Practical Modelling of Realistic Plants for a Real-Time 3D Environment

by

Graeme Tankard

November 2001

in partial fulfilment of a Bachelor of Science (Honours) degree at Rhodes University

Abtract

This thesis introduces three key aspects of a plant modelling system for use in a three-dimensional real-time environment: practicality of modelling, realism of models produced, and the rendering speeds obtained by these models. Four systems are implemented to demonstrate these key aspects, each system focusing on one aspect whilst trying to maintain acceptable levels of the other two. It is found that only one of the implemented systems is able to attend to all three aspects, while the other three fail to attend significantly to either of the two criteria outside their focus.

Acknowledgements

I would like to thank Shaun Bangay for his valuable input to this project, for never giving the game away, but for constantly prodding me in the right direction. I would also like to thank my girlfriend, Keryn, and my father for their continuous support throughout the year. Lastly, I would like to thank the members of VRSIG for always pretending to be interested in what I had to say.

Table of contents

1 Introduction

The modelling of plants is traditionally complex, and the high number of polygons produced often makes them unsuitable for real-time environments. The aim of this project is to design, implement and compare a set of modelling systems taking into account practicality of modelling, realism of plants produced and rendering speeds attained.

Plants are so common in our environment, whether we are indoors, outdoors or even underwater, that we often take little to no notice of them. It is only when we are presented with an environment without plants that we realise just how naked such an environment appears. In our quest for a realistic virtual environment, therefore, it stands to reason that virtual plants are essential.

Almost since their inception, vehicle simulations and computer games have used plants extensively to improve the realism of their scenes. As computers and their graphical capabilities have become better and faster, so the plant models in such applications have become more detailed and realistic – from single-polygon texture-mapped billboards to intricate meshes containing hundreds, even thousands of polygons.

Naturally, plant modelling also has an application in botany. Using L-Systems [PruHam 95], botanists are able to simulate the growth and development of plants, and are subsequently better able to understand the processes behind some of the complex plant structures found in nature. Detailed aspects of plant development such as the amount of light received and the concentration of nutrients in the soil can be modelled in great detail. Even more abstract phenomenon such as topiary (the art of pruning bushes and trees into shapes) can be accurately and easily modelled.

The modelling and rendering of plants is seldom a trivial task. In the design of a plant modelling system, several choices must necessarily be made as to whether the system focuses on practicality of modelling, realism of the resultant model, or the speed at which it can be rendered. It is rare that a system is able to excel at all three criteria. A high degree of realism, by nature, increases the complexity of the model, thus reducing the speed at which is can be rendered. It also generally imposes restrictions on the practicality of modelling, as realistic looking plants are naturally complex and difficult to describe in an intuitive manner. This is shown clearly in L-Systems where the modeller requires substantial experience in order to model realistic plants.

In this thesis, we begin with a discussion of the requirements for a practical plant modelling system. This is followed by the design and implementation of three modelling systems: an elementary L-System modeller, an interactive modelling system based on ramification matrices, and an object oriented based component modelling system. We compare these systems in terms of the ease of modelling, the realism of plants produced, and the speed of rendering.

2 Related Work

The modelling and rendering of plants is not a new field of research and has received considerable attention in recent years. Some of the research related to this thesis is mentioned in this chapter. The research that relates *directly* to the systems developed in this thesis will be introduced in somewhat more detail.

2.1 Lindenmayer Systems (L-Systems)

Introduced by a biologist, Aristid Lindenmayer, as a theoretical model of plant development, L-Systems have become a powerful and widely used tool for the creation of botanically accurate plant models. Recent research has been conducted into the interaction of L-Systems with their environment [MecPru 96], allowing elements such as sunlight and gravity to have an effect on the development of the plant and even allowing for the creation of topiary [Internet cpsc]. L-Systems are introduced in somewhat more detail here, as part of this project involves the design and implementation of an elementary L-System parser.

2.1.1 Definition of terms

It is important to clarify some of the terminology used in L-Systems so as to avoid misunderstanding.

2.1.1.1 Modules

A module, in the context of L-Systems, is defined as “any discrete constructional unit that is repeated as the plant develops” [PruHam 95]. In this thesis, a module is considered to be the encapsulation of a *symbol*, representing the name of the module, and a set of *numerical attributes*. The purpose of these attributes is discussed below. A string of modules representing a plant is referred to as an L-System string.

