretching a rectilinear texture over the meshes surface to fully cover the whole surface. The second method is to directly create the texture onto the meshes surface. Most commonly artists 3D model packages to draw directly onto the surface. This information is then directly mapped to a texture atlas through UV or texture coordinates. However this requires careful unwrapping of the models shape onto a flat plain. Similarly a texture can also be synthesised directly over a target mesh to remove the seams and distortion of texture wrapping.

### 4.2.1. The Discreet Solver

The discreet solver uses the method described by [9] to synthesise the texture directly over the meshes surface. This technique uses a point colouring method to synthesise the texture according to the layout of points across the surface. This is achieved through the use of a vector field covering the surface of the mesh, where there is a vector tangent to the surface at each point. Finally the vector field is used for indexing across the meshes surface so that the methods for texture synthesis can be easily used. The vector field allows the synthesis method to navigate up, down, left and right between pixels akin navigating a flat texture.

It is near impossible to create a regular grid of points over an arbitrary mesh surface. Instead points are evenly spaced over the surface of the mesh in an irregular arrangement. A point hierarchy is created through a point relaxation algorithm[30]. This is done by placing  $n$  points over the surface at random and space these over the surface evenly by using repulsion. These first  $n$  points are then fixed and will become the lowest level of the mesh hierarchy. Next  $3n$  new points are placed over

the meshes surface. These points are repelled from each other and the first  $n$  points are set in place to create the next level of the hierarchy of a total of  $4n$  points. This process continues to create the next mesh of  $16n$  points and so forth until the required amount of levels are completed. Once all points have been placed the mesh connectivity needs to be created. This is created by projecting the points onto a tangent plane and performing Delaunay triangulation.

Each point in the hierarchy contains information for the colour of the texture as well as the vector field orientation. The vector field is created by a few basis vectors placed over the surface, usually user specified or created from inherent knowledge of the model. The vector field is then interpolated from these basis vectors over the entire surface. This interpolation is achieved by performing a down sampling operation over the mesh to bring the basis vector values through to the highest level (lowest resolution) of the point hierarchy. The non-zero vectors in the mesh are then interpolated according to its surrounding vectors until every vector has a value. The vector values are then up sampled into the lower levels of the hierarchy, normalising the vectors at each level.

Once the vector field is created it is then used to order the pixels on the surface. This allows a process of surface sweeping to occur over the target mesh. This is created by assigning a sweep distance to each vertex according to an arbitrarily chosen anchor vertex. This is calculated again for the vector field rotated 90 degrees from its initial orientation. The distance is calculated between each vertex connected to another vertex in the mesh. The distance is found by measuring the distance between each vertex according to the average orientation the vertices field vectors. This is once again calculated at the highest resolution and up sampled throughout the hierarchy. From these distances the vertices are sorted according to their distances. These sorted lists create the regular indexing for texture synthesis.

## **Chapter 5. Implementation**

This chapter aims to provide a thorough explanation of the new features built to improve the variation and realism of bark. Improvements to the existing system include the addition of the Discreet Solver by Wei et. al. [27]. This technique was chosen as it fitted all the requirements as specified in Chapter 4. The implementation of the discreet solver is limited to the 2D texture synthesis only. This is due to the difficulty in the implementation of the surface texture synthesis and the constraints in the scope of the project. This chapter first gives an overview of the system, including the interaction with the previous system as well as the general workflow of the texture synthesis method. This is followed by a thorough explanation of the texture synthesis implementation and the discreet solver algorithm used.

### **5.1. System Overview**

The existing system was programmed in the C++ programming language and built using the Qt framework [31] for the graphical user interface (GUI). The system was built specifically for the Linux operating system but is extendable to both Macintosh and Windows operating systems. To fit into the existing system, the new features are also programmed in C++ and use Qt for additional GUI elements. The

existing system had additional dependencies including the boost C++ library. OpenMP [32] is added to the system as a new dependency for our additions. OpenMP is used for parallel programming as an alternative to QThreads provided by the Qt framework. OpenMP was chosen over QThreads because of: the ease of parallel programming provided by the #pragma directives, prior experience with OpenMP, as well as the portability of OpenMP.

### **5.1.1. Integration into the existing GUI**

The new texture synthesis method is attached to the existing system through an added synthesis GUI button. Once the button is clicked the synthesis algorithm that computes a new surface texture for the tree model can begin. A new pop-up dialog appears, as shown in Figure X, for the synthesis input. The synthesis input consists of the example texture, synthesis speed/quality and output texture size parameters. The example texture is chosen through the standard QFileDialog provided by the Qt framework. Synthesis speed/quality is set by using the slider provided by the synthesis dialog. The output size parameters are set through two spin-boxes, provided for input values. The spin boxes correspond to the pixel width and height of the texture resulting from the synthesis process.

### **5.1.2. Workflow for the texture synthesis**

The entire workflow for the texture synthesis is summarised in Figure 12 to give a high-level overview of how the system works.. The workflow begins with a user provided sample image as well as synthesis information. This information is then passed to the texture synthesis class, which begins by creating a matrix of coherent textures. These textures contain coherence information, that is used by the discreet solver. The matrix is made from coherence textures of each neighbourhood size for each resolution of the sample image. Once the coherence textures have been created from the sample image the TextureSynthesis class creates a randomly patched texture at a low resolution to initialize the discreet solver to begin. The random texture is created from random patches chosen from the sample image. An indexed texture is created from each pixel in the random texture. Each pixel from random texture stores its position in sample image, which is saved into the indexed texture.

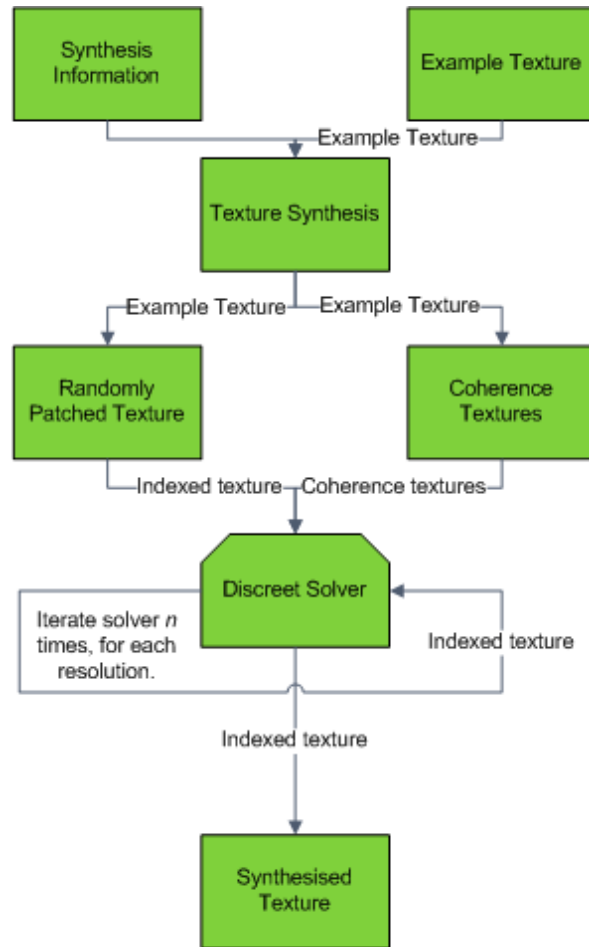


Figure 11. Workflow for the texture synthesis.

Once the discreet solver has both the sample and the coherence textures it may begin synthesising a similar texture to the sample image. The information provided by the coherence textures is used to synthesise a texture, with the random texture as a basis, that is closer to the appearance of the sample image. This is achieved iteratively with each iteration bringing the image closer in appearance to the sample. After  $n$  iterations at the initial synthesis resolution the resolution is increased. The resolution is increased and synthesised until  $n$  iterations have been applied at the required resolution, obtained in the synthesis information. The resulting texture is then saved once completed.

## 5.2. Texture synthesis implementation

The texture synthesis implementation is split into different sections organised by a central class called TextureSynthesis. This class calls different methods, which contribute to the overall synthesis process. The structure of the synthesis program is illustrated by the class diagram in Figure X. The main methods that are used in the Texture Synthesis class to load an image and synthesising. These methods, respectively control the analysis and synthesis phases of the discreet solver.

The image loading method takes the path of the texture to be loaded as a parameter. The image is loaded from this path and then stored as a Texture and collection of CoherenceTextures. These coherence textures are then used during the synthesis stage. After the *LoadImage* method has been called, the synthesis method follows immediately. The synthesis method first creates a random texture which is used as the basis for the coherence. Once the random texture is created then the synthesis process begins by iteratively solving the textures energy through the discreet solver. This generates a texture closer in energy and appearance to the sample image.

The remainder of this chapter will explain each step in further detail. First, the different texture types are discussed, including why they are important and how and where they are applied. The creation of the Gaussian hierarchy is then explained. This is followed by a short explanation of how the random texture is created. Finally, the implementation of the discreet solver is explained in detail.

### 5.2.1. Textures

The texture synthesis implementation requires the storage and manipulation of many images. All these images have the same basic storage structure and manipulation requirements. However, each texture has more advanced requirements that are not used by all other textures. For this reason a base class called Texture is created, that is extended by child classes. The parent class Texture provides storage of the pixel data for each image through a two-dimensional array of QRgb pixels. The QRgb class is a pixel class provided by the Qt framework that contains the intensity values for each pixel. These intensities are stored as integer values for each colour; red, green, blue and alpha. These values are in the range  $[0,255]$ , where the value 0 is the lowest intensity and the value 255 is the maximum intensity. The array of pixels in the texture can be accessed directly for manipulation at a per-pixel level.

