HONOURS PROJECT REPORT

Procedural Leaf Generation Using Biologically-Motivated Algorithms

Donovan Foster

Supervised by:

A/Prof James Gain

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

ABSTRACT

The aim of this project is to improve on the realism of 3D trees generated by an existing system. The current system lacks any leaf creation or foliage placement ability and therefore a method must be found to create, realistic, procedural trees, whiles still holding true to the previous system's goals of usability and sketch-based design, as well as the ability to create more than a single object for one set of input parameters. This thesis solves this problem by examining the different shape, texture and venation techniques available, and choosing those that fit the criteria. Specifically, this thesis discusses a system with a sketch-based leaf outline definition, using biologically motivated algorithms to grow veins out into the leaf, and well as colouration based on biological colour values and patterns. The leaves are then placed on a tree using a foliage system designed from observations of leaf behaviour in deciduous trees.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank all of those who have helped me on this project, particularly my fellow project members Richard Pieterse and Ryan Mazzolini, and of course, our supervisor A/Prof James Gain, for his unlimited patience and guidance.

CONTENTS

LIST OF FIGURES

Introduction

With the current trend in increasing computer generated content in entertainment and science, the need for content of this type far outstrips the supply. What is needed is a fast efficient way to make trees that are realistic, but also shaped and coloured as you need them to be. Movies and games often need more interaction with a tree that to just generate it, while scientists are often interested in a specific case with very specific parameters.

As hard as it is to build one tree, it is next to impossible to make multiple trees that are clearly related, and yet different and unique. This is where procedural generation and this project come into play. This project aims to have a system where you can design a tree, from branches to bark and leaves and then create that tree and others similar to it, without expert knowledge and without tweaking parameters.

Luckily, trees are well suited to procedural generation, as they can be modelled accurately using a sets of rules that govern their growth and behaviour.

1.1. General Procedural Generation

Procedural generation is a form of automated object creation. One can use simple rules and parameters to create something much bigger and more complex. Trees are well suited to procedural generation because they are too complex to be feasible by hand, and they are self—similar and fractal in structure. This means that the general structure of large piece of the tree (say the trunk) is echoed in its branches. This allows trees to be generated from just a few rules, as they can be reused as needed at different scales. The most popular technique for modelling tree structures is known as the parameterised L-System (Prusinkiewicz & Lindenmayer, The algorithmic beauty of plants, 1991). L-Systems were developed by botanist A. Lindenmayer in 1968, and are a type of rule-based formal grammar. L-Systems are used extensively in all fields of procedural generation.

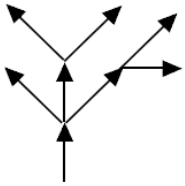
A basic structure for an L-System grammar consists of an alphabet, a seed string, known as an axiom, and a group of production rules. A production rule takes the form <target> -> <replacement>, where the target is a string of members of the alphabet and replacement is another such string. The process starts by making a string containing the axiom. At each time step, the production rules are applied in parallel to the string. If the rule finds its target, it replaces the target with its replacement.

$$\begin{array}{ll} \omega : & a_r \\ p_1 : & a_r \rightarrow a_l b_r \\ p_2 : & a_l \rightarrow b_l a_r \\ p_3 : & b_r \rightarrow a_r \\ p_4 : & b_l \rightarrow a_l \end{array} \qquad \begin{array}{l} a_r \\ a_l b_r \\ b_l a_r a_r \\ a_l a_l b_r a_l b_r \\ b_l a_r b_l a_r a_r b_l a_r a_r \\ \ldots \end{array}$$

Figure 1: Simple L-System with axiom (Ω) and production rules (left) and result (right)

These L-Systems can be used to model trees if one uses each member of the alphabet to represent a different action, for example moving forward or rotating. By this method, entire

trees can be built from just a few rules. Below is one such L-System for a 2D tree. 3D trees can also be modelled, however the L-Systems are more complex.



F[+F][-F[-F]F]F[+F][-F]

Figure 2: 2D Tree from L-System (Left) and the L-System result (Right)

However, trees are not perfectly self-similar. To this end, variations of an L-System are typically used. Stochastic L-Systems have many rules for the target each with a percentage chance. When the target string is found, one of the rules is randomly picked based on its percentage. This helps create unique trees. Another possible alteration to L-Systems is conditional L-Systems, in which each rule has can specify conditions about the string around its target that must be satisfied before it is used. This can help prevent putting fruit of a tree before the end branches for example.

PARAMETRIC

A term that arises frequently in procedural generation and this paper in particular is "parametric". To say that an object that is defined parametrically means that the data for the object has been stored as a small amount of intermediate variable that can be extrapolated out to regain the full object. For instance, a circle can be stored as a collection of points, or it can be stored and a radius and a centre point. Using these two parameters, it is possible to regain the original points of the circle. The advantages of storing object parametrically are twofold; Firstly, the parameters generally take up less space than the original data, although that is not always the case; and secondly, a slight change in the parameters can produce a similar, yet unique object. This is the principle behind procedural variation in both tree structures and leaf generation. This, of course, also depends on how similarity is defined. If a chosen parameter contains a number representing a certain shape, changing that number will produce a different shape, and thus a dissimilar object, however it would still be a shape, and share at least that similarity.

RANDOMNESS

The concept of randomness plays a big role in procedural generation. Whilst the parameters of a system constrain what outcomes are possible, random numbers allow the system to explore the full range of those constraints. Procedural generation focuses on the creation of new and unique objects, and thus it is not possible to store or hard-code that variation in. To do so would undermine the principles of procedural generation. That is not to say that everything in procedural generation is random. There are often occasions where the random data has to be transformed to make sense in the system. The Dart-Throwing algorithm, as used by (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005), generates uniform random points over an area. To do this, the random numbers generated are scaled over the area in question to ensure they are valid.

1.2. Biological Definitions

This paper does not concentrate heavily on the biological definitions of leaves when explaining the various algorithms inspired by nature, however, there are some specialised terms and concepts that should be kept in mind as one reads this paper.

1.2.1. Terms describing leaf shape

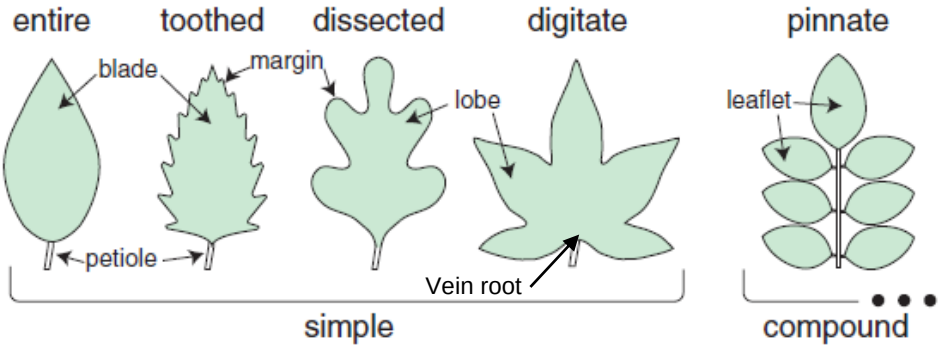


Figure 3: Terms relating to leaf shape (from Runions (2005))

1.2.2. Growth Types

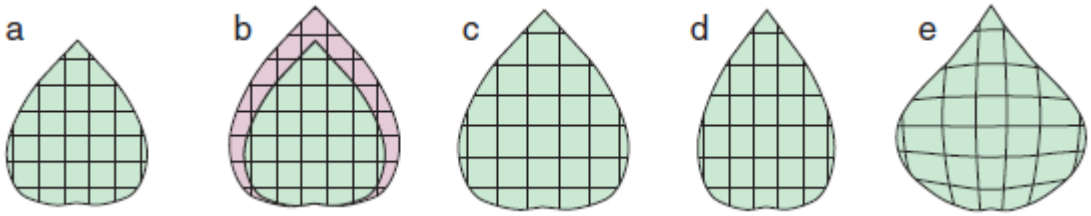


Figure 4: Growth Types (from Runions (2005))

In the above figure (2), if the leaf (a) represents the starting shape of the leaf, and the grid within is its growth tensor field (a representation of the position of points inside the leaf), then:

(b) Marginal growth. As you can see, the margin grows bigger, but the tensor field is not scaled. This means that area is added to the margin of the leaf without moving points within it.

(c) Uniform isotropic growth. The margin has grown, as in marginal growth, however the grid has similarly scaled up, meaning that the contents of the leaf have shifted outwards as well. Uniform growth is constant for each growth step, whilst isotropic means that growth is equal in all directions.

(d) Uniform anisotropic growth. The margin is extended, but the growth rate is not the same in every direction. The growth rate for anisotropic growth is usually defined as two rates with respect to the x- and y-axis. The tensor grid, and therefore the contents of the leaf are similarly unevenly scaled. The growth is uniform, so it is constant per step.

(e) Non-uniform anisotropic growth. Not only is the growth rate different depending on direction, it is different per step. The growth per step is typically represented by a graph.

Unless otherwise stated, growth should be assumed to be uniform.

1.2.3. Growth Hormones and Pigmentation

As this thesis uses biological models to create leaves, there are references to specific interactions with elements within the leaves that cause behaviour in growth and colouration. These are taken from (Rodkaew, Chongstitvatana, Siripant, & Lursinsap, 2003) and (Sanger, 1971)

Growth Hormone

The growth hormone modelled in this paper is known as auxin. It fulfils a large amount of roles within plant growth. It is present within leaves as they grow, and current studies suggest that as it moves in from the leaf margin toward the auxin creates channels that eventually form veins. This is known as the *canalisation theory* (Sachs, 2003). This is described in greater detail when venation is discussed.

Colour pigments

Three main pigments define leaf colour. These are chlorophyll, carotenoid, and anthocyanin which colour the leaf as well as perform other, unrelated tasks. (Sanger, 1971). Chlorophyll is green, carotenoids are orange or yellow, and anthocyanins are red or purple.

Leaves contain all of these pigments when they are green, however, when the leaf starts to die, it releases acidic substances from the blade. This destroys the chlorophyll, leaving the carotenoids and anthocyanins. As the leaf dies further, the carotenoids are converted into sugars and more anthocyanins are created. This changes the leaf into its final red form. When the leaf truly desiccates and dies, it turns brown.

1.3. Previous system

This project builds on a previous system written for these theses (Black, 2011) (Goldberg, 2011) (Danoher, 2011). The system is called Tree Draw. It allows the user to create 3D trees from a sketch-interface.

The system takes the user sketch and parses its structure to create an L-System that represents the phylotactic (structural) details of the tree.