2.1.1.2 Productions

A production specifies the way in which modules in an L-System string are replaced. In a context-free system, a production consists of a *predecessor* defined by a single module, and a successor defined by a set of zero or more modules.

2.1.2 Parallel rewriting systems

At the heart of L-Systems is the idea of parallel rewriting systems. An L-System is defined by a starting string and a list of productions. Each module in the L-System string is compared with the predecessor module of each production. If a match is found, the module is replaced with the successor modules defined in the production. An example of this is given below.

$$\begin{aligned} \omega &: \text{BAB} \\ p_1 &: \text{A} \rightarrow \text{C} \\ p_2 &: \text{B} \rightarrow \text{D} \end{aligned}$$

In the above L-System, ω is the starting string, p_1 and p_2 are the productions in which A and B are the predecessors and C and D are the successors. In the first iteration, the module named A in the starting string would be replaced with the module named C, and the module named B would be replaced with the module named D. The resulting L-System string would therefore be: DCD. This type of rewriting process is shown graphically in Figure 1 from [Internet cpsc].

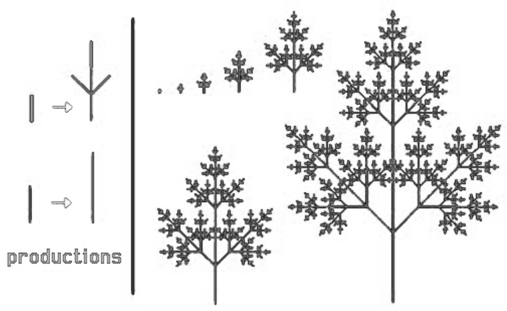


Figure 1 – A graphical rewriting of an L-System string

2.1.3 Parametric L-Systems

Parametric L-Systems make use of the numerical attributes of modules in the selection of an appropriate production. Productions are modified to include numerical conditions that must be satisfied by the module attributes in order for the production to be applied. This allows for the declaration of multiple productions with the same predecessor. These productions are distinguished only by differing numerical conditions. An example of a parametric L-System is given below.

$$\begin{array}{lllll}
\omega : & B(1)A(2,3)B(4) & & & \\
p_1 : & A(x, y) : & y > 2 & \rightarrow & C(1,2) \\
p_2 : & A(x, y) : & y <= 2 & \rightarrow & B(5)A(x-1,0) \\
p_3 : & B(x) \quad : & x < 1 & \rightarrow & B(x+1) \\
p_4 : & B(x) \quad : & x >= 1 & \rightarrow & D(x-1)
\end{array}$$

The first module of the initial string, B(1), has a symbol, B, which can be matched to either p_3 or p_4. The numerical condition of p_3, however, requires that the module attribute be less than 1, which in this case is not so. The only production whose numerical condition is satisfied by the attribute of the module B(1) is p_4. The L-System string, after the first iteration, will therefore be: D(0)C(1,2)D(3).

2.1.4 Interpretation of L-System generated strings

Once an L-System string has been generated, the corresponding geometry needs to be created. One popular method of accomplishing this, as described by Prusinkiewicz et al [PruHam 95], is to use a LOGO-style turtle. Each module in the L-System string is treated as a command to the turtle. Thus a module such as F(5) could be interpreted as “move the turtle forward by 5 units” and /(60) could mean “rotate the turtle by 60° in a counter-clockwise motion about the z-axis”.

2.1.4.1 Branching structures

Branching structures are important in the modelling of plants, and can be easily implemented through the creation of a stack that holds the turtle state (position and orientation). Two modules, "[" and "]", are then defined to access the stack. The interpretation of the "[" module is to push the turtle state onto a stack, while the "]" module is used to pop the turtle state from the stack. A sample branching L-System is shown below, and is graphically represented in Figure 2. The turtle position, at each stage in the parsing process, is shown as an open circle.