The parent class *Texture* is extended by two different child classes. These classes are the *IndexedTexture* and *CoherenceTexture* classes. These child classes are completely separate and extend the texture class in different ways.

#### *Coherence textures*

Coherence textures are created during the analysis phase of the discreet solver. This phase finds the  $k$ -coherence matrix for each pixel in the coherence texture. The  $k$ -coherence matrix for a pixel is the set of pixels with the most similar neighbourhood of pixels to the pixels own. The size of the set,  $k$ , is determined from the synthesis information passed to the TextureSynthesis class. The set is stored in the CoherenceTexture class in the form of two-dimensional array of PixelCoherence objects.

#### PixelCoherence

The PixelCoherence class contains a list of  $k$  pixels for each neighbourhood size. These lists correspond to the coherence sets of the pixel. The PixelCoherence class constructs the sets from its corresponding CoherenceTexture class. The set is determined by finding the distance

between neighbourhoods for every pixel against every other, for a set neighbourhood size. This is an exhaustive search in which each pixel checks against all others. This is of  $O(n^2)$  complexity, where  $n$  is the number of pixels in the texture. The distance between the neighbourhoods is found by the Euclidean distance between each corresponding pair of pixels in the neighbourhood, as shown in Algorithm 3.

#### Pixel Coherence

---

```

queue // Priority Queue of pixels and the distance, sorted by the distance
integer sum
for each p in X
  for each q in X
    sum = 0
    for each t, s in Xp, Xq
      sum +=  $\sqrt{(r_t - r_s)^2 + (g_t - g_s)^2 + (b_t - b_s)^2}$  //Euclidean distance
    end for
    queue ← q, sum
  end for
end for

```

Algorithm 3. The algorithm to find the coherence set for each pixel,  $p$ . Where  $s$  is the pixel being checked against and  $X$  is a set of pixels in the texture. The pixels  $t$  and  $s$  are the pixels from pixel neighbourhoods  $X^p$  and  $X^q$ , respectively. The values  $r_t$ ,  $g_t$  and  $b_t$  are the intensity values of the red, green and blue colours for pixel  $t$  respectively.

---

The neighbourhood sizes are determined from the coherence texture size. The starting neighbourhood size is a 9x9 matrix of pixels. If the next neighbourhood size, double the previous size, is less than a quarter of the area of the texture then a new coherence set is created for that size. This terminates when the neighbourhood size exceeds a quarter of the area of the texture.

#### Creating the collection of coherence textures

The exhaustive search as shown above can be executed very quickly for small images of less than 64x64 pixels. However, due to the complexity of the search the execution time of the algorithm grows exponentially. This can be reduced by introducing a Gaussian hierarchy of images. If the sample images width or height is larger than 64 pixels then a copy of the image is created at half the size of the image. If the copy's width or height remain larger than 64 pixels then the process of downsampling is repeated. This is done until we are left with a texture that's width and height are smaller than 64 pixels. All of the copies that are made are kept to form the Gaussian hierarchy.

When an image is halved the pixel values for the new image are created from the original image using a Gaussian blur matrix. This matrix is a 2x2 window that is moved across the surface of the image, with the window never covering the same spot twice, as shown in Figure 12. The colour of the pixel in the copy is created from the weighting of the Gaussian blur matrix. For a simple 2x2 window the weightings are all equal. Therefore the resulting colour is just the average colour of the four pixels.

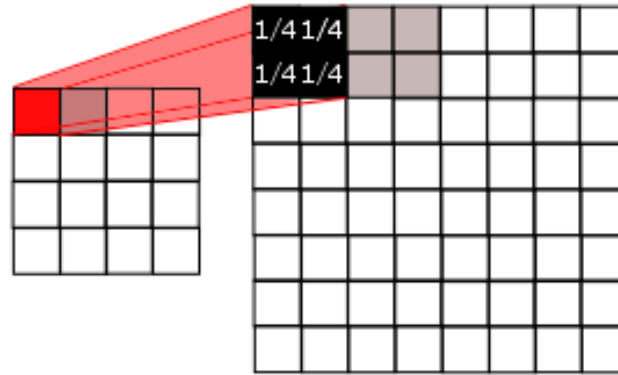


Figure 12 This diagram shows the how a copy of an image, half the size of the original, is created from a Gaussian blur of size 2x2. The weighting for each pixel is shown in white on a black pixel and the resulting pixel is highlighted in red. The next pixel to be blurred and the corresponding pixel window are shown in gray.

Once a hierarchy of images is created, a hierarchy of coherence textures can be constructed. The first coherence texture finds its coherence sets for the lowest resolution image through an exhaustive search as usual. The next coherence texture in the hierarchy finds its coherence sets from the sets found in the previous coherence texture. Each pixel creates its coherence set from the pixels in its own image that correspond to the other coherence set. This is illustrated in Figure 13. below.

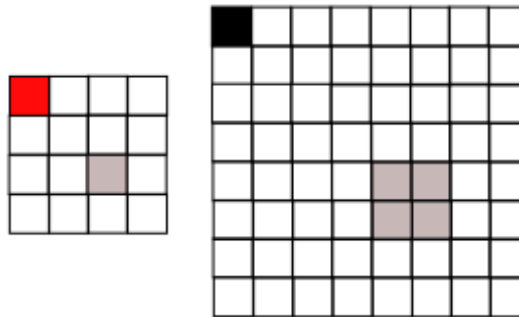


Figure 13. This image illustrates construction of a coherence texture from a previous coherence texture. The image to be synthesised is on the right, while its corresponding image in the lower resolution is on the left. Assume that the size of the coherence set is 1. The single gray block in the low resolution image corresponds to the coherent pixel to the red pixel. The four gray blocks on the right are the pixels the black pixel must now use to find its coherence set.

### *The Indexed textures*

Indexed textures are created from the textures constructed in the analysis phase of the discreet solver. The indexed textures work by copying individual pixels directly from the source. The

IndexedTexture class extends the Texture class by keeping a record of each copied pixels corresponding position in the source texture. When indexed textures are edited or created from other indexed textures, the changed pixel as well as the position from the source texture is copied. To maintain a consistent indexed texture the pixels, as well as their corresponding positions, must remain the same as the source texture at all times.

### 5.2.2. The synthesis method

The *synthesis* method synthesises a randomised image by using the discreet solver. The *synthesis* method and solver are only run once the *loadImage* method has been called at least once. This is because the solver needs an example image loaded and analysed into a hierarchy of coherence textures. Before a random texture is created, the size of the output texture is divided to create a hierarchy of synthesis sizes. This is done in the same way that the hierarchy is made for the coherence texture, continually dividing the texture in half if either width or the height is greater than 64.

#### *Creating a random texture*

The first thing the synthesis method does after the hierarchy, is create a randomised indexed texture. This is created by dividing the lowest resolution coherence texture into sub-patches. The patches are a subset of the texture with a size of an eighth of the full texture. Patches are used instead of random pixels from the texture so that a sense of structure is still passed from the original. However, this is not a requirement as any randomization is suitable for synthesis. This is provided the indexed information is retained from the coherence textures pixel positions. From the set of sub-patches a full indexed texture is made in the size of the lowest resolution of the synthesis hierarchy. This is created by tiling the patches randomly to create the full sized texture.

#### *Calling the discreet solver*

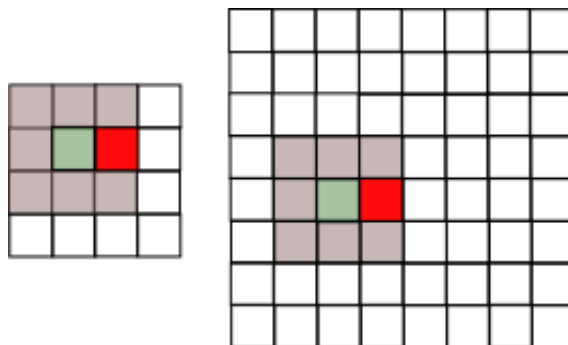
Once the random or output texture is created then the synthesis via the discreet solver can begin. The iterative synthesis process then begins on the random texture. The discreet solver is called to iterate on the texture for each neighbourhood size. The sizes are determined by the coherent texture being used by the output texture. Once all of the iterations have been run for each neighbourhood size then the output texture is up-scaled. The up-scaling process is the inverse operation to the halving seen in the section on coherent textures. The four corresponding pixels, from the higher resolution coherence texture, are copied into a texture double the size. This creates a new output texture completely from the higher resolution coherence texture. This continues and is further up-scaled and iterated over output size is equal to the desired size.

### 5.2.3. The discreet solver

The *discreetSolver* is a method in the *TextureSynthesis* class and is called from the synthesis method. The discreet solver optimizes a function according to the energy of an equation measuring the error of the synthesised image. By optimising the texture energy iteratively the image is synthesised to be increasingly similar to the sample texture after each iteration.



The iterates over the image calculating the closest neighbourhood  $x_p$  in the output texture to the neighbourhood  $z_q$  in the sample texture. The pixels in the closest neighbourhood calculation lie within the set  $X^*$ , a subset of the input image  $X$ .  $X^*$  is a subset where the pixels  $p$  start at coordinate (0,0) in the texture. The next neighbourhoods are spaced  $w/4$  pixels away, in the  $x$  and  $y$  directions, to provide enough overlap between pixel neighbourhoods.