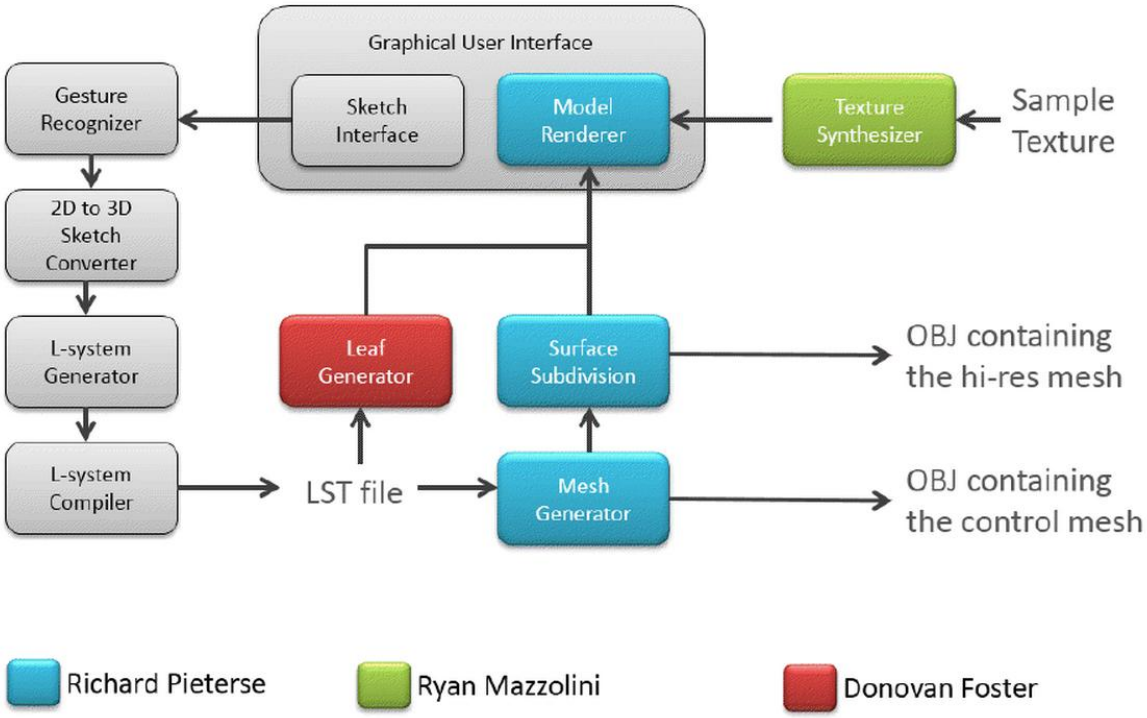
A second part of the system takes the constructed L-System and creates a specification for a tree. This specification is then interpreted into a set of cylinders, which the system then displays.

A specific goal of the system was the ability to allow for variation in the models generated. At sketching stage, the user could set a range of values of branch size or angle for each branch in the sketch. The system could then make multiple trees from a single sketch, each one being different, but related.

1.4. System Overview

This thesis is part of a project that is intended to increase the realism of the existing Tree Draw system. The shortcomings of the previous project were noted and three different elements were identified as needing improvement or implementation.

This project therefore sets out to research realistic methods to model these different tree generation elements, while keeping the axioms of the Tree Draw system with respect to usability and variation. The elements identified were; the lack of leaves and foliage; the lack of realistic bark texturing; and the lack of a single conjoined tree mesh. The combined system is shown below.



Each colour represents where each group member's modules are placed in the system. The grey represents the existing Tree Draw system.

1.4.1. Mesh Generation and Surface Subdivision

The mesh generation module takes the input originally given to the model generator in the Tree Draw system. It uses this data to create a joined mesh representing the tree and its branches. This mesh can then be textured, however it is quite coarse and unnaturally shaped.

The surface subdivision module takes in the generated mesh and splits face into smaller faces. It then uses these smaller faces to smooth and refined the surface until the mesh appears continuous. (Pieterse, 2012)

1.4.2. Texture synthesiser

The texture synthesiser module performs two tasks. Firstly, it creates a new seamless texture similar to one given by the user, but on a larger area. This process is known as

texture synthesis. This also allows for variation, as each tree can have a similar but different texture.

The second task is the attachment of the texture to each branch of the tree mesh in such a way that there are no seams. (Mazzolini, 2012)

1.4.3. Leaf generation modules

The leaf generation modules can be split into two logical pieces. The first is the leaf generator, and the second is the foliage generator.

The leaf generator creates a leaf from a sketch, first forming veins by slowly growing the leaf margin from a small size, every step growing veins inside the leaf. It then creates the colouration and texturing based on user given parameters. Lastly, it compiles this information into a texture material file that can be used in a modelling program.

The foliage generator takes the tree specification given by Tree Draw and a selection of leaves and creates leaves at ends of each of the branches. The size, number and orientation of the leaves depends on the given parameters.

1.5. Research Question

The research goal of the entire project is to improve the realism of the Tree Draw system. With consideration to the elements of Tree Draw identified as needing improvement, the specific research questions for this project are listed below, with number three being the question specifically answered in this thesis:

1) Can branching structure be more realistically modelled by sub-division surfaces?
2) Can texture synthesis be used to generate a realistic texture of bark that exhibits variation from a provided sample?
3) Can realistic leaves be generated from a sketch and allowing for variation?

The results of the research question are shown in the result section of this thesis. Unfortunately, formal user-testing was not possible due to external factors, however several runs have been created and compared to real leaves. Informal user-testing was applied throughout the design and implementation process.

1.6. Novel contributions

The novel contributions in this paper centre around variability. Trees are very complex and time-consuming to create by hand, and it is unreasonable to expect anyone to be able make several different trees. Last year's project, upon which ours is built, focused on the ability to generate multiple trees from a single sketch. This paper expands upon this idea by presenting the user with a way to create multiple related leaves to be used on a tree, all generated from a single sketch. Additionally, the user can attach these leaves to multiple trees without having to change the settings or parameters, allowing them the ability to create multiple, realistic trees.

1.7. Thesis Structure

After the introduction section, this thesis splits into two different sections, each of which contain an introduction, background, design and implementation chapter, as well as results. These two chapters are split because of the logical and coding separation between leaf texture generation and leaf foliage generation. As a leaf can exist off a tree, and foliage can be created from any leaf texture, it is possible, and advisable to look as these two entities separately.

Afterwards, the results of the system are shown, followed by the conclusion and future work section. The addendum after the bibliography contains a detailed explanation of the interface created for the system, as well as a detailed explanation of the parameters used.

2. Leaf Texture Creation

2.1. Introduction

Leaf textures are the image files that define the visual aspects of the leaf. Textures can define more than just the colour of a leaf. While diffuse textures are the most common, specifying the colour and shape of the leaf under normal (diffuse) lighting, bump or normal maps can define a fake geometry for the leaf, which is used to create shadows and detail without affecting the mesh. Specular maps affect how the leaf reacts to the specular part of a light source, which determines how a leaf shines at any given point.

There are many other types of texture that can be applicable to leaves, such as transparency maps, which dictates how much light can pass through a leaf, which is another characteristic of real leaves (Wang, Wang, Dorsey, Yang, Guo, & Shum, 2006). This was not included in this paper because the model format exported to (.obj) does not support transparency maps natively. A transparency map could be trivially created by finding the difference between the top and bottom bump maps. Since these represent the height of the leaf at that point, the difference between the two must necessarily be its thickness. The creation of soft bodies, such as leaves, as a field of study is relatively undeveloped when compared to branching or bark structures. This is due in part to ease of which pictures of leaves can be obtained and used without alteration, as well as the complexity of creating a leaf texture from scratch. Additionally, unless the viewer is placed extremely close to the tree, the details in leaves are unlikely to be visible, and so overall colour and shape is often more important than accuracy, and is not worth the cost of rendering involved.

2.2. Background

There are many different ways of creating effective leaf textures, each with their own strengths and weaknesses. The inherent fractal nature of leaves means that they are well suited to procedural generation techniques such as L-Systems. However, there is also an almost infinite amount of complexity in nature, and thus it can be argued that sketch-based and user-controlled methods can provide greater control and accuracy, at the cost of less automation of the process, as well as less potential for automatic variation. Several methods have been proposed for texture generation, but most can be broken down and mixed together within these three categories:

2.2.1. Shape Definition

The outline of the leaf, used to grow a leaf texture, or fix to a mesh, can be found of defined from a picture of a leaf, taken in from a user sketch, or defined using parameters. As a method is mentioned, it will be numbered as such: [i], this number corresponds to its index in the comparison in Table 1 below.

Image-based

Image based is likely the most accurate of the methods for shape definition. This was the first method used, referenced as far back as *Modelling the Mighty Maple* (Bloomenthal, 1985), where the leaf textures were captured and edited by hand before being placed on the tree [1]. This is the most accurate method, however is the most time consuming, and limits the amount of textures one can generate to leaves that can be found. The methods below are broken down depending on whether one wishes to use the picture of the leaf as the texture afterwards, or whether one is simply interested in the shape. While the outline is unlikely to be an important outcome by itself, once found it can be either redefined parametrically, or simply stored as a point cloud to be used in later stages of leaf modelling.

(Mundermann, MacMurchy, Pivovarov, & Prusinkiewicz, 2003) used a simple system of shape definition from a scanned image on a white background. Each pixel of the leaf that is touching a light background is captured and stored as the program crawls around the outside of the leaf, resulting in the outer perimeter. They then cull the least important points, turn the image shape, and create a B-spline curve, a parametric representation of the leaf, which they can use later [2]. Their use for this, a mesh creation phase, is discussed in the section 3.

This method has several flaws, which are flaws of image-based shape definition in general. The picture must be well scanned and in focus for the program to work. Equal, the image has to be pre-processed, removing any flecks from the image that the code might pick up whilst also changing the saturation and brightness to give the best outline. Additionally, while most leaves are well suited to being scanned, being largely planar in nature (de Reffye, Edelin, Françon, Jaeger, & Puech, 1988), there are leaves which cannot be flattened without damage, removing the value of the shape gained. Lastly, as this paper is stated to use a boundary following algorithm, it can miss holes within the leaf itself. This is an issue whether the texture is intended for use or not, as the hole in this case will be white and very noticeable. Mundermann's suggested fix is the ability to edit the b-spline once is it created by moving the points that define it, however this does remove its ability to generate outlines autonomously.

Another possible method is to recreate the shape in 3D. (Quan, Tan, Zeng, Yuan, Wang, & Kang, 2006) developed and tested a method where an entire plant is modelled from several photographs taken at precise angles around the plant [3]. While they admit that the methods could struggle with accuracy over an entire plant, it does show promise when the object to be modelled is against a stark background. This could make it invaluable for image-based shape definition in the future. Though currently it is not accurate enough, it is not difficult to see a future system that can be given a plant and retrieve all of the leaves in one run, for use on future models.

PROCEDURAL

Procedural shape definitions tend to act as the inverse of image-based techniques. Whilst image-based techniques work from the outside of the leaf, taking large amounts of data points and refining them, procedural techniques work form the inside out. The most common methods define a skeleton framework, much like a tree, and then build an outline around it, using parameters to determine specific details such as edge serrations or point shapes.

The most common system for this category is an L-System. As described in (Prusinkiewicz & Lindenmayer, The algorithmic beauty of plants, 1991), the internal structure of a leaf is not unlike that of a tree, with strict branching structures. In fact, due to largely planar and fractal nature of leaves, L-Systems can be an even better approximation than trees, for certain leaf types. Lobed leaves are particularly well suited to L-Systems, as well as compound leaves (Prusinkiewicz, Hammel, Hanan, & Mech, 1996, February) [4]. However the diversity of leaf types in nature means that while individual types of leaves can be successfully modelled using L-systems, it is not feasible to create an L-System for any kind of leaf. Additionally, creating an L-System for a leaf type is not trivial, and requires more skill than the ordinary user may possess, leaving this in the hands of professionals.