F(2)[+(45)F(1)][-(45)F(1)]

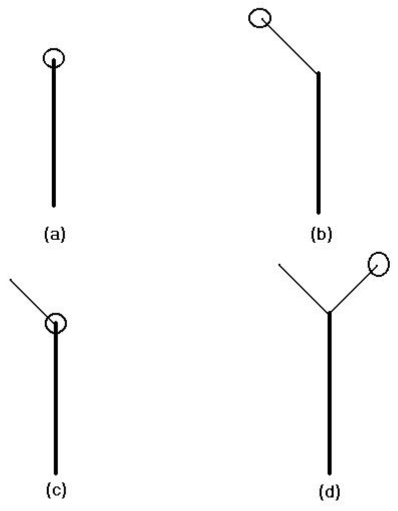


Figure 2 – The turtle-based traversal of an L-System string

2.1.5 Advantages and disadvantages of L-Systems

The primary advantage of L-Systems is their ability to model complex plant structures through the definition of a few key productions. These productions, however, often take many hours of trial and error to produce, even by experienced L-System modellers.

Due to its rewriting nature, the system must be “re-grown” every time a change to a production is made. This can make modelling L-Systems a tedious task, and the design of a real-time, interactive L-System modeller nearly impossible.

In general, the fractal nature of L-Systems causes the complexity of the generated string to increase exponentially with every rewriting-iteration. After only a few iterations, the generated string can run to many pages, and the geometry produced can bring even a powerful system to its knees.

2.2 Ramification Matrices

Introduced by Viennot *et al* [Viennot 89], this technique provides a purely statistical approach to modelling binary trees. We begin this section with a look at how ramification matrices are compiled.

2.2.1 Compiling a matrix

The following description follows closely that of Viennot, and demonstrates the relationship between the ramification matrix and the plant structure. We begin with a sample structure shown in Figure 3.

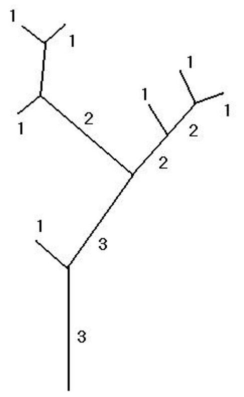


Figure 3 – An tree structure with ordered branches

The edges (branches) of the tree are labelled in the following recursive manner:

- The terminal branches are labelled 1
- If a branch has two children labelled i and j then the label of the branch is
  - the larger of i and j if $i \neq j$
  - i+1 if i = j

The label of a branch is called the *order* of the branch, and is proportional to the size of the tree contained within its children. In the calculation of the actual values of the ramification matrix, we define the following quantities (from Viennot):

for $k \geq 2$, a_k is the number of order k branches

for $1 \leq i < k$, $b_{k,i}$ is the number of branches whose children have order k and i

for $k \geq 2$, $b_{k,k}$ is the number of branches whose children have order k-1 and k-1

for $1 \leq j \leq k$, $p_{k,j}$ is the probability of a branch have children of order k and j and is equal to $\frac{b_{k,j}}{a_k}$

Thus, the ramification matrix corresponding to Figure 3 with k rows and j columns would be:

$$\begin{bmatrix} 1 & & \\ 3/5 & 2/5 & \\ 1/2 & 0 & 1/2 \end{bmatrix}$$

where $p_{1,1}$ has just been including for the sake of completing the form of the matrix. As can be seen from this matrix, the probability of a branch of order 2 (k=2) having children of order 2 and 1 (j=1) is 60%, while the probability of having children of order 1 and 1 (j=2) is 40%.

2.2.2 Advantages and disadvantages

Ramification matrices, due to their statistical nature, are able to store a tree-type rather than an actual tree model. A single matrix, consisting of only a handful of numbers, can describe a forest of similar trees, few of which are identical.

A major drawback to this method is the generation of the matrix itself. A ramification matrix can be compiled from an existing tree structure (as explained above), and while this can be useful in replicating real tree structures, few modellers have the inclination to count the branches in a tree and calculate branching probabilities. Quite a number of generic, pre-calculated, ramification matrices exist (such as binary and increasing-binary methods described by Viennot), but the fact remains that it is difficult to construct a ramification matrix from nothing.

Another drawback is the inability of ramification matrices to produce non-binary trees. At every branch point there is a split into exactly two branches. While this is suitable for the modelling of trees, it proves unsuitable for the modelling of small plants or flowers, as they often have more than one branch per node.