**Figure 14** This figure illustrates the candidate set choice from overlapping neighbourhoods. The red pixel in the right image is the pixel being synthesised. Its candidate set is created from the gray neighbourhood that overlaps the pixel (the green pixel marks the centre of the neighbourhood). From the candidate set each candidate pixel (the green, centre pixel) is used to find the corresponding pixel (in red on the left) to the pixel being synthesised. The distance between these pixels is used to choose if the corresponding pixel is to replace the pixel being synthesised.

Once all of the closest neighbourhoods in  $X^*$  have been found then the algorithm begins to find the best matching pixel, for each pixel in the output image. A candidate set is created from the k-coherence set of each of the neighbourhoods in  $X^*$ , that overlap across the point being synthesised, as illustrated in Figure 14. The best matching pixel is a pixel chosen from this candidate set that minimizes the energy equation. Which is the closest total distance to each neighbourhood (the sum of all distances from a point to its corresponding point in each neighbourhood).

## Chapter 6. Evaluation

The evaluation of this research is separated into user and performance testing. The user testing itself is composed of two different rounds. The first round consisted of expert realism testing and took place after the first prototype of the system. Feedback from this round of testing was qualitative in nature. The second round consisted of standard realism testing which took place just before the final system was complete. The null hypothesis for the results of the realism testing was that there is significant difference between the realism of the sample and synthesised textures. Feedback from this round of testing quantitative in nature with parts which were qualitative. This chapter explains the expert and standard user testing as well as performance testing. This includes the design, procedure and results for each type of testing.

## 6.1. Expert User Testing

People considered to be expert users were invited to participate in the first round of user testing. Expert users were chosen to be part of the study if they have had prior experience with computer graphics and the algorithms involved. The selection consisted of computer science lecturers and post-graduate students. The expert testing was intended to improve the understanding and design of the requirements before further implementation. This helped remove the cost of changing an already developed system.

### 6.1.1. Design

The format for the expert testing was designed to occur early in the development lifecycle and was kept informal. This informality was intended to reduce the overhead of designing and running a full experiment during crucial development time. Due to this informality and the small selection of expert participants, ethical clearance was also not needed. The reason that the first round testing occurred so early in development, was so that design flaws and conceptual inconsistencies could be ironed out early-on. For this reason The interviews consisted of the following:

- An explanation of the prototype to be presented, as well as the plan for the full system.
- A demonstration of the prototype, and
- Feedback from the participant.

This demonstration of the prototype included examples of textures produced from synthesis. The synthesised textures are created from a small set of bark textures with a large variation of feature types and sizes. The variation of features, tests the robustness of the synthesis technique used, while the difference in sizes tests the stability and performance of the system.

### 6.1.2. Procedure

The participants were recruited by emailing an invitation. Each participant was interviewed separately to focus on the feedback from each expert and remove external influences on the interview. Once a participant arrived they were briefed about the overall project. The participants were encouraged to ask questions and give advice throughout the interview.

First, an explanation was given about the existing system (Tree Draw). This was followed by a discussion of the new system and what it should ideally achieve. After the discussion another explanation was given about the prototype being demonstrated. The demonstration of the prototype was given immediately after explanation. Finally, the participants were asked for any closing advice or input on the remaining implementation and design of the system.

### 6.1.3. Results

The results of the expert testing highlighted strengths and weaknesses in the prototype.

#### **Strengths**

- The discreet solver was seen to have the potential to fulfil all the requirements.
- Local structure was maintained and features from the sample image were visible.
- The performance was seen to be reasonable in comparison to experience with other synthesis techniques.

## Weaknesses

- Many of the participants were concerned that all of the intended features might not be feasible for the scope of the research.
- The prototype produced noticeable repeating patterns and a loss of global structure.
- While the performance was reasonable it was not yet as fast as the discrete solver could be.

From the strengths and weaknesses it was decided that implementation should continue on the discrete solver. The prototype did not feature a resolution and neighbourhood size hierarchy, which was recommended to maintain global structure. Further optimizations should be investigated to enhance the speed of the algorithm. It was decided that additional features should only be implemented if they fitted into the scope of the research. These additional features include: synthesising the texture directly over the mesh and normal or displacement maps.

## 6.2. Final User Testing

The final user testing was conducted to evaluate the results of the completed system. The results from the discrete solver were compared against the sample textures used in synthesis. The two textures are compared according to how similar they look to real bark textures. Users were recruited to participate in the experiment through poster advertisements. The user population group was limited to students currently studying at the University of Cape Town. To be able to recruit students, access to students and ethical clearance was applied for and granted by the Department of Student Affairs and the Faculty of Science Research Ethics Committee respectively. These were both obtained before user test recruiting began. Both documents are available in Appendix A and B. The posters were placed across the university campus to attract students from different backgrounds and fields of study. An offer of R30 per person for their evaluation was also used to try to attract participants. The students then contacted the researchers through the contact information provided on the poster and signed up for a testing slot. In each testing slot the users were given a set of tasks to complete. The tasks were given out digitally through a program designed for the user testing.

### 6.2.1. Design

Feedback from this round of testing was quantitative in nature with parts which were qualitative. The null hypothesis for the results of the realism testing, was that there is significant difference between the realism of the sample and synthesised textures. A t-test is used to try and disprove this result. The type of test used is a two-sample test assuming equal variances.

The evaluation platform for the final user testing was an interactive program. Each user's session consisted of three different sections to evaluate the results of the texture synthesis and capture demographic information. The first section required the user to rate a set of images (sample images obtained from the paper by Liang et. al. [12]). The second asked the user to provide comments on these images. Finally, the third section asked the participant for information about themselves.

### *Section 1*

This welcomes the user and explains the format of the program. Once the user selects the button labelled "Begin" the first section begins. This consists of a rating system for displayed images. The section is separated into a set number of questions. A single image is displayed per question, chosen from a selection of images. The selection of images consist of images of bark, as well as images produced by the synthesis. The program randomises the order of the images in the questions as a pre-process before the user begins the section. Randomisation provides control over the sequence of events, ensuring that effects produced by a specific order do not occur with every subject. Each question has an adjustable realism rating from 0-100 situated below the image, as shown in Appendix D. The user is able to change the rating of the images with a score of 0 considered "not at all realistic", and a score of 100 considered "highly realistic". The participant may change the rating of the displayed image by using the slider or by entering a number into the spin-box provided. Once the rating is set the participant may continue to the next image by pressing the "Next" button.

### *Section 2*

Once all of the questions in the first section have been completed the second section will begin automatically. This displays an image, similar to the first section, however, the rating system is replaced with a comment box. Two images, with the lowest ratings, are selected from the first section. Only the two lowest ratings were chosen to limit the size of the comment section, while retaining the most important information. This also reduces the time that each person needs to spend on the comments section, as commenting can take considerable time. The participant is asked to provide a comment for the images, explaining why they have received low scores. Once a comment has been given the participant may continue by pressing the "Next" button.

### *Section 3*

Once the two images have received comments in the second section, the evaluation automatically continues to the final section. This consists of a single page with multiple questions. The participant is asked information about themselves that may provide insight into their answers. The information is recorded through written comments, values placed into spinboxes, as well as toggle-button areas.

The following questions are asked in section 3:

- The age of the participant.
- The main field(s) of study pursued by the participant.
- How much experience the participant has with the botanical sciences.
- How much experience the participant has with computer graphics.
- How much experience the participant has with 3D modelling packages.
- How much experience the participant has with computer games.

### *Saving and loading sessions*

When a user begins a session the program automatically creates a save file, named uniquely. This makes the task of acquiring the information for analysis much easier. To keep the anonymity of the participant each file is named according to its session and computer numbers. The computer number is determined from a configuration file which is set by the session moderators. The session number is determined from the save files already contained in the "Save Files" directory. Each new session determines its number as the next highest session number, with the first session starting with the number 1. This results in session numbers starting at the number 1, incrementing until the number of sessions created. In addition to the standard session numbers a backup save file is created to prevent losing data in the event the computer or program stops during the save process

The session is saved each time the "Next" button is pressed in-between pages. The save files contain all of the important information from a session, so that a session can be reloaded in the case of a fault in the program. The information is separated by placing each page on a separate line. Individual pieces of information on a page are separated by commas.

The format for a save file is as follows:

Key:

// - Comment line, which is not a part of the format.

[Number, Number2] - A number in the range between, and including, Number and Number2.

. - An unknown number of additional lines in the same format as the line above.

Format:

Section Name

//Start of the rating section

The image name for the question, the rating for the image [0,100]

The image name for the question, the rating for the image [0,100]

.

.

.

//Start of the comment section

The image name for the question, the rating for the image [0,100], the comment for the image

The image name for the question, the rating for the image [0,100], the comment for the image

.

.

.