An added complexity exists in that the same L-System used to describe the shape may also be a good approximation to the venation of the leaf itself, as suggested by (Prusinkiewicz & Lindenmayer, The algorithmic beauty of plants, 1991). This can either sufficient or completely inadequate, depending on the type of leaf. This is discussed in greater detail when venation is discussed below.

It is also possible to take advantage of the fractal nature of leaves and define a shape that is inserted into itself to form a fractal leaf. The maple leaf is one such example of an approximately fractal leaf.

SKETCH-BASED

Sketch-based shape definition is an amalgamation of both image-based and procedural techniques. Which it resembles closest depends largely on the intended outcome of the sketch.

Sketch-based techniques in which the user draws the outline are very similar to image-based techniques, as the resulting data is a collection of points to be processed. This technique is used in (Okabe, Owada, & Igarash, 2005) can be used to simply draw the outline of the leaf [5]. This technique is not the subject of many papers, due to the simplicity of its implementation, however it is a practical and powerful way to provide input from a user that is easy to understand for both the system and person, and is the first step in many papers that deal primarily with the later stages. (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005) is one such example. The greatest drawback with this method is that it relies on the drawing skill of the user, as well as the same drawbacks that sketch-based interfaces have around difficulty in creating variation. Runions attempts to mitigate the skill factor by having the user define points for a curve.

In the other direction, sketches are frequently used to specify parameters for procedural generation. A good example of this method in several forms is found in (Anastacio, Prusinkiewicz, & Sousa, 2008), where sketches are used for defined the relative size of leaves on a compound leaf (by sketching a bounding shape) [6]. They are also used for defining graphs that give growth over time, as well as growth for each axis, and as a method of defining the phylotaxis (structure) of a compound leaf, showing where each leaf should be with a sketch of a grid. The greatest problem with this system is the complexity. It takes a great deal of internal logical to convert sketches into meaningful constraints. Even though Anastacio had good results for some of their goals, the most difficult parameters remained undeveloped at the end of the paper.

COMPARISON

This table below summarises the advantages and disadvantages of using a particular technique.

Method for defining leaf shape	Computation cost	Accuracy	Usability	Potential for variation
[1] – Manually from picture	Low	High	Low (manual work)	Low
[2] – Travel around boundary of picture	Low/ Medium	Medium/ High	Medium (good when it doesn't fail)	Medium (affine transforms of outline possible)
[3] – Get 3D shape from multiple images	High	Low	Low/Medium (could be automated)	Medium (No actual variation, but many leaves gained)
[4] – Outline around L-System	Medium	Medium/ High (dependant on leaf)	Low/Medium (If the parameters are pre-set)	High
[5] – Sketching an outline	Medium	Medium (depends on user skill)	High	Medium (Affine transformations to outline)
[6] – Sketching parameters	Medium/ High	Medium/ High	Low/Medium	High

Table 1 : Comparison of Shape Definition Techniques

The technique for shape definition used in this paper was derived from sketching the outline. This was due partly to the fact that the preferred system, creating an outline from an L-System, was out of scope. However, it was also felt that the sketch interface fitted well with the sketch interface in the previous project for drawing trees. An option to load an outline from a two-colour image file was added to mitigate the reliance on user skill.

2.2.2. LEAF VENATION

Leaf veins are an important element in leaf texturing. They are the most noticeable trait of a particular leaf, after shape, and cannot be ignored. There are relatively few good ways of creating veins, which are discussed below.

L-System based veins

As stated above, one of the advantages of using L-Systems to define leaf shape is that the same structure can provide veins for the leaf. This is very much like modelling leaves as planar trees. This is covered in detail by (Prusinkiewicz & Lindenmayer, The algorithmic beauty of plants, 1991). Prusinkiewicz continues to use this technique in his papers (Boudon, Prusinkiewicz, Federl, Godin, & Karwowski, 2003). One flaw in this system is that it limits the leaves that are able to be made to those that can be generated with L-Systems, which limits the number of different leaf types possible.

Biologically –motivated venation

These algorithms are based around creating vein as a simulation of the natural processes found in leaves. As such, they take many iterations to complete and can be very computationally expensive. The algorithms involve the movement of the growth hormone, auxin, from within a growing leaf to down into the tree. As the auxin travels, it creates veins in the leaf. This is known to botanists as *canalisation* (Sachs, 2003), due to the fact that the auxin creates veins in a way similar to how water creates rivers and canyons.

The first attempt at modelling this process to generate veins was by (Gottlieb, 1993), which still holds with current knowledge of vein structure. Its use of grids for auxin position definition makes it difficult to use for image generation however. This process was adapted and implemented by (Rodkaew, Chongstitvatana, Siripant, & Lursinsap, 2003). Auxin is placed at random into a full leaf outline. At every iteration, the auxin move toward the average of their nearest neighbour and the root of the leaf. As the auxin moves, it creates a vein behind it. If it reaches a vein created by a previous auxin, it is destroyed (carried away by the vein). Once all of the auxin are destroyed, the vein width is calculated as the sum of its children vein's energy (where the end veins have an energy of 1).

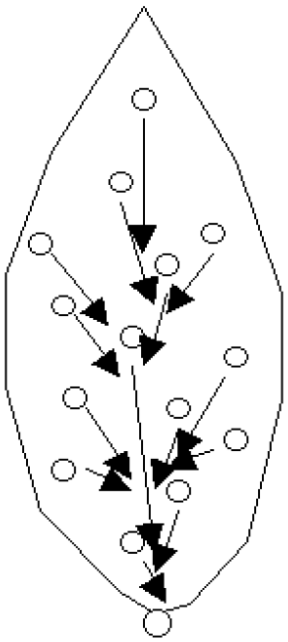


Figure 5: Auxin Growth toward root (Rodkaew et al, 2003)

This algorithm gives good results, with very high realism. However it can break under several different leaf conditions, where the root is not in sight of the node. Additionally, it is difficult to improve upon their process as it deviates from what is known about the natural processes.

(Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005) suggested an improved technique that deals with many of the issues of the previous paper. In their paper, the leaf starts at a small size and is expanded every iteration. During an iteration, auxin is spawned randomly throughout the leaf. The auxin find the closest vein, however, in this algorithm, it is the vein that grows toward the auxin, this is a better simulation of the process in real plants and creates several speed up opportunities, mostly in the nearest node search patterns, that take the execution time down from hours to seconds per leaf.

The fact that the leaf grows also allows the incorporation of different types of growth, each modelling a different leaf growth pattern. These have the effect of elongating, or shortening the veins, increasing or decreasing branching and other effects that increase realism. Unfortunately, the implementation complexity of non-uniform growth (where the growth changes per iteration according to a function) made it future work for their paper. This is unfortunate, because it is the most general model of growth (any uniform growth can be modelled using non-uniform growth systems with constant growth).

These algorithms can provide extremely realistic venation, but struggle to simulate leaves that have symmetric venation, as they do not look at the overall venation structure. However, with the right parameters, specifically larger initial leaf sizes, the algorithm can gain fairly straight and symmetrical venation.

As may be concluded from the title of this paper, the method chosen for the leaf texture venation was taken from the biologically- motivated algorithm by Runions, as it provided very interesting opportunities for variation, in that the growth of the veins depends entire on the growth of the leaf and auxin. This means that potentially, unlike the L-System, which must be changed, any leaf can be modelled with this algorithm, if the actual implementation is accurate enough.

2.2.3. COLOURATION

Leaves are typically coloured in two main ways, from an image, or procedurally. (Wang, Zhao, Lu, & Guo, 2009).

IMAGE-BASED

Getting colour data from an image is a viable option, and one that occurs frequently, as in (Reche-Martinez, Martin, & Drettakis, 2004), which takes colour of leaves scanned while doing a 3D reconstruction to use as the base colour of the reconstructed leaf.

Image-based techniques assume that leaves on a single tree are similar enough that it is difficult to see that they are, in fact, the same leaf. (Mundermann, MacMurchy, Pivovarov, & Prusinkiewicz, 2003) attempt to mitigate this fact by changing the shape of the leaf using deformation of the underlying mesh. (Mochizuki, Horie, & Cai, 2005) propose a system that runs through the foliage structure once it has been generated, and discolours leaf textures

using ray tracing from a sun point about the tree, which is expensive, but could provide good results.

PROCEDURAL

Procedural techniques use the underlying structure of the leaf to inform the colour at a point. This includes veins, edges and orientation.

The most basic form of this technique is applied in (Rodkaew, Chongstitvatana, Siripant, & Lursinsap, 2003). In this paper, leaf colour is set to a solid shade, to which is added noise. Veins are added to the texture and the vein colour is blurred out away from the veins. This can make a fairly convincing leaf texture. The purpose of the noise in this paper is to simulate smaller veins that have not been modelled.

The simulation of autumn colours has been studied for some time. (Chiba, Ohshida, Muraoka, & Saito, 1996) was one of the first to look at how the observed effects of autumn leaves could be modelled. The basic colours are set similarly to those above; however afterwards, the aging process is applied to the leaves, warping their colour. Their paper contains a similar model to (Mochizuki, Horie, & Cai, 2005), where the leaves are first placed on the tree, and then the colour is recomputed. The effect of the sun on a leaf is that it eventually destroys the green chlorophyll in each leaf, leaving behind the yellow carotenoid, and eventually only the red Anthocyanin. This turns the leaf colour slowly red. This is modelled after studies done on real leaves, such as (Sanger, 1971).

This is taken to the final logical step in (Zhou, Dong, & Mei, 2006), where the colour of maple leaves is defined by a graph of concentration for each of the three pigmentations over time. These graphs are then recalculated based on numerous factors, such as sunlight strength, climate and precipitation. This allows the creation of a very realistic leaf. However, this implementation is specific to maple leaves, and those few others that exhibit this particular seasonal change.

A simple model for simulating damage to leaves due to frost, hail or other environmental factors is laid out in (Mochizuki, Horie, & Cai, 2005), where damaged points in leaves are treated as if they were in direct sunlight, discolouring them and deforming the leaf at that point.

For this thesis, the chosen colouration scheme was modelled after the one used in (Rodkaew, Chongstitvatana, Siripant, & Lursinsap, 2003), as the internal structures of the leaf shape are very similar to those used in the paper that is the basis for venation (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005). While it would be best if seasonal colour changes were also included, systems such as (Mochizuki, Horie, & Cai, 2005) are not in the scope of this project. However, a modified and simplified version of (Zhou, Dong, & Mei, 2006) was used to simulate seasonal leaf colour changes and leaf damage, by estimating pigmentation level based on hue values.