2.3 Component Modelling

This modelling method, introduced by Bernd Lintermann and Oliver Deussen [LinDeu 98], aims at separating the plant structure from the geometrical properties of the plant. “Both tasks require their own optimised modelling techniques” [LinDeu 98]. They introduce the idea of components: objects that encapsulate both structural and geometrical data, as well as optimised algorithms for the manipulation and representation of the data. These components are then linked together as nodes in a directed graph that describes the overall structure of the plant.

Lintermann and Deussen describe three types of component: components for generating geometry, components for the iteration and arrangement of other components, and components that implement constraints and geometrical transformations. These components are briefly introduced below (the descriptions closely follow those of Lindermann and Deussen).

Geometry-generating components include:

- ***Simple***
    - Produces simple geometry such as cubes and spheres
- ***Revo***
    - Produces a surface of revolution
- ***Horn***
    - Used to generate stem-type geometry
- ***Leaf***
    - Produces leaf geometry

Iterating components include:

- ***Tree***
    - Distributes subsequent components as branches about a stem
- ***Hydra***
    - Distributes subsequent components in a radial fashion, perpendicular to its parent component
- ***Wreath***
    - Similar to the hydra except that subsequent components are distributed parallel to its parent component
- ***PhiBall***
    - Distributes subsequent components on the section of a sphere

Geometrical transformation and constraint components include:

- ***World***
    - Allows environmental elements such as light and gravity to affect subsequent components

- ***Pruning***
  - Constrains subsequent geometry through a pruning algorithm
- ***Free-form-deformation***
  - Allows manipulation of the overall plant structure by means of a 2D grid of control points
- ***HyperPatch***
  - Similar to free-form-deformation, except that a 3D grid of control points is used

2.4 Other modelling methods

Prusenkiewicz *et al* introduce the "use of positional information in the modelling of plants" [PruMün 01]. Based on L-Systems, this technique aims at the integration of artistic elements such as posture, arrangement of components and plant silhouette into plant models.

Jirasek *et al* [JirPru] introduce the idea of rotational springs in the modelling of the effects of gravity on plants. Each branch is divided into a number of segments connected by rotational springs. Each segment has length, width and mass properties, and each rotational spring is assigned an individual spring constant. Hanging plants are modelled particularly effectively with this method.

Slicing and blending [Jakulin 00] is an interesting extension to billboarding, using alpha-blended textured polygons to achieve the effect of parallax in trees. A tree model consists of a number of sets of "slices" (alpha-blended, textured polygons) which are arranged one behind the other to give the impression of a true 3D model. As the view is rotated, the old set is blended out and the new set blended in. This gives the appearance of a 3D model while maintaining the speed of rendering achieved with billboarding.

2.5 Summary

In this section, we have introduced some of the currently researched techniques of plant modelling. Most notable of these are the component modelling technique, L-Systems, and ramification matrices that will be explored further in the course of this project. It was understood from the literature survey that plant modelling is an intensely researched field, and many attempts are being made to make modelling more practical and the models produced more realistic, while attempting to maintain a low complexity.

3 Design

In this chapter we discuss the design of a number of plant modelling systems. Our goal is to demonstrate various levels of practicality, realism and rendering speeds within the various modelling systems. We begin with an analysis of these requirements, followed by the design of four plant-modelling systems, each demonstrating very different aspects of plant modelling. Each system focuses on one of the three requirements mentioned above, while trying to maintain acceptable levels of the other two.

3.1 Requirements analysis

The requirements for the practical plant modelling system suggested in the opening chapter can be divided into three categories:

- ***Practicality***
  - The system must be easy to learn and intuitive to use
- ***Realism***
  - Plants generated must resemble real plants
- ***Rendering speed***
  - Plant models generated must be simple enough for use in a real-time environment where rendering speed is of great concern

3.1.1 Practicality

Plant modelling systems that require the user to have a vast knowledge of plant structures, and require the input of hundreds of parameters, are not usually trivial to master. A practical modelling system is a system that is simple and intuitive, and while easy-to-learn modelling systems often fail in the modelling of botanically accurate plant models, ease-of-modelling is often a greater priority. The practicality of a modelling system is often improved with the addition of a graphical user interface, allowing the user to model interactively.