//Start of the personal information section

The participants area of study, age, [0-999], gender, [0,1], experience in Botany, [0,4], experience in Computer Graphics, [0,4], experience in 3D modeling, [0,4], average hours spent playing games per week, [0,999], the number of years playing games for, [0,999],

An example of the save file format:

Bark

FileName,Image1.jpg,Score,50,

FileName,Image2.jpg,Score,50,

Lowest, Image1.jpg,Score,50,Comment,This is a comment,

Lowest,Image2.jpg,Score,50,Comment, This is a comment,

Branches

FileName,Image1.jpg,Score,50,

FileName,Image2.jpg,Score,50,

Lowest, Image1.jpg,Score,50,Comment,This is a comment,

Lowest,Image2.jpg,Score,50,Comment, This is a comment,

Leaves

FileName,Image1.jpg,Score,50,

FileName,Image2.jpg,Score,50,

Lowest, Image1.jpg,Score,50,Comment,This is a comment,

Lowest,Image2.jpg,Score,50,Comment, This is a comment,

StudyDetails,ComputerScience,Age,23,Gender,0,Botany,2,ComputerGraphics,4,Modeling,1,GameHoursPerWeek,8,GameYears,17,

### 6.2.2. Procedure

The venue used for the evaluation was a small computer laboratory containing a selection of computers. The computers available were three dual-boot Windows/Ubuntu PC's of similar specifications. The evaluation program was built specifically for the Ubuntu operating system, so each computer needed to be booted into Ubuntu and the program installed beforehand. In the case of a double booked appointment a laptop running the program in Ubuntu was used. Before the experiment begins, users are given a briefing about the evaluation. The users are given a short description of the previous system and what it achieves. This is followed by an explanation of what the new features are trying to add to the system. After the background explanation an overview of the format of the evaluation takes place. For this research it was explained that we would like to improve the bark textures of the tree models generated by the old system.

Once the explanations were completed the participants were asked to fill out a consent form. After this they began their evaluation using the program. The participants continued with the evaluation and were allowed to ask the moderator questions at any point. Once the user was finished they would need to let a moderator know. The users would be paid R30 for their participation and sign a proof payment for the moderator. This was finally finished off by inquiring if the participant had any additional questions, and then thanked for their participation.

### 6.2.3. Results

#### *Pilot test*

Before the main testing began a pilot test consisting of three users was run to test the procedure of the evaluation and the stability of the program. Three users participated in this pilot study. From this two different faults were found with the implementation of the program. Firstly the program did not parse the comments section properly, resulting in only the first word of the comment being saved. The second fault occurred in the loading and saving of the session. When a session was loaded, it would not load all of the information correctly. Then when the program would try to automatically save the session the program would crash and corrupt both the save file and the backup.

#### *Main test*

##### Demographic data and sample significance

As the realism is assumed to stay the same between the real and synthesised images, a two-tailed t-test was used to test for significance. Using a significance level of 0.05, it was calculated that a minimum population size of 21 participants was required. The final population size for the study, was 35 users. The age range for the population was [18-24] years old, with the average age of the participants at 21. The gender of the population was split roughly 1:4, with 9 males and 26 females.

##### Test for normality

The distributions, for both real and synthesised images, are created from the mean values of each image type. Before a t-test could be run over the real and synthesised population groups, a check for normality was used. The normality test was calculated through the Shapiro-Wilk method, using the R statistical package. Shapiro-Wilk tests the null hypothesis that a sample came from a normally distributed population. The  $W$  values for were found to be 0.9498 and 0.9154, with a  $p$  values of 0.7275 and 0.4345 respectively. The  $p$  values were compared to an alpha value threshold of 0.05, and it can be concluded that the null hypothesis for both populations cannot be rejected. Therefore, each population group come from a normal distribution.

##### T-test for the population groups

A t-test is now calculated for the real and synthesised distributions, as shown in the Box and Whisker plot in Figure 15. The mean values for each population are 45.8968254 and 40.81349 respectively. The t-test used is a two-tailed, assuming equal variance test. The t-stat value calculated is 0.688949552, while the  $p$  value for the two-tailed test was found to be 0.503958297. This is much larger than the alpha constraint of 0.05. Unfortunately for this reason the null hypothesis cannot be rejected. This means that the two distributions cannot be assumed to have equal realism. From this information and value of the mean it can be concluded that the synthesised images are less realistic than the sample images.



Figure 15 A box and Whisker plot for the distributions real and synthesised image respectively. The grey crosses represent the median values for the distributions, while the blue lines represent the first and third quartiles (Q1 and Q3).

### 6.3. Performance Testing

Due to the parallel nature of the discreet solver specific tests need to be run to test how parallelised the implementation is. By testing the performance over a number different threads we are able to determine how useful the parallelisation of the technique is. The criteria used to measure the parallelisation are: the speed-up gained through parallelisation and the efficiency of the parallelisation.

#### 6.3.1. Design

To measure the performance of the new parallel code we will be using the speedup associated to the best sequential method. Speedup is defined as  $S_n = \frac{T_s}{T_p}$  where  $S_n$  is the speed up with  $n$  processors,  $T_s$  is the best sequential execution time (not using any parallel methods) and  $T_p$  is the parallel execution time. Another metric associated with parallel algorithm analysis is the efficiency of the parallel method. Efficiency is defined as  $E_n = \frac{S_n}{n}$  where  $E_n$  is the efficiency of  $n$  processors,  $S_n$  is the speed up with  $n$  processors and  $n$  is the number of processors.

#### 6.3.2. Procedure

For each test the same sample image was used as a benchmark for the calculations. The calculations were run on a single laptop running the Ubuntu operating system. The specifications for the laptop are as follows: a Quad Core Intel i7 2.00Ghz with hyper-threading and 8GB of DDR3 Ram. The hyper-threading of this particular laptop creates a processor having 8 logical cores for testing.



The testing was run by first measuring the time taken to execute the sequential method. This was followed with the parallel OpenMP solution with a variable number of threads used. The number of threads,  $n$ , is in the range of [2,8] with regular increments. The QTime object provided by qt was used to track the time taken for each

### 6.3.3. Results

The results for the speedup and the efficiency are graphed below. The same sample image of size 80x80 pixels was analysed and then another images was synthesised at 30x30 pixels. The results for each of these methods are placed on the same graph for comparison.

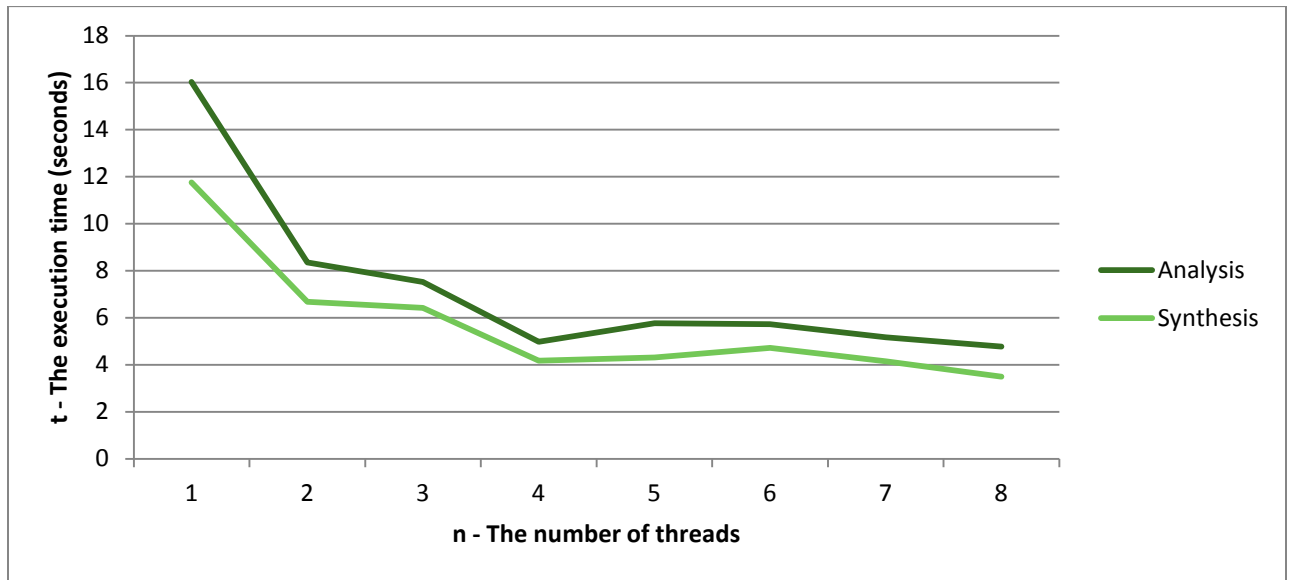


Figure 16 The time of execution for analysis and synthesis, graphed against the number of threads used.