2.3. Design and Implementation

2.3.1. Process Flow Diagram

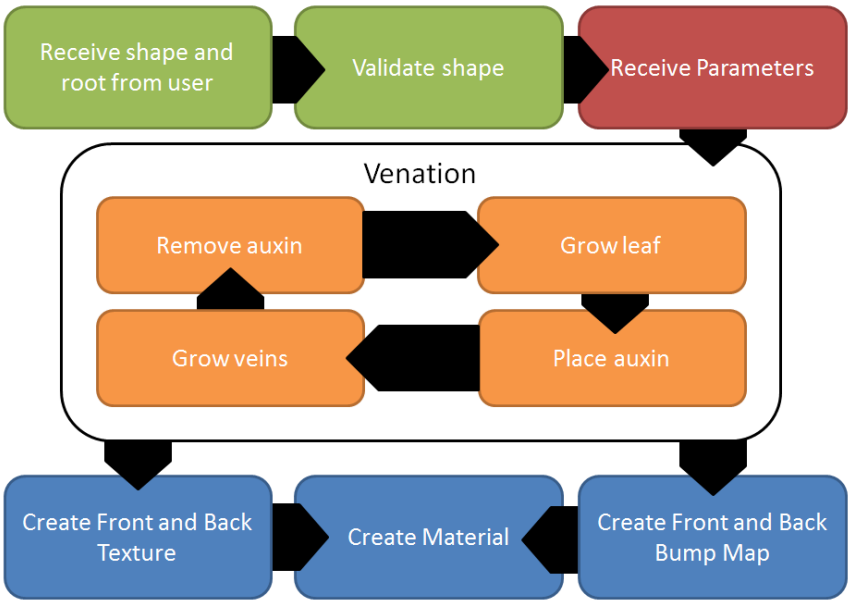


Figure 6: Texture Creation Process Flow Diagram

Where each colour represents a different module

2.3.2. User interface

The user interface for this section contains the sketch interface and parameter selector. A diagram and explanation of their use can be found in the appendix (Section A.1)

2.3.3. Shape Creator

The shape creator defines the root and outline of the leaf that is used in the venation and texturing processes. It differs from the style used in (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005) which defines its shape as the starting shape of the leaf, which is then expanded into the final leaf. This thesis uses instead a sketch-based interface for the leaf shape definition. This means that the input into the rest of the system is the final size of the leaf, from which the starting size must be extrapolated as a function of the number of steps the leaf must grow for. Each approach has advantages and disadvantages. By defining the starting shape, Runions stays closer to the biological model the leaf uses, which in turn creates more realistic leaves. It is difficult to foresee the end result of a particular growth however. By contrast, taking the final shape of the leaf gives greater control of the final look of the leaf, but the leaf itself is somewhat less biologically accurate as a result. Additionally, it allows the system to calculate the texture size in advance, since the maximum size of the leaf is known.

Receive outline and root from user

The user may sketch a leaf shape in the sketch interface using the pen tool and eraser provided. Alternatively, the user may choose to load the outline from a picture. The picture is converted into a two colour picture as follows:

The program takes the colour value in the top left corner as the white value. It then creates a new image, and iterates through both the old and the new side by side. If the pixel in the old picture is the white value, then the pixel in the new picture is made white, otherwise the new pixel is made black. The need for a more complex conversion was not felt needed, as it is not difficult to format the image into the appropriate style in any image editor.

The image loader does not find the outline of the image, only doing a pixel-by-pixel conversion. This means that the loaded outline may not be just a single pixel thick and may even be a solid black shape. This can be fixed in the program using the pen and eraser tool if required, or alternatively simply reformatting the file and loading again.

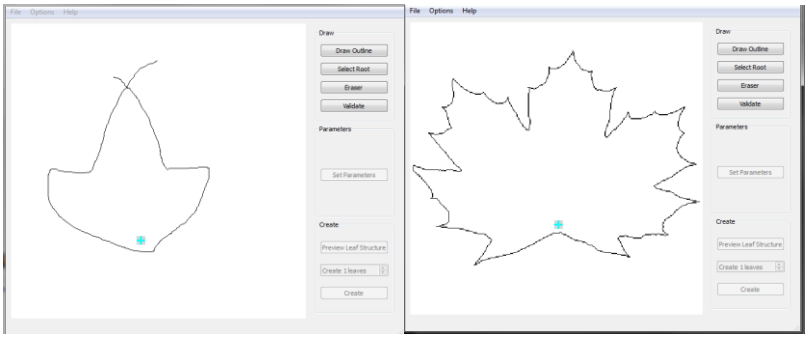


Figure 7: User Drawn Sketch (left) and successfully loaded leaf (right)

Any sketch created in the sketch interface can be saved to one of several file formats (defaulting to .png) for use in another session.

The user then selects the position of the root using the select root tool. Clicking anywhere on the sketch area will place the root at that location.

VALIDATE SHAPE

Once the user is satisfied with the design, the validate button is pressed. This initiates the validate routine ensures the outline is whole and contains the root.

The process starts at the root node and checks that it is white (and therefore not on an outline). If so, the pixels above, below, left and right of the pixel are added to a stack. The front pixel is popped off the stack, and if it is not an outline, the pixels around it are added to the stack and the pixel is marked as visited. The process then continues until the stack is empty. This is a simple flood fill procedure.

While the process is running, two conditionals are checked on every pixel. If the pixel checked is an outline, its position is added to the shape definition. If the pixel is off the canvas, this implies that the outline is not whole, and the outline is not valid.

If the outline runs its full course without being invalidated, the shape definition is created as a list of vectors, each one representing the vector to an edge pixel from the root node. For example (10,2) represents an outline pixel 10 to right and 2 down from the root. The outline is stored as such so that the outline can be grown out again easily. The outline will pick up on line drawn within the leaf area. These are stored as well and can be used if a leaf has a very close divide, of serrations.

If the leaf is valid, the parameter menu will be enabled and the interior of the leaf will turn green, both as an indicator that the validation is complete, and also to show the recognised shape that the leaf will use. As the process attempts to find the single line outline around the root, it does not pick up any closed shapes away from the root. It does not find pixels across diagonals either. This is to prevent spilling over the outline unintentionally. However, should the user be dissatisfied, they can edit the outline and try again.

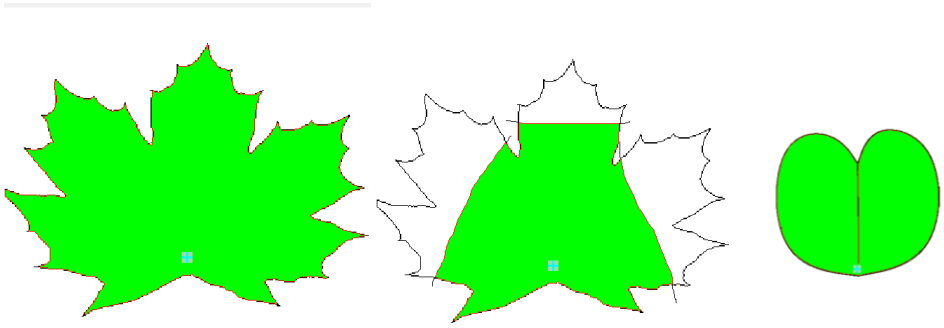


Figure 8: Validated leaf (left) Validated interior only (centre) Validated with dividing line (Right)

2.3.4. Parameters

The parameters are collected from the parameter selection interface. Each parameter is set with a minimum and maximum value. This choice allows for variation when multiple leaves are created from one outline and parameter set.

A detailed explanation of each parameter can be found in the appendix (section B).

A detailed explanation of the parameter selection interface, as well as the two panels used to input parameters can be found in the appendix (section A.2).

The parameter selector can save and load text files with the parameter data. This allows the recreation of previous leaf parameters. Along with the ability to save an outline, this allows the user to recreate leaves later if so required, although the venation and colouring will still differ slightly, as they are regenerated.

Once the user is satisfied with their parameters, they can select OK to return to the main interface. This unlocks the next set of options: Preview Leaf, which creates a single top texture using the parameters supplied, and create leaves, which, along with the number of leaves required, starts the leaf venation process (once per leaf).

2.3.5. Leaf Venation

This module takes in the leaf shape, as well as the growth parameters explained in section A.2, but specifically, those affecting leaf and vein growth and auxin placement and death, creates a vein structure for the leaf. The veins are stored as a tree with the root being the root of the leaf. The auxin are simply stored in a list.

The initial state of the leaf is created, with the root as the centre of growth and the margin at step 0 is set up from the outline vectors. For marginal or isotropic growth, the starting positions for each point of the margin are set as

$$p_0 = v * \frac{1}{G_t + 1} + r$$

For root point *r,* outline vector *v* and total growth G_t. This ensures that the margin starts with area.

If the growth is anisotropic, the x- and y- growth rates are different. This means that the number of steps taken to grow horizontally differs from the amount of steps vertically. The solution suggested in (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005) works well for an input of the starting outline, but does not help with this format.

The solution is to take the growth steps for x- and y- and find the highest. This is then made the total amount of steps. The difference between the highest and the lowest is the number of steps the other growth rate does not complete. Since the end result of this growth is defined, it must be that the lower growth steps started at (highest-lowest) steps closer to their goal.

The leaf must be then be adjusted so that the start positions allow the margin to grow to right size in the same amount of steps for both x- and y- growth.

For more detail, see the appendix (section B.2).

If there is no growth, the margin is just set to the maximum leaf size.

Similar assumptions are taken to those from (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005). Specifically, it is assumed the veins and auxin create a feedback loop, where veins affect auxin placement and destruction, and auxin in turn control vein growth direction and development.

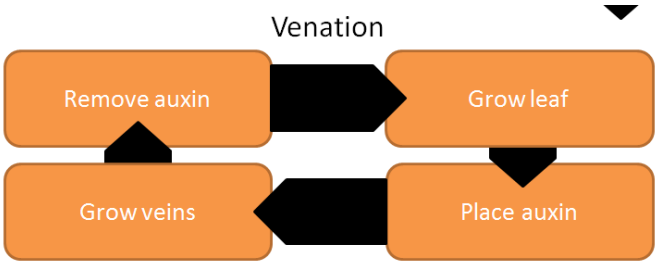


Figure 9: Venation cycle

Once the parameters are in place, the process starts to create veins. The program runs for a number of iterations, (total growth - growth start value)*scale with step value *n*. *n'*s value starts at (start growth)*scale and ends at the value of the (total growth)*scale. At each iteration, the cycle is run through as below.

Once the iterations are over, the system stops, no matter how much auxin is left.

LEAF GROWTH

For root point *r,* outline vector *v* and proportion of growth completed after *n* steps of total growth G_t, the position of the outline is calculated as

$$p_n(x,y) = (v_x * \frac{n}{G_h} * G_t + p_{0x}, v_y * \frac{n}{G_v} * G_t + p_{0y})$$

If the growth type is *isotropic* or *anisotropic*, the veins and auxin are shifted along with the margin, so that for each vein or auxin point p_{n-1},

$$p_n(x,y) = (p_{n-1} * \frac{n}{G_t}, p_{n-1} * \frac{n}{G_t})$$

With an assumed "starting position" of $p_{n-1} * \frac{1}{G_t}$ for each iteration

AUXIN PLACEMENT

The auxin is placed randomly according to the dart-throwing algorithm. If the auxin lands within the spawn distance from a vein or leaf, it is destroyed.