3.1.2 Realism

The realism of models plays an important role in the development of a virtual environment, in order for the virtual environment to be believable. While plants are often represented by an abstract set of geometry (for example, a green sphere atop a brown cylinder) that is easy to model and rapid to render, “realism” in the scope of this thesis refers to the creation of models that resemble actual plants.

3.1.3 Rendering speed

In the development of models for use in a real-time environment, we need to ensure that the complexity of the models is low enough to enable acceptable rendering speeds. For the purposes of this thesis, we will consider real-time rendering to be a frame rate (frames rendered per second) of ten or higher. A frame rate lower than this causes the animation to appear jagged to the user, and the virtual environment may sacrifice much of its realism.

3.2 An object-oriented approach to modelling with components

Our aim in this section is to design an object-oriented component-modelling system, with particular focus on practicality of modelling and realism of models produced. We use as a foundation the component modelling system of Lintermann and Deussen discussed in section 2.3. This section begins with an analysis of their system, in terms of practicality, realism and rendering speed. Possible changes and additions to the system are then discussed, followed by ways in which component parameters can be defined.

3.2.1 The current system

3.2.1.1 Practicality

Lintermann and Deussen show that their component modelling system is indeed simple and intuitive by means of a test in which inexperienced users were required to model complex plant structures with only a brief introduction to the system. The results produced by the inexperienced users were impressive (see

Figure 5), demonstrating the ease-of-modelling provided by the system. Their system also includes a powerful user interface, through which plant components can be quickly and easily modelled and linked to form an overall plant structure.

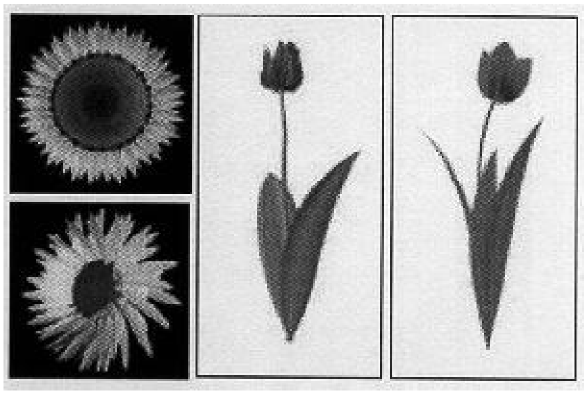


Figure 4 – Models produced by inexperienced users using a component modelling system

3.2.1.2 Realism

The degree of realism of this technique lies heavily in the hands of the user. Through manipulation of the component parameters and component hierarchy, both realistic and non-realistic looking plant models can be achieved. The models demonstrated by Lintermann and Deussen, while not necessarily being botanically accurate, show a strong resemblance to the actual plants they were intended to mimic.

3.2.1.3 Real-time rendering

Lintermann and Deussen have given a fair amount of thought to the speed of rendering of the models produced. Many of the models generated by their system required upward of one second per frame, rendering on an SGI Indigo 2 Extreme (134 MHz). While such times are unacceptable in a real-time environment (as discussed in section 3.1.3 above), we do realise that modern hardware is able to render polygons at a much higher rate than when this system was introduced. Therefore, while we expect our rendering times to outperform those of this system, the models produced may not be of significantly lower complexity.

Lintermann and Deussen suggest two techniques to improve poor rendering times:

- Complexity reduction
  - Reducing the number of triangles displayed thus increasing the rendering speed
- Hiding of components
  - A hidden component produces no geometry but generates subsequent components in the usual manner

The first is the most important, and widely used, technique for achieving rendering speed-up. The increased rendering speed obtained by a reduction in the number of polygons of a model often outweighs the loss of detail, and hence realism. Small plant models with few components are able to make use of complex geometry while maintaining suitable rendering speeds, while large plant models often need to sacrifice geometric complexity.

3.2.2 Changes and additions

One of the major differences between our system and that of Lintermann and Deussen is the presence of a graphical user interface. Our system takes a more abstract approach to component modelling, providing no graphical interface, instead requiring the modeller to preset the parameters of each component and set the associations between the components. The practicality lost through the exclusion of the user interface is offset somewhat by the improved modelling precision attained.

Because we are providing an object-oriented framework for modelling, rather than a fully-fledged modelling application, our system is designed to be highly extendable. We allow for the easy creation and integration of new components into the modelling framework. This is further discussed below.