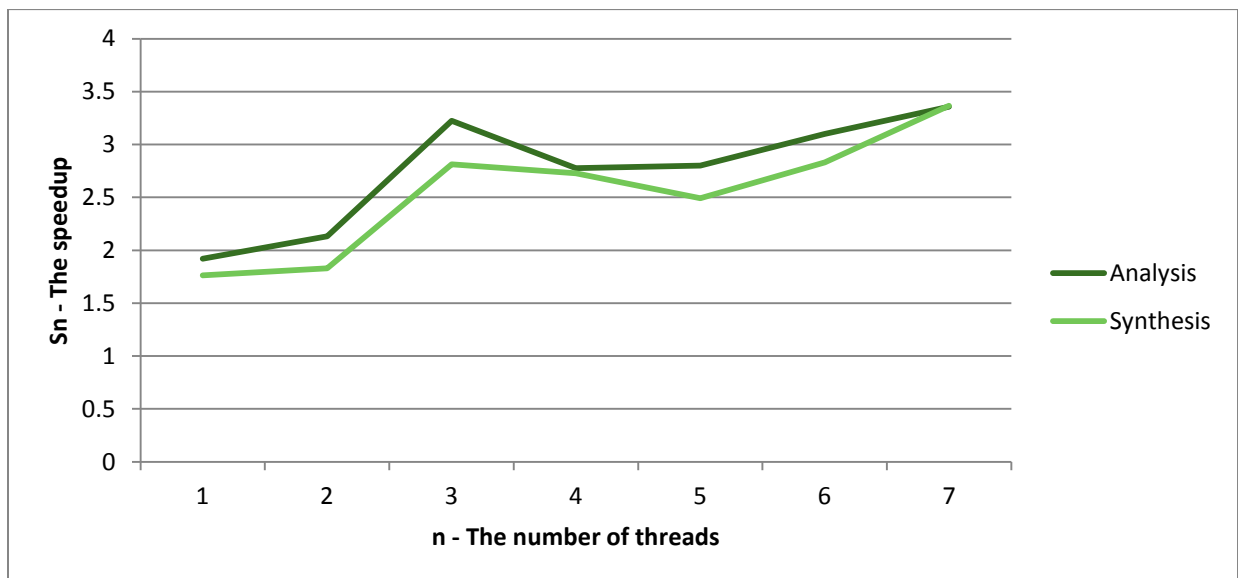


Figure 17 The speedup for analysis and synthesis, graphed against the number of threads used.

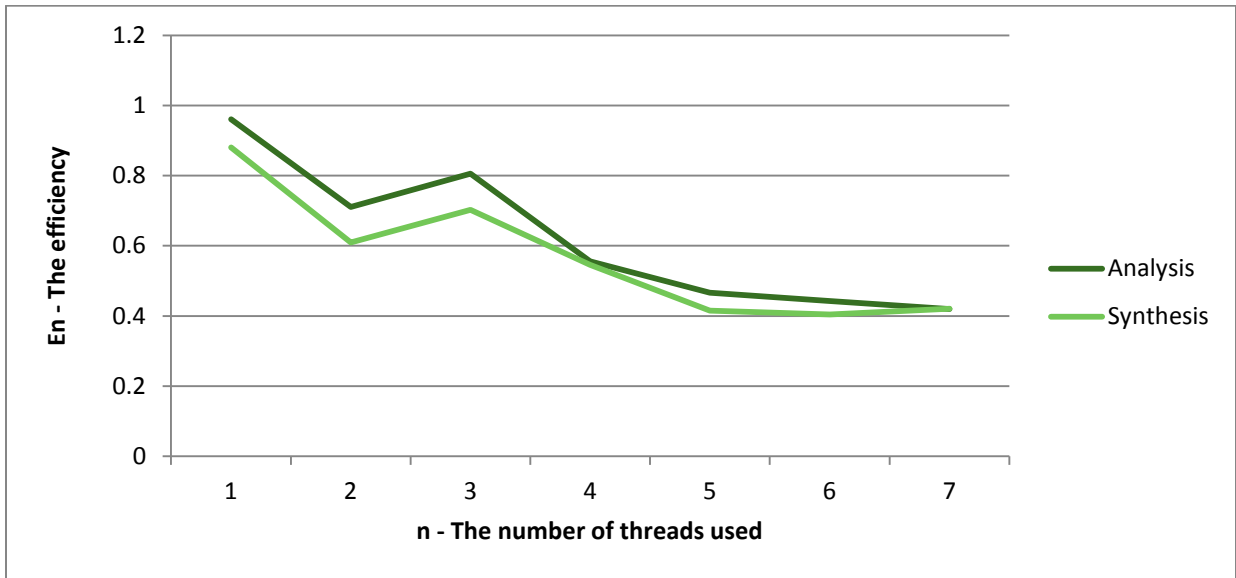


Figure 18 The efficiency for analysis and synthesis, graphed against the number of threads used.

#### 6.4. Testing conclusion

From the expert testing conducted it was determined that the discreet solver would fit all of the requirements highlighted in chapter 3. It was also decided that each of the intended additional features would most likely not fit into the scope of the thesis, and should only be implemented once the discreet solver was fully implemented.





**Figure 19** Two examples of failed synthesis results produced by the discreet solver in the realism testing. The examples shown are the sample images [left column] [12] and the synthesised images [right column]. The synthesised images were discovered to be inaccurate due to an error in implementation, where the sampled neighbourhoods were situated tightly in the middle of the sample image. This resulted in the obvious repeating patterns visible throughout the images.

The final realism testing measured the assumption that the realism of the synthesised images were the same as the real images. It was concluded through the use of t-tests that the distributions were different, with the sample images having a higher realism than the synthesised images. It was discovered later that an error in the solvers implementation caused repeating patterns in the synthesised images, as seen in Figure 16. Since this implementation it is assumed that the realism is increased from further development. However, this cannot be proven conclusively without further testing. Additional examples of synthesised textures from the discreet solver are shown in Appendix C.

The results produced during the performance testing conform to the expected results from parallelisation. The execution time drops drastically with the introduction of threads, however, it quickly plateaus after four threads. This is mirrored in the speedup and efficiency graphs, as the speedup is gained quickly and begins to plateau, similarly for the efficiency. This is expected due to the increasing overhead created from thread organisation and synchronisation, when the thread count begins to get very large.

## **Chapter 7. Conclusion**

The demand for varied and realistic 3D assets has created many solutions for the procedural generation of models. TreeDraw is one such solution that addresses the procedural generation of tree models. It addresses the problem of procedurally generating trees that accurately fit the specifications provided by users. Users create a basic sketch of a tree, which is then interpreted by L-systems to create 3D tree models with similar structure to the base sketch.

Project Yggdrasil aims to extend the work found in TreeDraw by creating more realistic tree models. This is achieved through three different areas: subdivision surfaces, procedural leaf generation and texture synthesis. This thesis aims to focus on the investigation of texture synthesis for the generation and variation of bark textures. Through background research and expert testing it was decided that the discreet solver, an example-based synthesis technique, would fit the criteria of all of the texture improvements. These improvements include creating seamless bark textures that are similar in structure and realism to a sample image. The discreet solver was implemented to provide seamless 2D textures that resemble the sample image as accurately as possible.

From user testing, flaws in the accuracy of the implementation were highlighted. These flaws were fixed and it is assumed that the accuracy and realism of the solver fit the requirements adequately. However, further testing is needed to prove these results. Other improvements can also be added, but these are out of scope for this thesis and are set for future work.

## **7.1. Future Work**

Future work for the texture mapping for project Yggdrasil include:

- The addition of normal/displacement mapping to increase the realism of the textures in 3D. These techniques use texture mapping to displace the geometry and vertex normals of a 3D model to produce bumps and surface features.
- Synthesising the bark texture directly over the target mesh, using the technique applied in the discrete solver. This technique can remove seams created across the joins of branches during texture mapping.


## Chapter 8. References

- [1] "SpeedTree." [Online]. Available: <http://www.speedtree.com/>. [Accessed: 22-Oct-2012].
- [2] M. Black, "A sketch-based interface for realistic tree generation with variation," University of Cape Town, 2011.
- [3] A. Lindenmayer and P. Prusinkiewicz, *The Algorithmic Beauty of Plants*. 1990, pp. 1–46.
- [4] M. Runions, Adam Fuhrer, B. Lane, P. Federl, A.-G. Rolland-Lagan, and P. Prusinkiewicz, "Modeling and visualization of leaf venation patterns," in *Proceedings of ACM SIGGRAPH 2005*, 1999, pp. 702–711.
- [5] L. Wei, S. Lefebvre, V. Kwatra, and G. Turk, "State of the art in example-based texture synthesis," in *EG-STAR*, 2009, no. Section 2, pp. 93–117.
- [6] L. Wei and M. Levoy, "Fast Texture Synthesis using Tree-structured Vector Quantization," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques SIGGRAPH 00*, 2000, pp. 479–488.
- [7] K. Perlin, "An Image Synthesizer," *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 291–300, 1985.
- [8] S. Worley, "A Cellular Texture Basis Function," in *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 96)*, 1996, pp. 291–294.
- [9] G. Turk, "Texture Synthesis on Surfaces," in *Proceedings of SIGGRAPH '01*, 2001, pp. 347–354.
- [10] X. Zhang and Y. J. Kim, "Efficient texture synthesis using strict Wang Tiles," *Graphical Models*, vol. 70, no. 3, pp. 43–56, May 2008.
- [11] A. A. Efros and T. K. Leung, "Texture Synthesis by Non-parametric Sampling," in *Computer Vision. The Proceedings of the Seventh IEEE International Conference*, 1999, vol. 2, pp. 1033 – 1038.
- [12] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics*, vol. 20, no. 3, pp. 127–150, Jul. 2001.
- [13] V. Kwatra, I. Essa, and A. Bobick, "Graphcut Textures : Image and Video Synthesis Using Graph Cuts," *ACM transactions on graphics*, vol. 22, no. 3, pp. 277–286, 2003.
- [14] Y. Liu, "Near-regular texture analysis and manipulation," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 368–376, 2004.
- [15] Y. Zhang, Z. Sun, and W. Li, "Texture synthesis based on Direction Empirical Mode Decomposition," *Computers & Graphics*, vol. 32, no. 2, pp. 175–186, Apr. 2008.