VEIN GROWTH TOWARD AUXIN

Each auxin creates finds its closest vein and stores the normalised vector from the vein to itself in a list. If a line from the closest vein to the auxin leads through the outline, the vein is not added to the growth list. The line check uses Bresenham's line algorithm to check a straight line between the points. It initially checked each node until the closed non-blocked node was found, but this was far too expensive. It now checks only at the end, and if the node is not visible, it simply does not add it.

The program then iterates through the list, for each vein that has been added at least once. It adds up the all the vectors that point towards one vein node and then normalises that vector.

Lastly, a new node is created from each vein node mentioned in the list, at the position of its parent plus the vector.

Occasionally the vector will be less than 1 before it is normalised, this happens if a vein is between two auxin. If this happens, the first vector is removed and creates a new vein node from the target vein node in the direction of its normalised vector. This repeats until the averaged vector is greater than 1, or the list is depleted.

AUXIN DEATH

The auxin each check for the distance to the closest vein node to themselves. If it is less than the kill distance, the auxin is destroyed. This test does not worry about line of sight.

EXAMPLE

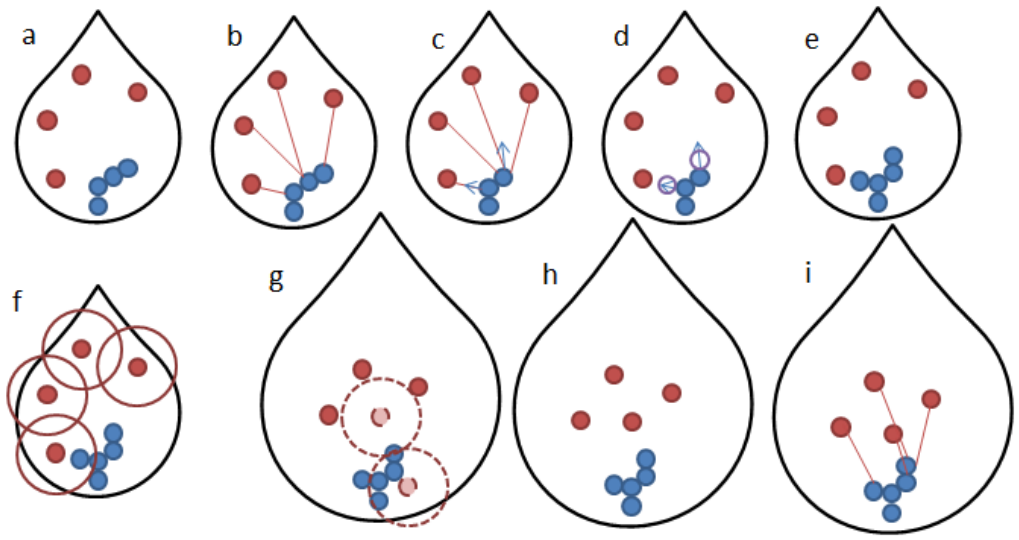


Figure 10 : Example of marginal vein growth

(a)Represents the current leaf, midway into. At (b) The auxin each find their closest vein. (c)The veins find the normalised average of their target auxin. (d) The vein nodes create new nodes along the vector. (e) The veins are confirmed. (f) The auxin check to see if they are outside of the kill radius. One is removed. (g) The leaf grows and adds more auxin. The auxin check to see if they are outside of the spawn radii. (h) The Auxin are confirmed, one is removed. (i) The cycle starts again.

The vein for a vein node *v* is determined by the following value using Murray's Value (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005):

$$v_{width} = \sqrt[3]{children^3_{width}}$$

2.3.6. Texturing

The texturing stage makes two textures, one for the front of the leaf, and one for the back. It also makes two separate bump maps. The texture for the back of the leaf is flipped, so that is matches with the front when back-to-back. The front bump maps has the veins slightly recessed, while the back bump map has them stick out over the surface, this is based on real leaves (Judd, Campbell, Kellogg, Stevens, & Donoghue, 1999).

Figure 11: Front and back textures and bump maps

See the appendix for more on texture values (B.3).

TEXTURING

The texture is created by making a new image an filling the interior of the outline with the base colour. The colouration is then applied to the leaf. If the leaf has chosen a colour type, the colour is applied. For each colour type, the value of the colour at point *p,* a point near the target that is creating colour is a function of the distance from the target *t*, and spread *s* and the colour values *base*, the existing value and *alt*, the new value, as follows:

$$p = (\frac{dist(p,t)}{s} * base + \left(s - \frac{dist(p,t)}{s}\right) * alt)$$

Which means the new colour is slowly lost the further is from the target point. The colour value is the average of the HSV colour values of the two points. Colour is only applied to the interior surface of the leaf.
From Veins: The alternate colour is spread from each vein node outward at (*spread**root width).

From Edges: The colour is spread out from each edge point to the maximum of *spread* distance.

Speckled: Points are created using the dart-throwing algorithm. 1/10 of the parameter for auxin placement points are placed. Colour is then spread out from these points to a distance of spread.

Any growth type not applied is still used, with a colour value just less than the base colour to give the leaf texture and simulate the small veins not made during the venation process.

The petiole is then added to both textures. It is drawn on top of the texture for the back texture and drawn underneath the texture for the top leaf.

SPECULAR VALUE

The specular value taken from the parameters is added to the leaf material.

NORMAL MAP

The normal maps are height maps that use the red value to indicate height from 0 to 255. They are created by making two new textures and filling in the blades with red (128). It then draws the veins onto the leaf, higher up for the bottom texture, lower down for the top texture. It then speckles both textures and draws the petiole at the same height as the roots according to each texture.

MATERIAL

The all of the materials are then placed into a material file, .mtl, used by .obj objects for texture data. They include both textures, their height maps and specular value parameters. An example material can be found in the appendix.

3. LEAF MESH CREATION AND PLACEMENT

3.1. Introduction

This section places the completed leaves onto a tree mesh. The position of the leaves is found from the LST file produced from the Tree Draw system. This file contains the rotations and translations required to get from the root to every branch. These are stored in a tree structure. The full specification can be found in the appendix.

3.2. BACKGROUND

3.2.1. LEAF MESH GENERATION

The preferred method of leaf meshing is that of (Mundermann, MacMurchy, Pivovarov, & Prusinkiewicz, 2003), wherein each leaf is placed onto a full mesh made from an internal web created using the leaf's outline. This was however, far beyond the reach of this project.

Another, simpler alternative was the simple mesh used in (Bloomenthal, 1985), which created three quads that could hinge in such a way as to give a good impression of shape.

3.2.2. FOLIAGE CREATION

Foliage creation for this system was hampered, mostly because the favoured methods for foliage generation involved the use of bounding hulls. (Deussen & Lintermann, A modelling method and user interface for creating plants, 1997) and (Chen, Neubert, Xu, Deussen, & Kang, 2008).

Additionally, due the enhancements created in the mesh joining part of this project, this thesis was unable to change the branching structure, or easily get an accurate point on the tree, other than the end points.

Under these conditions, and taking scope into consideration, the final foliage generation scheme was taken from (Bloomenthal, 1985)

3.3. Design and Implementation

3.3.1. User interface and parameters

The parameters are taken in from the user interface, including the LST of the file to be used, and the material files for the leaves. The LST files the file type used by Tree Draw to define branch structure.

The user is able to control how many leaves spawn on a single edge node, how far away these branches spread from each other, and how big they are. The angle of the leaves downward is calculated as a function of their size.

For each leaf material file, the user can place a value, determining the ration in which the leaf is to be assigned to a branch. The leaves are randomly picked with a probability of ratio/(total ratios)

3.4. Foliage Creation

3.4.1. Placement

The edge nodes are found by building out the tree for the LST file and then finding those nodes that do not have any children. The LST file is parsed using code from (Pieterse, 2012), after the tree mesh has been produced.

For each edge node, the program spawns the number of leaves set by the parameter. The leaves are arranged in natural looking groups using a heuristic as suggested by (Bloomenthal, 1985). Each leaf then picks a texture and calculates the right size.

The program creates a new quad, representing the leaf, and rotates it by the spread and tilts it according to its size (the bigger it is, the more is hangs down) . It then rotates it by the edge node rotation (so that it faces the same way as the branch), and is translated to the edge node position. The position is shift slightly to ensure that the petiole is touching the branch.

This quad and its data are then added to a list to be written to file.

3.4.2. Saving to file

The quads are written into an .obj file, following OBJ specifications. The top texture and bottom texture are mapped to the same 4 vertices, but in opposite directions, this ensures that one texture will display on top and one on the bottom.

A sample render of the result can be found in the testing results.

4. TESTING

4.1. PERFORMANCE TESTING

Leaf creation was timed from initiation of venation to the end of texturing and was initially found to take several hours for even a small leaf. The solution was to create a grid structure over the leaf that was used to quickly and efficiently add and remove veins and auxin, making finding the nearest neighbour much quicker. This resulted in algorithms that ran in under a minute.

Several tests were run to test the effects of changes in parameters using the sample leaf below. For details on the parameter, see the appendix.

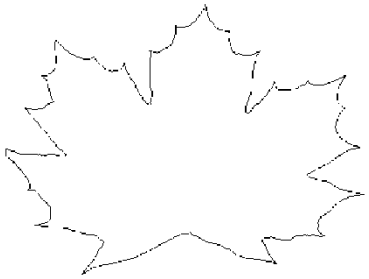


Figure 12: Test Leaf Outline

All unstated parameters are set to constant according to the list below.

Auxin Spawn Rate	600
Auxin Spawn Distance	5
Vein Spawn Distance	5
Auxin Kill Distance	5
Growth Type	Isotropic
Total Steps	600
Starting Steps	300

Tests to judge the effects of growth parameter changes are detailed below:

Spawn distance (both vein and auxin)	Time (secs)
1	34
5	25
10	11
100	9

Auxin Kill Distance	Time (secs)
1	43
5	25
10	18
100	3

Auxin Spawn Rate	Time (secs)
1000	212
600	25
400	12
100	8

Growth Type	Time (secs)
Marginal	38
Isotropic	25
Anisotropic	26
None	32

Total Steps	Time (secs)
600	25
1000	370
3000	>6000
5000	>6000

From these figures it is easy to see that the amount of auxin in the leaf greatly affects performance, this makes sense, as the most costly operation in the code is finding the nearest neighbour. When the kill distance is high, the auxin is never placed. Because the auxin is placed per step, increase the number of steps without accommodating for the change results in catastrophic generation rates.

4.2. Results

4.2.1. Venation

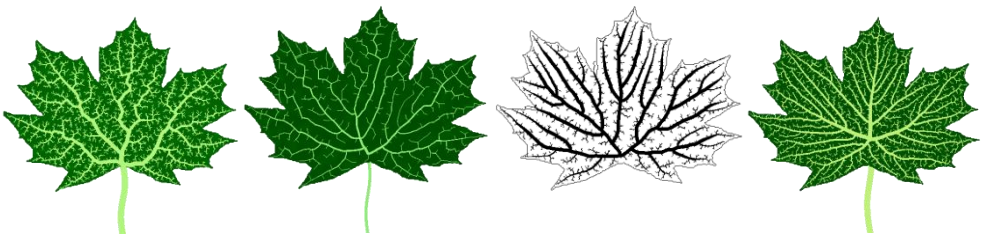


Figure 13: Growth types (Left to Right) Isotropic, anisotropic, marginal, none

As you can see, Isotropic veins can wander slightly, whilst the anisotropic grew up toward the top quicker. Marginal veins were slightly straighter, and no growth produced the straightest veins with more branching than marginal.