Another difference between the two systems is the number of components provided. We feel that although the large number of components provided by the current system allows for the creation of almost any plant, a small subset of those components is all that is required to model the majority of plants. The provision of extra features often increases the learning curve, and can complicate the system. By allowing new components to be designed and integrated into the system, we have not ruled out the inclusion of other components. The components that we feel are essential to the system are (along with their original names):

- Stem component (Horn)
- Leaf component (Leaf)
- Tree component (Tree)
- Radial component (Hydra)

3.2.3 Component design

It is important to clarify at this point that little is known regarding the actual implementation of the component modelling system of Lintermann and Deussen. The way in which component attributes are modelled in the new system may differ substantially from the way in which they are modelled in the current system.

Collectively, components represent the plant in its entirety – they are, in fact, the only objects in this system. While the components themselves specify the geometry of the plant elements, the manner in which the components are connected specifies the manner in which these elements relate to one another. We begin this section with the design of the component structure, followed by a discussion on the ways in which component parameters may be represented. Finally, we discuss the design of the four components used in this system.

3.2.3.1 Structure

Components are intended to encapsulate data (attributes) as well as algorithms for the manipulation and representation of that data. It is for this reason that we have chosen a purely object-oriented approach – the encapsulation of data and corresponding methods is an inherently object-oriented idea. In addition to these, all components have the following common properties:

- A set of component connectors
  - All objects have the ability to have other components connected to them as children
- Each component is responsible for its own visualisation
  - Each components contains algorithms for the visual representation of its data
- Each component, after visualising itself, instructs its children to do the same

3.2.3.2 Attributes

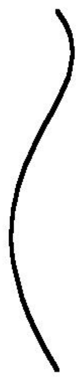
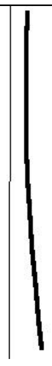
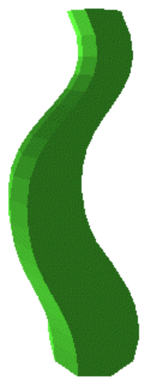
As we are unsure of the manner in which Lintermann and Deussen model component attributes, we cannot compare our methods of handling attributes with theirs. We introduce here two techniques for the specification of plant attributes.

3.2.3.2.1 Single-valued attributes

Plant parameters often require only a single value in their definition. Examples of this, in the context of component modelling, are the amount of geometric detail required and the number of branches tree should generate.

3.2.3.2.2 Spline-valued attributes

Inspired by the research of Prusinkiewicz et al [PruMün 01] into the “use of positional information in the modelling of plants”, this method of plant-attribute specification involves the definition of a spline curve that allows a given attribute to vary over distance. Examples of attributes that are suited to spline definition are the shape and radius of a stem. Figures 6, 7 and 8 show the spline definition of shape and radius attributes for a stem component and their effect on the resulting geometry.

 **Figure 5 – A spline definition of the stem shape**	 **Figure 6 – A spline definition of the stem radius. The distance between the spline and the line represents the radius at that point.**	 **Figure 7 – The resultant geometry.**

3.2.3.3 Geometry components

The components introduced here are those that generate geometry. We will discuss the purpose of each component as well as various ways in which the component geometry can be created, comparing the methods with one another in an attempt to find the best.

3.2.3.3.1 Stem component

This component generates cylindrical geometry for the representation of stems, branches, twigs and any other plant element with similar geometric properties. There are a number of ways in which stem geometry can be produced. Two of these techniques are introduced below.

3.2.3.3.1.1 Cylinders and cones

A cylinder is a “solid or hollow body with straight sides and a uniform cross-section” [Oxford 92]. Most stems and branches are, to an approximation, cylindrical in nature making cylinders a natural choice in the representation of stem-type geometry. Not only are cylinders trivial to model, their low complexity also makes them fast to render.

These advantages are offset by the fact that cylinders, by nature, are without curvature, and while this can be a fair approximation, few stems or branches on plants are actually straight. Another problem when using cylinders for stem geometry occurs at the point where two branches meet. There is no smooth transition from one branch to another, causing the plant to have an unnaturally angular appearance and also causing “holes” to appear in the model as demonstrated in Figure 9.