- [16] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen, "Wang Tiles for Image and Texture Generation," in *Proceedings of SIGGRAPH '03*, 2003, pp. 287–294.
- [17] T. Ng, C. Wen, T. Tan, X. Zhang, and Y. J. Kim, "Generating an  $\omega$ -Tile Set for Texture Synthesis," in *Proceedings of the 23rd Computer Graphics International*, 2005, pp. 177–184.
- [18] M. Ashikhmin, "Synthesizing natural textures," in *Proceedings of the 2001 symposium on Interactive 3D graphics - SI3D '01*, 2001, pp. 217–226.
- [19] A. A. Efros and W. T. Freeman, "Image Quilting for Texture Synthesis and Transfer," in *Proceedings of SIGGRAPH '01*, 2001, pp. 341–7.
- [20] S. Lefebvre and H. Hoppe, "Parallel Controllable Texture Synthesis," in *Proceedings of SIGGRAPH '05*, 2005, pp. 777–786.
- [21] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra, "Texture optimization for example-based synthesis," in *Proceedings of SIGGRAPH '05*, 2005, p. 795.
- [22] "Silver Birch Bark." [Online]. Available: <http://www.geograph.org.uk/photo/1290234>. [Accessed: 05-Oct-2012].
- [23] "Oak tree bark." [Online]. Available: <http://www.geograph.org.uk/photo/640929>. [Accessed: 05-Oct-2012].
- [24] "Bark of a London Plane (*Platanus × acerifolia*)." [Online]. Available: <http://www.geograph.org.uk/photo/3019538>. [Accessed: 05-Oct-2012].
- [25] B. Y. T. C. Whitmore, "STUDIES IN SYSTEMATIC BARK MORPHOLOGY I. BARK MORPHOLOGY IN DIPTEROCARPACEAE," *New Phytologist*, vol. 61, no. 2, pp. 191–207, 1962.
- [26] "Sycamore tree bark." [Online]. Available: <http://www.geograph.org.uk/photo/1045706>. [Accessed: 05-Oct-2012].
- [27] J. Han, K. Zhou, L. Wei, M. Gong, H. Bao, X. Zhang, and B. Guo, "Fast example-based surface texture synthesis via discrete optimization," *THE VISUAL COMPUTER*, vol. 22, no. 9–11, pp. 918–925, 2006.
- [28] H. O. Hartley, "Maximum Likelihood Estimation from Incomplete Data," *Biometrics*, vol. 14, no. 2, pp. 174–194, 1958.
- [29] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum, "Synthesis of bidirectional texture functions on arbitrary surfaces," in *Proceedings of ACM SIGGRAPH*, 2002, pp. 665 – 672.
- [30] G. Turk, "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion," *Computer Graphics*, vol. 25, no. 4, pp. 289–298, 1991.
- [31] "QT Developer Network." [Online]. Available: <http://qt-project.org/>. [Accessed: 22-Oct-2012].

- [32] "The OpenMP API specification for parallel programming." [Online]. Available: <http://openmp.org/wp/>. [Accessed: 22-Oct-2012].

# Appendix A

	<b>RESEARCH ACCESS TO STUDENTS</b>	<b>DSA 100</b>
---	--	----------------

**NOTES**

1. This form must be FULLY completed by applicants that want to access UCT students for the purpose of research.
2. Return the completed application form together with your research proposal to: Moonira.Khan@uct.ac.za; or deliver to: Attention: Executive Director, Department of Student Affairs, North Lane, Steve Biko Students' Union, Room 7.22, Upper Campus, UCT.
3. The turnaround time for a reply is approximately 10 working days.
4. NB: It is the responsibility of the researcher/s to apply for and to obtain ethical clearance and access to staff and/or students, respectively to the (a) Faculty's 'Ethics in Research Committee' (EIRC) for ethics approval, and (b) Executive Director, HR for approval to access staff for research purposes and the (c) Executive Director, Student Affairs for approval to access students for research purposes.
5. For noting, a requirement of UCT (according to Senate policy) is that items (1) and (4) apply even if prior clearance has been obtained by the researcher/s from any other institution.

**SECTION A: RESEARCH APPLICANT/S DETAILS**

Position	Staff / Student No	Title and Name	Contact Details (Email / Cell / land line)
A.1 Student Number	FSTDON001	Mr. Donovan Foster	FSTDON001@myuct.ac.za Cell: 072 934 5664
A.2 Academic / PASS Staff No.			
A.3 Visiting Researcher ID No.			
A.4 University at which a student or employee	UCT	Address if <u>not</u> UCT:	
A.5 Faculty/ Department/School	Department of Computer Science		
A.6 APPLICANTS DETAILS If different from above	Title and Name	Tel.	Email

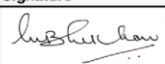
**SECTION B: RESEARCHER/S SUPERVISOR/S DETAILS**

Position	Title and Name	Tel.	Email
B.1 Supervisor	A/Prof. James Gain	021 650 4058	jgain@cs.uct.ac.za
B.2 Co-Supervisor/s (a)			

**SECTION C: APPLICANT'S RESEARCH STUDY FIELD AND APPROVAL STATUS**

C.1 Degree (if a student)	BSc (Honours)
C.2 Research Project Title	Procedural Tree Generation Improvements
C.3 Research Proposal	Attached: Yes <input checked="" type="checkbox"/> No <input type="checkbox"/>
C.4 Target population	UCT registered students
C.5 Lead Researcher details	If different from applicant:
C.6. Will use research assistant/s	Yes <input type="checkbox"/> No <input checked="" type="checkbox"/>
C.7 Research Methodology and Informed consent:	Research methodology: Scientific Method Informed consent will be obtained for participants. Participation is voluntary. Confidentiality, anonymity and privacy of participants, is assured by the researcher.
C.8 Ethics clearance status from UCT's Ethics in Research Committee (EIRC)	Approved by the EIRC: Yes <input type="checkbox"/> No <input type="checkbox"/> Awaiting response: <input checked="" type="checkbox"/> If yes, attach copy and state the date and ref. no of EIRC approval: Date of application if awaiting response: 07-09-2012

**SECTION D: APPLICANT/S APPROVAL STATUS FOR ACCESS TO STUDENTS FOR RESEARCH PURPOSE  
(To be completed by the ED, DSA or Nominee)**

D.1 APPROVAL STATUS	Approved / * With Terms	* Conditional approval Terms	Applicant/s Ref. No.:	
	*Yes <input checked="" type="checkbox"/>	(a) Access to students for this research study must only be undertaken after written ethics approval has been obtained. (b) In event any ethics conditions are attached, these must be complied with before access to students.	FSTDON001 / Donovan Foster	
D.2 APPROVED BY:	Designation	Name	Signature	Date
	Executive Director Department of Student Affairs	Moonira Khan		13/09/2012



## Appendix B

Department of Environmental and Geographical Science  
University of Cape Town  
RONDEBOSCH 7701  
South Africa

e-mail: [Michael.meadows@uct.ac.za](mailto:Michael.meadows@uct.ac.za)  
phone : + 27 21 650 2873  
fax : +27 21 650 3791



27<sup>th</sup> September 2012

Mr Donavon Foster  
Department of Computer Science  
[FSTDON001@myuct.ac.za](mailto:FSTDON001@myuct.ac.za)

Dear Mr Foster

### **Procedural Tree Generation Improvements**

I am pleased to inform you that, having scrutinized the details of your above-named applications for research ethics clearance, the Faculty of Science Research Ethics Committee has approved your proposal in terms of its attention to ethical principles. You must have formal agreement from the Department of Student Affairs in order to approach UCT students with the survey.

Your approval code for the project is: SFREC 42\_2012

I wish you success in the work involved.

Yours sincerely

*M Meadows*

Michael E Meadows  
Professor and Head of Department  
Chair: Science Faculty Ethics in Research Committee

## Appendix C

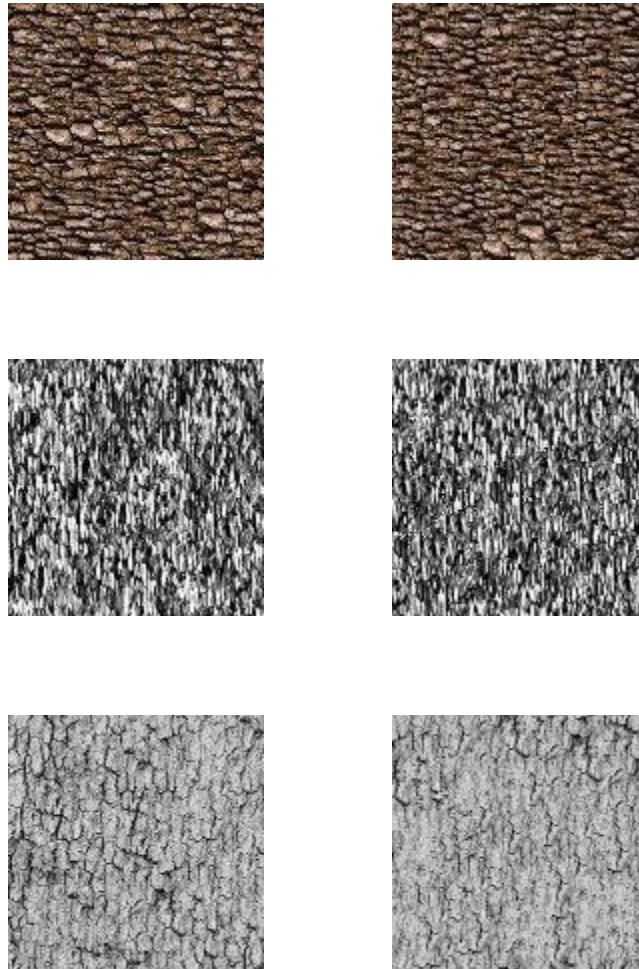



Figure 20 Examples of synthesis results produced by the discreet solver. The examples shown are the sample images [left column] [12] and the synthesised images [right column].