4.2.2. TEXTURING

Figure 14: Left to Right - From Edges (front texture), From Veins (back texture), Bump Map. Each taken from a different leaf created with the same parameters, but with variation

4.2.3. FOLIAGE

Figure 15: Foliage Render

4.2.4. Full Render of leaves compared to real leaves

Figure 16: Red Maple Leaf Rendered

Figure 17: Brown Maple Render

4.3. Conclusion

The system worked extremely well, and it was fairly easy to create leaves that looked like the specified pictures. The performance results were extremely encouraging. While changes in the growth rate took a toll on performance, this is not as big of an issue as it first seems. The leaf algorithm tends to work best in all areas at under 2000 total steps. Over that, the veins become too thick and numerous and the increments in growth are simply too small. This is because of the relatively small texture size chosen, which works well for rendering, but is perhaps a bit too small for generation.

4.4. Experimental Errors

The foliage algorithm only works on one PC. It is suspected to be caused by a conflict between 32- and 64- bit software, however the true cause has not been found.

5. Conclusion

Leaves and foliage are a complex, yet crucial part of procedural generation. Although these leaves have not reached the full realism of a hand crafted leaf, the shear variation of them make every tree unique and interesting. The true conclusion that can be drawn out of this thesis is that there is a reason that leaf generation is studied significantly less than other types of procedural generation: It is extremely hard and complex. Even a good system will only work for so many kinds of leaves. It is true that leaves follow well defined patterns that can be modelled with rules, however it is also important to consider that these rules are not the same for every leaf.

The results of the colouration were favourable, although there were some issues with extremely different colours and blending, this is more due to allowing the user to pick colours outside of the system tolerance and natural leaf colour variation. However, for the most part, the choice of HSV created a good approximation for leaf pigmentation presence.

The foliage creator produces adequate foliage. However, it was not all that foliage generation can be. Unfortunately, the limitations set by the previous system meant that foliage could not be generated using the most popular current method: convex hulls. In many newer papers, trees are generated by defining the trunk and a bounding hull for the foliage. The leaves then populate the hull and branches are grown back from the leaves to the trunk, in much the same way as veins are formed in this paper. This result works best when the user cares about the overall shape of the tree, rather than biological accuracy. (Chen, Neubert, Xu, Deussen, & Kang, 2008)

As a research into the methods of generating leaves, this thesis can quite clearly show that it is possible to generate realistic leaves using biologically motivated algorithms. For a large percentage of leaf types, this system will produce good results, particularly for venation.

6. Future Work

There is much room for future work on this system. The implementation of both non-uniform growth and closed venation would significantly increase the types of leaves that could be created. As well as improvements and speed ups in the nearest neighbour search to speed up the process. The algorithm would be much improved by the addition of parameters to control, to some extent, the straightness and symmetry of the veins, perhaps by using con strained auxin placement.

Additionally on the texturing side, recreating the colouration to use more parameters would allow the user to get better results .It is also possible to infer colour from the position of the leaves once they are on the tree model, which would greatly increase realism. Enhancements such as specular maps would also greatly increase the realism.

For mesh generation, attaching the leaf textures to a more detailed mesh will allow for the formation of macro-details and allow for the deformation of the leaf based on environmental factors.

Lastly, under the foliage creation, a system that could dynamically and interactively replace, move, or distort leaves once they are on the tree would help shape the foliage better, something that is lacking in this system and Tree Draw as a whole.

7. REFERENCES

Anastacio, F., Prusinkiewicz, P., & Sousa, M. C. (2008). Sketch-based parameterization of l-systems using illustration inspired construction lines. *In Proceedings of 5th eurographics workshop on sketch-based interfaces and modeling.* SBIM'08.

Bloomenthal, J. (1985, July). Modeling the mighty maple. *In ACM SIGGRAPH Computer Graphics*, pp. 305-311.

Boudon, F., Prusinkiewicz, P., Federl, P., Godin, C., & Karwowski, R. (2003, November). Interactive design of bonsai tree models. *In Computer Graphics Forum*, pp. 591-599.

Chen, X., Neubert, B., Xu, Y. Q., Deussen, O., & Kang, S. B. (2008, December). Sketch-based tree modeling using Markov random field. *In ACM Transactions on Graphics (TOG)*, p. 109.

Chiba, N., Ohshida, K., Muraoka, K., & Saito, N. (1996). Visual simulation of leaf arrangement and autumn colours. *The Journal of Visualization and Computer Animation*, 79-93.

Cook, M. T., & Agah, A. (2009). A survey of sketch-based 3-D modeling techniques. *Interacting with Computers*, 201-211.

de Reffye, P., Edelin, C., Françon, J., Jaeger, M., & Puech, C. (1988). Plant models faithful to botanical structure and development. *In ACM SIGGRAPH Computer Graphics* (pp. 151-158). ACM.

Deussen, O., & Lintermann, B. (1997, May). A modelling method and user interface for creating plants. *In Graphics interface*, pp. 189-197.

Deussen, O., Hanrahan, P., Lintermann, B., Mech, R., Pharr, M., & Prusinkiewicz, P. (1998). Realistic modeling and rendering of plant ecosystems. *Realistic modeling and rendering of plant ecosystems. In Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (pp. 275-286). ACM.

Dorsey, J., & Rushmeier, H. (2009, August). Advanced material appearance modeling. *In ACM SIGGRAPH 2009 Courses*, p. 3.

Ford, A., & Roberts, A. (1998). *Colour space conversions.* London: Westminster University.

Gingold, Y., Igarashi, T., & Zorin, D. (2009, December). Structured annotations for 2D-to-3D modeling. *In ACM Transactions on Graphics (TOG)*, p. 148.

Godin, C., & Caraglio, Y. (1998). A multiscale model of plant topological structures. *Journal of theoretical biology*, 1-46.

Ijiri, T., Owada, S., & Igarashi, T. (2006). The Sketch L-System: Global Control of Tree Modeling. *In Smart Graphics*, pp. 138-146.

Mech, R., & Prusinkiewicz, P. (1996). Visual models of plants interacting with their environment. *In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (pp. 397-410). ACM.

Mundermann, L., MacMurchy, P., Pivovarov, J., & Prusinkiewicz, P. (2003, July). Modeling lobed leaves. *Computer Graphics International* (pp. 60-65). IEEE.

Neubert, B., Franken, T., & Deussen, O. (2007, August). Approximate image-based tree-modeling using particle flows. *In ACM Transactions on Graphics (TOG)*, p. 88.

Okabe, M., Owada, S., & Igarash, T. (2005, September). Interactive Design of Botanical Trees using Freehand Sketches and Example-based Editing. *In Computer Graphics Forum*, pp. 487-496.

Olsen, L., Samavati, F. F., Sousa, M. C., & Jorge, J. (2008). *A taxonomy of modeling techniques using sketch-based interfaces.* Eurographics State of the Art Reports.

Prusinkiewicz, P., & Lindenmayer, A. (1991). *The algorithmic beauty of plants.* The Virtual Laboratory.

Prusinkiewicz, P., Hammel, M., Hanan, J., & Mech, R. (1996, February). L-systems: from the theory to visual models of plants. *In Proceedings of the 2nd CSIRO Symposium on Computational Challenges in Life Sciences*, (pp. 1-32).

Prusinkiewicz, P., James, M., & Mech, R. (1994). Synthetic topiary. *In Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (pp. 351-358). ACM.

Prusinkiewicz, P., Mündermann, L., Karwowski, R., & Lane, B. (2001). The use of positional information in the modeling of plants. *In Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (pp. 289-300). ACM.

Quan, L., Tan, P., Zeng, G., Yuan, L., Wang, J., & Kang, S. B. (2006, July). Image-based plant modeling. *In ACM Transactions on Graphics (TOG)*, pp. Vol. 25, No. 3, pp. 599-604.

Reche-Martinez, A., Martin, I., & Drettakis, G. (2004, August). Volumetric reconstruction and interactive rendering of trees from photographs. *In ACM Transactions on Graphics (TOG)*, pp. Vol. 23, No. 3, pp. 720-727.

Rodkaew, Y., Chongstitvatana, P., Siripant, S., & Lursinsap, C. (2003). Particle systems for plant modeling. *Plant Growth Modeling and Applications*, 210-217.

Runions, A., Fuhrer, M., Lane, B., Federl, P., Rolland-Lagan, A. G., & Prusinkiewicz, P. (2005). Modeling and visualization of leaf venation patterns. *In ACM Transactions on Graphics (TOG)* (pp. Vol. 24, No. 3, pp. 702-711). ACM.

Sanger, J. E. (1971). Quantitative investigations of leaf pigments from their inception in buds through autumn coloration to decomposition in falling leaves. *Ecology*, 1075-1089.

Smolenová, K., & Hemmerling, R. (2008). Growing virtual plants for virtual worlds. *In Proceedings of the 24th Spring Conference on Computer Graphics* (pp. 67-74). ACM.

Wang, L., Wang, W., Dorsey, J., Yang, X., Guo, B., & Shum, H. Y. (2006). Real-time rendering of plant leaves. *In ACM SIGGRAPH 2006 Courses*, 5.

Wang, X., Zhao, C., Lu, S., & Guo, X. (2009). Survey on Modeling and Visualization of Plant Leaf Color. *Third International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications (PMA)* (pp. 417-424). IEEE.

Weber, J., & Penn, J. (1995). Creation and rendering of realistic trees. *In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (pp. 119-128). ACM.

Zakaria, M., & Shukri, S. (2007). A sketch-and-spray interface for modeling trees. *In Smart Graphics*, pp. 23-35.