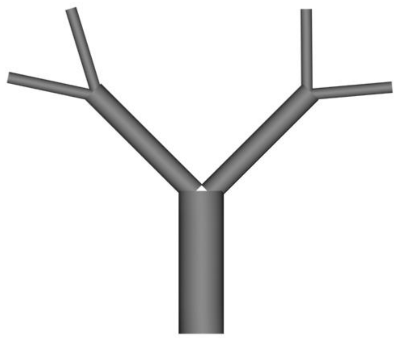


Figure 8 – Problems observed when modelling with cylinders. Holes, such as the one above the trunk in this image, can appear in the mesh detracting from the realism of the model.

A possible solution to this problem is to use cones. The only difference between a cone and a cylinder lies in the fact that cones have different start- and end-radii, thus causing them to appear less uniformly straight. Another possible solution is to apply texture mapping to the cylinder or cone as this often gives the appearance of smoother geometry.

3.2.3.3.1.2 Generalised cylinders

A vast improvement on regular cylinders and cones, generalised cylinders overcome many of the problems introduced above. A generalised cylinder can follow any path and can have a varying radius along its length. They allow for the creation of smoothly curving stems and branches and, if modelled correctly, allow smooth transitions between branches and prevent holes from appearing at branching nodes.

The only major disadvantage of generalised cylinders is their complexity. Due to their curvature, generalised cylinders require many times more polygons that regular cylinders. This is demonstrated in Figure 10. They also require significantly more computation to generate, due to the number of mathematical operations needed to calculate vertex positions.

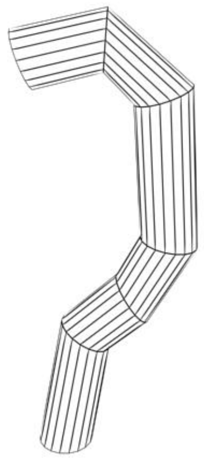


Figure 9 – A generalised cylinder

The benefits of generalised cylinders greatly outweigh their shortcomings. The added realism produced by non-rigid stems and branches is well worth the sacrifice of having higher complexity.

3.2.3.3.2 Leaf component

The leaf component is responsible for the generation of leaf-type geometry. This includes plant elements such as petals and even grass. These elements are two-dimensional by nature – a fact that can be taken advantage of for the purposes of practical modelling. We introduce two methods of modelling leaves and discuss the advantages and disadvantages of each.

3.2.3.3.2.1 Polygon definition

Defining leaf geometry by polygon definition involves the explicit definition of each polygon point, or vertex, of the leaf outline. The advantage of this technique is the ability to model the leaf shape exactly, allowing for the creation of any types of leaf.

The primary disadvantage of this technique is its inability to model smooth leaves accurately. Since each vertex in the leaf shape must be explicitly specified, a smooth curve would require the definition of many vertices – a highly impractical approach.

3.2.3.3.2.2 Spline definition

Using a spline to define the outline of the leaf alleviates some of the problems introduced in the polygon definition method above. Smooth curves are easily modelled with splines using only a few strategically place control points.

The major disadvantage of this technique is the inability of a single spline to model jagged edges. Leaves seldom have smooth edges all around. Most at least have a pointed edge.

A partial solution to this problem is to model only one half of the leaf. Due to the symmetry of leaves in general, the other half of the leaf can be assumed to be a mirror image of the first. This allows for the modelling of a pointed edge as shown in Figure 11, but still does not allow for jagged leaf edges.

Either method of leaf-geometry generation is acceptable. The arguments for each of these two methods is based on potentially enhanced realism (polygon definition) versus ease of modelling (spline definition). For this system, we choose the more practical approach, and implement the spline-based method.

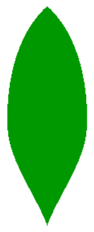


Figure 10 – Through spline definition of one half of a leaf, the other half can be mirrored, allowing for the creation of pointed leaves.

3.2.3.4 Distribution components

Distribution components generate no geometry directly. The Tree component, as discussed below, has a Stem component attribute that may generate geometry, but for the purposes of this discussion this will not be considered direct creation of geometry. The purpose of a distribution component is to generate multiple instances of subsequent components and arrange them in a specified manner about a parent component.

3.2.3.4.1 Tree

The Tree component provides for the distribution of subsequent components as branches about a central axis, defined by a Stem component. The Stem component is not only used to calculate the positions and orientations of the Tree branches, but can also be used to generate the geometry of the Tree stem. The distribution properties of the Tree component are introduced here.