**Appendix D - The evaluation program**

Question 17/62



Not at all realistic Highly realistic

Value:

Section:  25% Next

Overall:  23%

### User Details

Gender:  Male  Female  N/A

Age:

0

Field(s) of study:

How experienced are you in the area of botany?

Not at all  Not Much  Moderate  Very Much  Highly

How experienced are you in the area of computer graphics?

Not at all  Not Much  Moderate  Very Much  Highly

How experienced are you with 3D modeling applications?

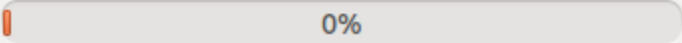
Not at all  Not Much  Moderate  Very Much  Highly

How many hours per week do you play video games?

0

How many years have you played video games for?

0

Section:  0%

Next

Overall:  98%

# Appendix E - The raw data from the evaluation

----- Bark images -----			
Real			
Rbark12_P30N8_128.jpg	20	Once again the yellow patches make it seem as if this is not a tree and the fact that its blurry makes it hard to ma	
Rbark14_P30N4_128.jpg	50		
Rbark14_P30N4_128.jpg	50		
Rbark3_P32N4_128.jpg	30		
Rbark3_P32N4_128.jpg	10	It looks like a bar code... Cannot really see that that is a bark of a tree.	
Rbark3_P32N4_128.jpg	10	Same as before. This one shows slightly more detail than the previous bark image but it is still a poor quality rend	
Rbark3_P32N4_128.jpg	8	it kind of looks like little perfect squares and rectangles dumped up together and thats not really how trees look	
Rbark3_P32N4_128.jpg	8	it kind of looks like little perfect squares and rectangles dumped up together and thats not really how trees look	
Rbark3_P32N4_128.jpg	2	you can differentiate the different parts of that section of the tree! shows that it was created using some sort of p	
Rbark3_P32N4_128.jpg	10	I think the colour doesn't give it much of a nature vibe and it mostly looks like a bush or growth on a tree. The con	
Rbark3_P32N4_128.jpg	5	The colour as well as the texture of the bark. It is undefined	
Rbark3_P32N4_128.jpg	28	extremely blurry and complicated	
Rbark3_P32N4_128.jpg	1	Colouring. Pixelation. Too random	
Rbark3_P32N4_128.jpg	0	It looks like a pained wall by some artist whose trying to tell an unclear story.	
Rbark3_P32N4_128.jpg	36	it's too blurry for me it's difficult to make out what it is.	
Rbark3_P32N4_128.jpg	40	can't tell that it's bark. It looks like a black and white blur	
Rbark3_P32N4_128.jpg	6	The colour and the amount of pixels make the image look unrealistic.	
Rbark3_P32N4_128.jpg	10	For me this bark seems too "grainy" and too "jovely"	
Rbark3_P32N4_128.jpg	20	the image is blurry and unrealistic one cannot see the actual bark components	
Rbark3_P32N4_128.jpg	33	Too many white spaces.No distinct portions can be seen clearly	
Rbark3_P32N4_128.jpg	0	this	
Rbark3_P32N4_128.jpg	30	its not clear what it is supposed to be so its not realistic as bark	
Rbark3_P32N4_128.jpg	20	The colour used is not suitable for trees.	
Rbark3_P32N4_128.jpg	36	again this does not show the right image of exterior of the tree	
Rbark3_P32N4_128.jpg	0	it just doesnt look like bark it looks lly my tv when there is no signal	
Rbark3_P32N4_128.jpg	2	it looks like when a tv screening plays funny	
d630_1977_o_128x128.jpg	27	it's just too abstract, so it automatically gives it a computer feel rather than a realistic one	
d630_1977_o_128x128.jpg	18	The texture of this bark looks a bit too smooth to be realistic. Even the smoother barked trees have a pattern here	
d630_1977_o_128x128.jpg	41	if you had not told me that it's bark i would have never guessed it looks like a background of colour it needs a bit	
d630_1977_o_128x128.jpg	2	It is too bleyr	
Synthesized			
Rbark0k_P30N8_128_1est.jp	37	the image is not really clear i do not understand if that is the bark of the tree or just a piece of the whole tree	
Rbark12_P30N8_128_1est.jpg	32	the	
Rbark12_P30N8_128_1est.jpg	30	I can't tell that maybe the yellow is lichen or moss but it does not have enough clarity and the colours do not look r	
Rbark12_P30N8_128_1est.jpg	10	The yellow patches don't make it seem like a tree!ne fact that it's blurry makes it hard to make out what it really is	
Rbark12_P30N8_128_1est.jpg	0	its really blurry and i have never seen such markings on a tree	
Rbark12_P30N8_128_1est.jpg	15	It's a bit too blurry. Looks more like a low resolution ground tile for a game.	
Rbark12_P30N8_128_1est.jpg	50	it is too blurry	
Rbark12_P30N8_128_1est.jpg	45	seems blurry if it was more clear it would look realistic	
Rbark12_P30N8_128_1est.jpg	30	it doesnt look realistic and has a techno look to it.	
Rbark12_P30N8_128_1est.jpg	50		
Rbark14_P30N4_128_1est.jpg	1	not clear whether it is the ground or a tree	
Rbark1_P32N4_128_1est.jpg	1	it is blur	
Rbark3_P32N4_128_1est.jpg	10	It's really blurry.	
Rbark3_P32N4_128_1est.jpg	8	Too pixelated It looks more like a blur. There are no definite bark segments since bark tends to dump.	

/Rbark3_P32N4_128_test.jpg	0	the image is too blurryit doesnt exactly show the details
/Rbark3_P32N4_128_test.jpg	22	It is too blurry and the colour does not make me think of bark
/Rbark3_P32N4_128_test.jpg	10	The image is too busy and it also doesn't show any characteristic of belonging to a bark. It is also not very clear.
/Rbark3_P32N4_128_test.jpg	5	The texture of the bark and the colour. The image is blurry to an extent
/Rbark3_P32N4_128_test.jpg	1	Again. Colouring and pixelation. Here even more than in the first picture there is no overall pattern to the shading
/Rbark3_P32N4_128_test.jpg	40	here the colour makes it look like a bar code that has been expanded on the screen. Rather than true bark texture
/Rbark3_P32N4_128_test.jpg	0	its black and white really? a black and white tree? there is no sych it also looks really pixelated
/Rbark3_P32N4_128_test.jpg	0	There is too much giong on in this picture very unclear to be a tree
/Rbark3_P32N4_128_test.jpg	40	there's isnt shadows or something that helps you make out what it isit can be anything really
/Rbark3_P32N4_128_test.jpg	8	It is not clear
/Rbark3_P32N4_128_test.jpg	15	The colour and the amount of pixels make the image look unrealistic.
/Rbark3_P32N4_128_test.jpg	20	This bark is too "grainy"
/Rbark3_P32N4_128_test.jpg	50	I PERSONALLY THINK THAT IT DOES NOT LOOK LIKE A TREE IT LOOKS LIKE A TV THAT HAS AN ARIEL
/Rbark3_P32N4_128_test.jpg	20	its not very clear as to what it is and you cant really see the pieces ofbark as clearly as some of the otehr images
/Rbark3_P32N4_128_test.jpg	45	the image should be more defined more clear
/Rbark3_P32N4_128_test.jpg	0	This iamge gives me no sense of what object is being displayed. It gives no distinct feature form which the object
/Rbark3_P32N4_128_test.jpg	30	It looks like a maze if not a computer chip.
/Rbark3_P32N4_128_test.jpg	35	cannot see clearly it is not possible to tell if this a tree or something else...
/Rbark3_P32N4_128_test.jpg	0	colour
/Rbark3_P32N4_128_test.jpg	0	the blurred shadowing
/d30_1977_o_128x128_test.jpg	38	the
/d30_1977_o_128x128_test.jpg	20	The blur in the image makes it look more like a rainy day picture the is no emphasis on any natural detail.
/d30_1977_o_128x128_test.jpg	10	its really smoth and a bit blurry it looks like a water painting its got too little texture
/d30_1977_o_128x128_test.jpg	10	its really smoth and a bit blurry it looks like a water painting its got too little texture
/d30_1977_o_128x128_test.jpg	20	its blurry and it looks like it has been ordered
/d30_1977_o_128x128_test.jpg	15	the marking are not vivid
/d30_1977_o_128x128_test.jpg	15	It looks like some kind of painting than a bark of a tree.
/d30_1977_o_128x128_test.jpg	16	The image looks as if it has rows of colour almost parallel to each other. This makes it unrealistic as it looks too 'r
/d30_1977_o_128x128_test.jpg	16	The image looks as if it has rows of colour almost parallel to each other. This makes it unrealistic as it looks too 'r
/d30_1977_o_128x128_test.jpg	10	the image is slightly grainy and i am unable to tell whether or not it is tree bark
/d30_1977_o_128x128_test.jpg	45	THIS IMAGE IS NOT CLEAR TO ME IT DOESNT LOOK LIKE A TREE
/d30_1977_o_128x128_test.jpg	38	It is certainly not clear at all its like looking through a wet window
/d30_1977_o_128x128_test.jpg	41	Bad contrast makes it look unrealistic
/d30_1977_o_128x128_test.jpg	30	again the texture of the bark is not very realistic. the colour of the bark is but the image is not as clear as some of
/d30_1977_o_128x128_test.jpg	0	that
/d30_1977_o_128x128_test.jpg	0	It is difficult to tell what the image is exactly without any prior knowledge that the study is based on tree images. I