APPENDIX

A. USER INTERFACE

A.1. SKETCH INTERFACE

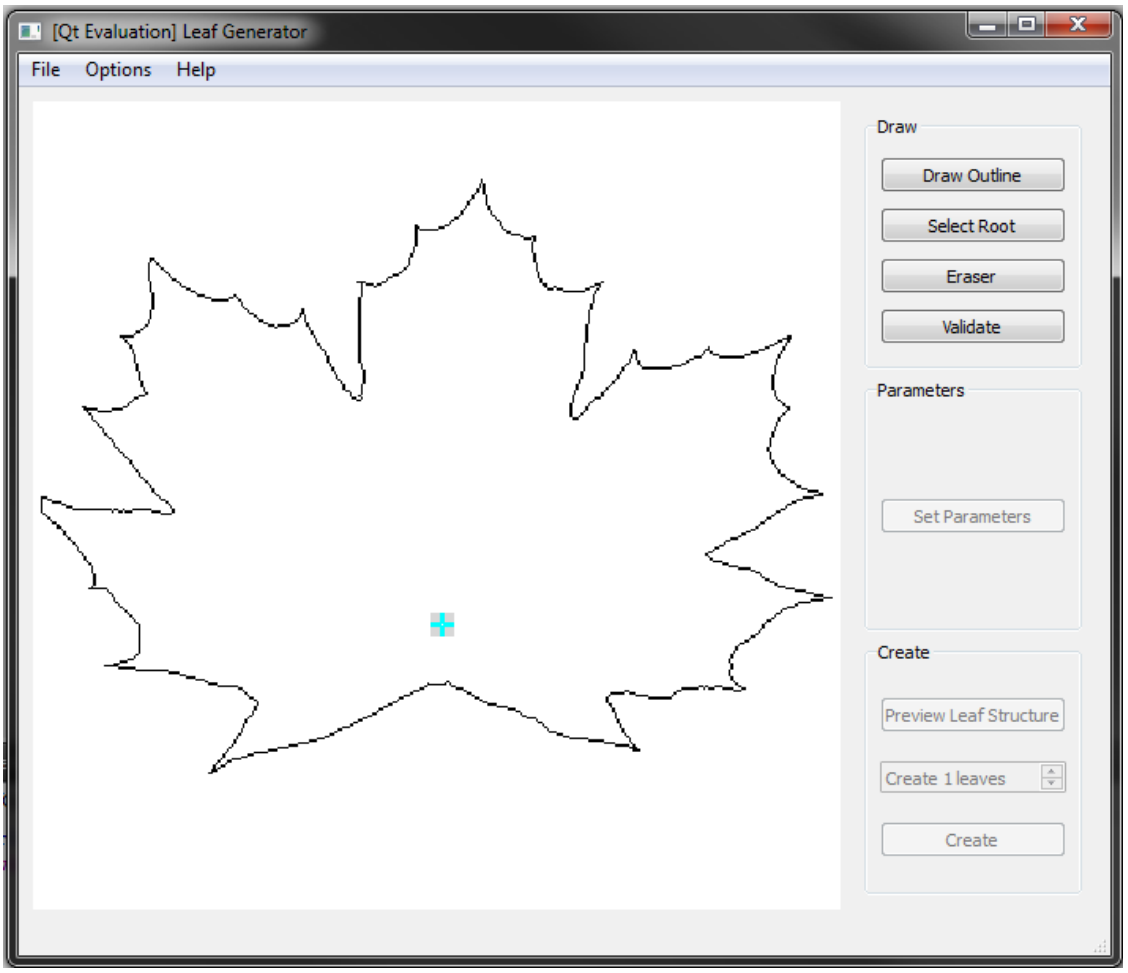


Figure 18: Sketch Interface

This is the sketch interface at the first stage of sketching, before validation, After validation, the parameters box becomes available, and after setting the parameters, the create box can be interacted with. This is intended to make sure the user has a well-formatted model before the generation starts.

In the file menu, the user can create a new leaf or open one from a file.

A.2. PARAMETER SELECTION INTERFACE

GROWTH PARAMETER TAB

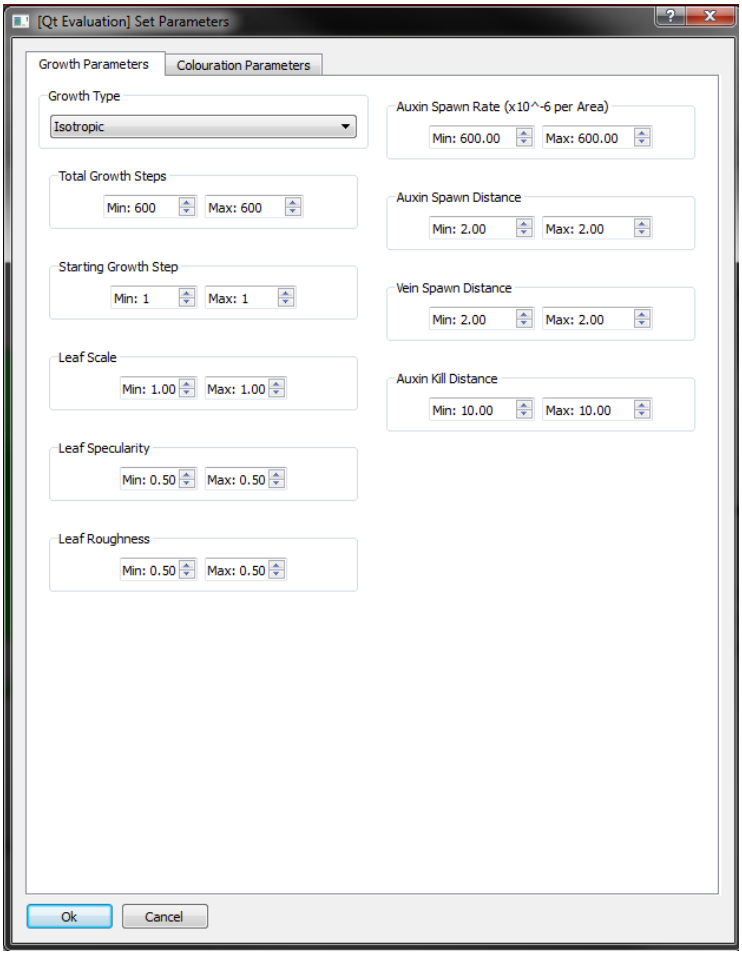


Figure 19: Growth Parameter Panel

This panel allows the user to set the grow parameters of the leaf. Each variable is given a maximum and minimum value, and the final value sent through to generation is taken randomly from between these two. The form is designed so that each parameter can only be set between two values where the parameter is well defined. Additionally the form checks and ensures that the minimum is below the maximum.

COLOUR SELECTION TAB

Figure 20: Colour Parameter Panel

This panel allow the user to pick the colour type and variation. For each texture, the main colour is shown, with its potential variation between the two colours. Underneath is the colour relating to the colour type, which is off for uniform. This colour also has a range and a potential colour range below it. The spread distance determines how far the effect spreads from its source.

To the right of a parameter is the preview blending of the effect for the middle potential colour value of the parameters.

A.3. FOLIAGE CREATION INTERFACE

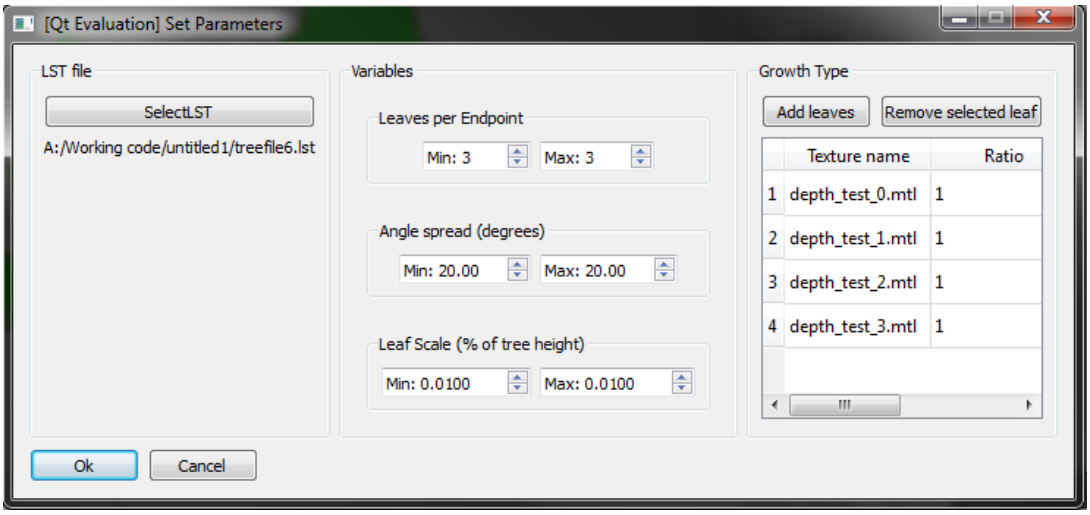


Figure 21: Foliage Creation Interface

This is the foliage creation interface. It allows the user to specify a tree to build, as well as how many leaves per end point, how far they spread from each other, and the size (as a percent of the tree).

On the right, the user selects the leaf materials he wishes to use and the ratio that they should appear on the tree. The percentage chance of a being used on a tree is its ratio/sum(all ratios)

B. Explanation of Parameters

B.1. Texture Parameters

The parameters used for texture creation can be split into two categories, listed below. To facilitate variability between leaves in the same creation run, the parameters (excluding growth and colour type) are given with a maximum and minimum, from which the final value is taken randomly for each new leaf. These parameters are used to define the structure and shape of a leaf, when provided along with the outline of the leaf retrieved from the sketch and a point representing the root of the leaf, the point at which the veins start their growth. All points that define the outline are stored as vectors from the root point, which allows the algorithm to grow the leaf by moving the outline out from the root node along the vectors that define the outline.

B.2. Growth and Venation Parameters

These parameters control the way the leaf grows from its starting state to the final leaf size. This affects the way the veins on the leaf are formed, and thus the parameters in this category are the ones that directly affect the venation process.

Growth Type

This parameter defines how the area inside the leaf changes per growth step. The four options are *none*, *marginal, isotropic*, and *anisotropic.*

For the *none* growth type, the leaf outline remains fixed at every step at its final size, the size of the original sketch.

Marginal growth increases the area of the leaf at each step by increasing the area at the borders of the leaf. This results in the outline of the leaf being moved outward at each step without moving any of the vein nodes or auxin sources already placed in the leaf.

For *isotropic* growth, the area of the leaf is increased every step by the same factor in both x- and y- coordinates. However, unlike *marginal* growth, the contents of the leaf that are already in place are scaled along with the leaf, causing the vein nodes and auxin sources to expand out from the root along with the leaf outline.

Anisotropic growth is similar to *isotropic* growth, except that the x- and y- growth rates are decoupled, allowing for different growth rates along either axis.

Total, Horizontal and Vertical Growth

This parameter sets the number of steps required to grow from the starting state to the final inputted sketch. Larger values for the growth rate correspond with smaller changes in the area of the leaf per step, which in turn tends to create veins that are closer together, as well

as veins that branch more often, as branching happens on a per-step basis. Conversely, smaller values for growth create a less packed leaf, with less tertiary veins.

When the growth type is *marginal* or *none*, there runs the danger of veins not reaching the end of the leaf if growth is too small. The growth rate is a singular value for *Marginal* and *Isotropic* growth, and is therefore represented by Total Growth. The distance travelled by each point in the outline per step is calculated as a function of the starting and ending coordinates, as well as the number of steps. Specifically, for a growth rate, starting point at step 0 p_0, and end point at step G p_G, and *r*, the root node, the position at step *t* can be found using the following formula:

$$p_t = p_0 + \frac{p_G - p_0}{G} t$$

where

$$p_0 = \frac{p_G}{G} + r$$

Anisotropic growth requires the growth rates for both *x*- and *y*-coordinates, and thus has two inputs, Horizontal and Vertical Growth, G_h and G_v. This means that the number of steps taken to grow horizontally differs from the amount of steps vertically. The solution suggested in (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005) works well for an input of the starting outline, but does not help with this system's format.

The solution is to take G_h and G_v, and find the highest. This is then made the total amount of steps G_t. The difference between the highest and the lowest is the number of steps the other growth rate does not complete. Since the end result of this growth is defined, it must be that the lower growth steps started at (highest-lowest) steps closer to their goal.