----- Bark Images -----																																				
Real																																				
/RBarkdrk_P36N8_128.jpg	80	75	81	48	63	60	55	50	50	60	50	11	60	31	56	70	55	70	73	57	50	25	69	69	55	95	96	70	40	56	24	65	50	46	53	34
/Rbark12_P36N8_test.jpg	50	55	40	23	23	35	32	35	20	30	31	8	65	50	40	45	45	40	30	45	24	35	35	50	55	70	11	60	35	55	46	14	40	40	17	60
/Rbark14_P36N4_128.jpg	78	50	70	45	48	50	35	58	41	40	39	12	70	64	35	60	30	40	40	50	70	30	40	56	67	79	97	70	20	65	44	25	50	60	35	52
/Rbark1_P32N4_128.jpg	70	80	72	54	55	40	57	65	45	15	45	12	72	50	50	45	60	82	58	63	25	35	70	63	51	73	37	80	75	64	39	40	40	50	30	12
/Rbark3_P32N4_128.jpg	42	30	10	10	8	2	32	10	35	5	28	1	46	0	0	35	15	30	32	40	6	10	20	51	46	33	5	30	0	47	2	0	30	20	0	2
/Rbark8_P32N8_128.jpg	88	70	80	58	80	66	50	90	80	60	40	25	70	50	60	75	80	61	50	80	40	61	60	47	83	100	80	40	69	50	40	60	80	55	83	
/d30_1977_o_128x128.jpg	70	40	20	34	19	71	50	50	50	45	42	22	27	82	15	50	30	20	18	41	40	30	48	51	59	72	2	50	0	46	6	30	50	57	39	26
Synthesized																																				
/RBarkdrk_P36N8_128_test.jpg	70	60	63	40	47	47	53	45	50	50	40	9	48	23	45	60	90	60	64	55	50	20	43	57	37	93	76	60	40	46	2	50	50	57	48	36
/Rbark12_P36N8_128_test.jpg	32	40	40	22	20	30	30	30	10	30	30	5	60	0	40	40	20	15	25	50	20	33	30	52	53	60	5	40	30	45	38	10	30	40	15	55
/Rbark14_P36N4_128_test.jpg	50	55	18	46	59	50	56	47	46	70	35	6	58	75	40	60	80	38	68	50	50	40	55	65	54	92	70	40	50	59	1	55	50	55	45	31

/Rbark1_P32N4_128_test.jpg	60	70	80	42	53	60	53	70	40	10	40	10	60	5	45	50	60	72	52	53	20	35	56	50	50	66	20	70	70	60	1	10	40	41	18	20
/Rbark3_P32N4_128_test.jpg	40	39	10	8	46	0	22	10	33	5	38	1	40	0	0	40	8	30	41	44	15	20	40	50	38	63	33	20	3	45	5	0	40	30	0	0
/Rbark8_P32N8_128_test.jpg	70	70	90	46	56	50	48	95	70	65	40	13	50	24	60	50	55	58	75	49	45	55	74	65	50	87	92	70	40	70	18	60	60	55	50	29
/d30_1977_o_128x128_test.jpg	38	20	60	30	10	9	30	26	44	10	20	2	50	13	10	45	15	15	16	43	35	30	10	45	38	41	55	30	0	50	31	0	40	52	21	50

Study/Details	Commerce	Age	18	Gender	1	Botany	0	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	0	GameYears	0
Study/Details	Commerce Boom Economics and Statistics	Age	19	Gender	1	Botany	1	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	1	GameYears	10
Study/Details	BSc. HUB Ecology & Evolution and Genetics	Age	19	Gender	1	Botany	4	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	0	GameYears	0
Study/Details	International politics and Economic History (Bachelor of arts)	Age	19	Gender	1	Botany	1	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	6	GameYears	10
Study/Details	bsc geomatics	Age	19	Gender	0	Botany	0	ComputerGraphics	1	Modeling	2	GameHoursPerWeek	7	GameYears	9
Study/Details	Bachelor of Science	Age	19	Gender	1	Botany	1	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	0	GameYears	0
Study/Details	Bachelor of Business Science	Age	19	Gender	0	Botany	2	ComputerGraphics	1	Modeling	1	GameHoursPerWeek	3	GameYears	5
Study/Details	B.Sc Human Physiology	Age	23	Gender	1	Botany	1	ComputerGraphics	1	Modeling	1	GameHoursPerWeek	0	GameYears	0
Study/Details	BSc - Environmental Science and biology	Age	19	Gender	1	Botany	1	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	1	GameYears	5
Study/Details	Bsc Biochemistry and Chemistry	Age	21	Gender	0	Botany	3	ComputerGraphics	2	Modeling	2	GameHoursPerWeek	10	GameYears	7
Study/Details	Commerce - Economics	Age	24	Gender	1	Botany	0	ComputerGraphics	1	Modeling	1	GameHoursPerWeek	0	GameYears	0
Study/Details	Commerce	Age	21	Gender	1	Botany	0	ComputerGraphics	1	Modeling	0	GameHoursPerWeek	1	GameYears	11
Study/Details	law	Age	19	Gender	1	Botany	0	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	0	GameYears	0
Study/Details	Biochemistry	Age	23	Gender	1	Botany	2	ComputerGraphics	1	Modeling	0	GameHoursPerWeek	0	GameYears	3
Study/Details	MCB (biochemistry)	Age	23	Gender	0	Botany	2	ComputerGraphics	2	Modeling	1	GameHoursPerWeek	6	GameYears	13
Study/Details	Electrical Engineering	Age	22	Gender	0	Botany	2	ComputerGraphics	2	Modeling	1	GameHoursPerWeek	30	GameYears	12
Study/Details	Computer Science	Age	26	Gender	0	Botany	1	ComputerGraphics	1	Modeling	1	GameHoursPerWeek	1	GameYears	10
Study/Details	BScEng in Chemical Engineering	Age	18	Gender	1	Botany	1	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	4	GameYears	10
Study/Details	microbiology/applied bioogy(botany and zoology combined)	Age	22	Gender	1	Botany	4	ComputerGraphics	1	Modeling	0	GameHoursPerWeek	2	GameYears	8
Study/Details	Applied mathematics	Age	19	Gender	0	Botany	1	ComputerGraphics	1	Modeling	2	GameHoursPerWeek	3	GameYears	11
Study/Details	Maths and Psychology	Age	20	Gender	1	Botany	1	ComputerGraphics	2	Modeling	1	GameHoursPerWeek	5	GameYears	12
Study/Details	sociology	Age	21	Gender	1	Botany	1	ComputerGraphics	1	Modeling	0	GameHoursPerWeek	0	GameYears	0
Study/Details	HUMANITIES	Age	21	Gender	1	Botany	0	ComputerGraphics	0	Modeling	1	GameHoursPerWeek	0	GameYears	0
Study/Details	social development and sociology	Age	20	Gender	1	Botany	1	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	0	GameYears	10
Study/Details	Computer science/microbiology	Age	20	Gender	1	Botany	2	ComputerGraphics	2	Modeling	2	GameHoursPerWeek	1	GameYears	1
Study/Details	humanities	Age	20	Gender	1	Botany	0	ComputerGraphics	0	Modeling	0	GameHoursPerWeek	0	GameYears	0
Study/Details	ba law	Age	20	Gender	1	Botany	0	ComputerGraphics	1	Modeling	0	GameHoursPerWeek	0	GameYears	0
Study/Details	Business														
Study/Details	bsocsci law	Age	19	Gender	1	Botany	1	ComputerGraphics	1	Modeling	1	GameHoursPerWeek	9	GameYears	8
Study/Details	Geomatic engineering	Age	19	Gender	0	Botany	0	ComputerGraphics	2	Modeling	2	GameHoursPerWeek	1	GameYears	4
Study/Details	BSC genetics and biology	Age	22	Gender	1	Botany	2	ComputerGraphics	2	Modeling	2	GameHoursPerWeek	1	GameYears	5
Study/Details	BsocSci politics and english major	Age	20	Gender	1	Botany	1	ComputerGraphics	1	Modeling	1	GameHoursPerWeek	3	GameYears	10
Study/Details	B Com	Age	19	Gender	1	Botany	1	ComputerGraphics	1	Modeling	0	GameHoursPerWeek	1	GameYears	4
Study/Details	BSc Chemistry	Age	23	Gender	0	Botany	1	ComputerGraphics	0	Modeling	2	GameHoursPerWeek	0	GameYears	5
Study/Details	humanities psychology	Age	19	Gender	1	Botany	2	ComputerGraphics	2	Modeling	1	GameHoursPerWeek	0	GameYears	0
Study/Details	bcom accounting	Age	21	Gender	1	Botany	1	ComputerGraphics	2	Modeling	1	GameHoursPerWeek	0	GameYears	0