If $G_h < G_v$, then $G_t = G_v$,and the starting point p_0 for each vector *v* in the outline is calculated as a function of the end value $v + r$, growth steps G_h , G_v and $delta_h$, $delta_v$, the components of $\frac{v}{G_t}$ (distance on axis per total steps) below:

$$p_0(x, y) = (\frac{v_x}{G_t} + r_x + delta_h(G_v - G_h), \frac{v_y}{G_t} + r_y)$$

Otherwise, If $G_v < G_h$, $G_t = G_h$ and

$$p_0(x, y) = (\frac{v_x}{G_t} + r_x, \frac{v_y}{G_t} + r_y + delta_v(G_h - G_v))$$

and therefore p at step *t* is

$$p_t(x, y) = (p_{0x} + t(delta_h),\ p_{0y} + t(delta_v))$$

This results in an initial shape that is deformed in such a way that when it is grown with the correct respective x- and y- growth rates, the original sketched outline is the result of the final step.

The growth type *none* defines a leaf with no change in area over the total number of steps, and as such every point in the outline is constant for every step:

$$p_0 = p_G = p_t$$

STARTING GROWTH STEP

This parameter controls the step that algorithm starts on. For values close to the total growth, the leaf veins tend to be long and straight, growing steadily until they reach the initial size of the leaf before branching out. Conversely, smaller values will create less straight venation.

SCALE

Scale controls what proportion of the steps are completed before the algorithm stops. The scale is measured between 0 and 1, with 0 representing no steps, while 1 represents all of the steps being completed. If scale is set to less than 1, this represents a leaf that is part-way through its growth, and is useful for representing budding leaves. It can also be used to ensure a size variation in the leaves produced, however if scale is too small, the end result can be leaves with veins that do not sufficiently span the leaf, much like if starting growth step is set too high, the algorithm lacks to the steps to create the correct structure.

AUXIN SPAWN RATE

This parameter controls the number of auxin points spawned per step. For *isotropic* and *anisotropic* growth, the parameter given is converted into n_a, the number of auxin to spawn at a particular step, by taking the value *x* provided, along with the total growth steps *G,* and the leaf area at step *t*, $Area_t$ as follows:

$$n_a = x \frac{Area_t}{G(10^6)}$$

This equation is taken from (Runions, Fuhrer, Lane, Federl, Rolland-Lagan, & Prusinkiewicz, 2005), and is an approximation of the equations set out by (Judd, Campbell, Kellogg, Stevens, & Donoghue, 1999) in their paper on leaf phylotaxia.

When the leaf is set to the *marginal* or *none* growth type, all of the auxin are placed before the first step starts, to fill the area of the leaf at the final step. This is done for efficiency reasons, as *marginal* growth requires auxin to be placed only in the area gained by the growth in the previous step, which is expensive to calculate. If the auxin are already placed, it costs far less to simply include them as they enter the expanding outline. For *none*, the leaf does not grow, so it is required that auxin are placed beforehand.

High values for spawn rate will cause larger amounts of branching in the veins as the veins find more targets to grow toward. This parameter also has the greatest effect on performance, as high values can often spawn thousands of auxin per step, drastically increasing computation time, without necessarily increasing realism.

AUXIN SPAWN DISTANCE

This value is the minimum straight-line distance that any auxin can spawn from any other auxin. High values cause less auxin to spawn, and also spread out the auxin, resulting in thinner, longer venation.

VEIN SPAWN DISTANCE

This value is the minimum straight-line distance that any auxin can spawn from any vein. High values cause auxin to spawn toward the edge of the edge of the leaf area, creating vein with less branching, as it is more likely that auxin will share a closest vein node when they toward the edges, attracting nodes near the ends of the veins.

AUXIN KILL DISTANCE

This value is the maximum straight-line distance an auxin source can exist from a vein. Large values cause shorter vein branches, as veins kill auxin before they can grow further toward the auxin source.

B.3. COLOURATION AND TEXTURING PARAMETERS

These parameters the colour and light interaction properties of the leaf. Once the basic vein structure has been created, the colouration can then occur. The colour spectrum used in this paper is stored and calculated as HSV (Hue, Saturation, Value), rather than traditional RGB (Red, Green, Blue) because this better suits the gamut and interactions between leaf colours, primarily the presence and decay of chlorophyll (green), xanthophyll (yellow), carotene (orange) and anthocyanin (red or purple). See section 3.3 for more details on their interactions.

The colour parameters are specified with variation, similarly to the integer and real number parameters. The method used to specify this variation is explained along with the rest of the user interface in the appendix A.2.

COLOUR TYPE

This parameter determines which colouration algorithm will be used. The choices listed are based on several common colouration schemes found in tree and bush leaves (Sanger, 1971). The options are *uniform, colour from veins, colour from edges,* and *speckled.*

All options, excepting *uniform*, take in three parameters. These are base colour, the overall colour of the leaf; alternate colour, the colour that applies specifically to the scheme; and intensity, which is the radius of effect for a scheme.

Uniform colour specification uses only one colour parameter, the base leaf colour. This colour is spread over the entire area of the leaf before the veins are drawn. This colour pattern imitates many of the common leaves in nature, where there are no significant colour variations over a single leaf. (Wang, Zhao, Lu, & Guo, 2009)

Colour from veins takes two colour parameters, the base colour, and the alternate colour. The base colour is applied over the entire leaf, after which the alternate colour is spread out from the veins. Each vein node is radiates out colour from itself out to a maximum radius of intensity*vein width, the colour at the points surrounding the vein node are altered from

their current colour , which may differ from the base colour due to previous nodes, as a function of their distance from the vein node as follows:

$$colour(h,s,v) = \frac{dist}{max_{radius}} colour_{base} + (1 - \frac{dist}{max_{radius}}) colour_{alternate}$$

This creates spreading colour that is stronger at larger veins and smaller at the tertiary veins. This colouration is congruent to several types of leaves, some with natural colour differentiation close to the root, and others in the process of autumnal decay (Chiba, Ohshida, Muraoka, & Saito, 1996). During decomposition, several leaf types, including those of the oak and maple, release acids contained in the veins into the leaf surface. This action kills the green chlorophyll present, resulting in a change of colour outward from the veins as it spreads.

Colour from edges applies the base colour over the entire leaf and then radiates colour out from each of the edge nodes into the leaf with a maximum radius defined by the intensity. The equation for the alteration of colour at any point within the maximum radius is similar to that of *colour from veins,* with the maximum radius set to the intensity value. *Colour from edges* is used to model both advanced stages of decomposition, when only the edges still contain chlorophyll, as well as instances where leaves have suffered damage, either through excessive rain, which causes rot and discolouration to the outside of a leaf, or frost damage. There are also leaves which naturally contain this colouration.

Speckled is the last option. It applies the base colour to the leaf and then places a number of points into the leaf using a dart-throwing algorithm (the same used for auxin placement). The number of points placed is dependent on the auxin spawn rate value divided by 10. The alternate colour is then spread out from each point to a maximum radius set to the intensity value. The equation for alteration of colour with this radius is the same as that of *colour from edges.*

Any option not chosen is still implemented, however it is done using a colour just offset from the base colour. This gives helps prevent the leaf from looking one-dimensional, and follows from the work done in (Rodkaew, Chongstitvatana, Siripant, & Lursinsap, 2003), with noise to simulate small veins not present in the vein model.

FRONT AND BACK TEXTURE COLOUR

These parameter sets control the front and back colour parameters for the leaf, as used in the colouration types above. It is typical for leaves to be a lighter shade underneath than on top, due to the chlorophyll build up near where the sun hits the leaf. (Judd, Campbell, Kellogg, Stevens, & Donoghue, 1999) However, this is not universal, so the user has the ability to set the top and bottom colour variations independently. The colouration patterns are kept the same for both top and bottom, as veins and colouration elements run through the entire leaf and affect both top and bottom in a similar manner, if not to the same magnitude.

There is a danger of nonsensical leaves being produced, should the user specify too much variation for each texture. This can result in colours from the front and back not matching

up. This is a small risk however, as leaves for a single tree tend to exhibit very small variations in colour between leaves. Should the user desire vastly differing colours, it is advisable to create each in a different sequence, and join the leaves together again during the foliage step. This is trivial to achieve using the same variables.

LEAF SPECULARITY

This leaf controls how shiny the leaf is on a scale of 0 to 1, where 0 represents a completely matte leaf, and 1 represents a completely shiny leaf (akin to a very clean car). Note that this shininess is different to reflectiveness, as the leaf produces a white highlight as a result of direct lighting, rather than reflecting any image.

This value imitates the build-up of waxes and oils on the leaf surface, which then deflect and absorb UV rays, protecting the leaf (Weber & Penn, 1995). This value is given through to the final leaf material to specify the leaf's reaction to light when placed into a model. The value is taken to a third when applied to the underside of the leaf, to account for the lessened amount of oils on the side away from the sun. This value is a simple heuristic taken from the average of the values given in (Judd, Campbell, Kellogg, Stevens, & Donoghue, 1999).

LEAF ROUGHNESS

Leaf roughness controls the scale of the bump map that corresponds to the minor geometric detail that is not included in the leaf mesh constructed later. This scale is from 0 to 1, where 0 represents a perfectly smooth leaf, and 1 represents a very rough one. Even at a scale of 1, the leaf is not overly rough, this is because any roughness above this point should be modelled macroscopically in the mesh. The included height detail, such as damage or the effect of veins is baked (written) onto a special texture known as a height map. This height map contains a value for each picture in the texture representing the height of the pixel at this point. When loaded into modelling software, this height map is called a bump map and informs the software to create shadows on the texture as if there were ridges and depressions on the leaf.

The height maps are written into the red channel of a new texture, and one exists for each texture on the leaf, top and bottom.

The vein height is calculated using a heuristic assumption that leaves contain topologic noise (they are not perfectly flat), and that the veins are slightly indented into the leaf at the top, and come out above the leaf at the bottom, based on evidence from (Judd, Campbell, Kellogg, Stevens, & Donoghue, 1999).

C. SAMPLE FILES

C.1. SAMPLE .MTL FILE

```
newmtl leaf_4_top
Ka 0.0435 0.0435 0.0435
Kd 0.6
Ks 0.5
illum 2
Ns 125

map_Ka -clamp on -s 1 1 1 -o 0 0 0 -mm 0 1 leaf_4_top.png
map_Kd -clamp on -s 1 1 1 -o 0 0 0 -mm 0 1 leaf_4_top.png
```

```
bump -clamp on -imfchan r -s 1 1 1 -o 0 0 0 -bm 1 leaf_4_topN.png

newmtl leaf_4_bottom
Ka 0.0435 0.0435 0.0435
Kd 0.6
Ks 0.25
illum 2
Ns 125

map_Ka -clamp on -s 1 1 1 -o 0 0 0 -mm 0 1 leaf_4_bottom.png
map_Kd -clamp on -s 1 1 1 -o 0 0 0 -mm 0 1 leaf_4_bottom.png
bump -clamp on -imfchan r -s 1 1 1 -o 0 0 0 -bm 1 leaf_4_bottomN.png
```

This file shows the material file. There are two materials, top and bottom. Each has the diffuse and ambient texture map set to the leaf texture, and the bump map set to the leaf's bump map. The specular values (Ns) are set to the specular parameter.