3.2.3.4.1.1 Positioning of branches

There are a variety of ways in which branches can be position along the stem. One method is to define the distance between each branch. This can be as a constant value, in which case branches are equally spaced along the stem, or as a spline (as in [PruMün 01]).

A simple method of positioning branches is to specify the number of branches required, and spread the branches equally along the stem. This is identical to defining a constant-valued distance between branches, except that a specification of the number of branches required is more intuitive. This is the method we will use in the design of this system.

3.2.3.4.1.2 Orientation of branches

Now that we have determined the positioning of the branches, we need to determine the orientation of the branches. Two orientation properties are used in the Tree component and both will be discussed here and are demonstrated in Figures 12 and 13.

3.2.3.4.1.2.1 Branching angle

The branching angle is the angle that the branch makes with the stem. This can be accomplished through the use of a spline (as in [PruMün 01]), but this proves clumsy without the use of a graphical user interface with interactive feedback. We settle for a more practical approach in which the branching angle is the same for all branches. This can, however, detract from the realism of the plant as few stems have branches that all have the same branching angle.

3.2.3.4.1.2.2 Inter-node angle (Phyllotaxis)

The inter-node angle, or phyllotaxis, of a plant specifies the angle about the stem axis between two successive branches. In a majority of plants, this angle is fairly constant. While a random angle could provide interesting results, for the sake of realism and better modelling control, our system implements a constant valued inter-node angle.

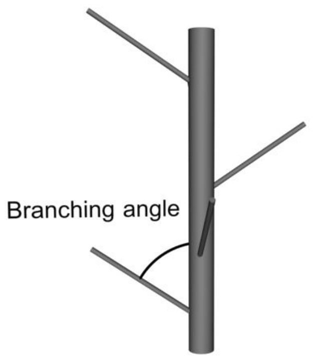


Figure 11

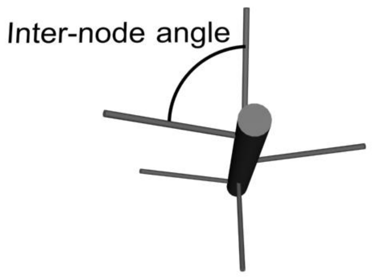


Figure 12

3.2.3.4.1.3 Scaling of branches

Very seldom in a tree structure are the branches all the same size. The size of the branches usually varies over the length of the stem in a pattern that differs from one plant to another; some plants have large branches at the base of the stem and small branches at the top, while others are exactly the opposite and still others have small branches at the top and bottom with large branches in the middle. For the sake of realism, we cannot therefore specify a general scaling function, but rather allow the user to define one. The most obvious method of definition is a spline (in the same manner as a spline was used in the definition of the Stem component radius above). The branch scaling spline effectively defines the shape of the Tree component.

3.2.3.4.2 Radial

The function of this component is to distribute subsequent components in a radial fashion about a centre point. Some of its most common uses are in the distribution of petals to form a flower, or the distribution of leaves about a stem. By assigning Radial components as branches of a Tree component, the effect of multiple branches per “branch-node” can be achieved.

The distribution properties in Radial component affect only the orientation of subsequent components. As all subsequent components are distributed about a single centre point, there is no need for any positional properties as in the Tree component. The three basic properties that affect the distribution of components in the Radial component are:

- Inter-node angle
  - Similar to phyllotaxis – specifies the angle, about the axis of the parent component, between nodes
- Node elevation angle
  - Similar to branching angle – specifies the angle that each node makes with the axis of the parent component
- Node twist angle
  - Specifies the angle of rotation of a node about its own axis

These parameters are fairly self-explanatory, and as such will not be discussed in any more detail here.

3.3 An interactive modelling system based on ramification matrices

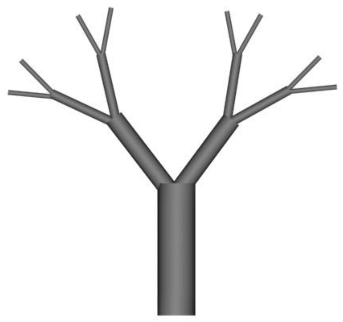
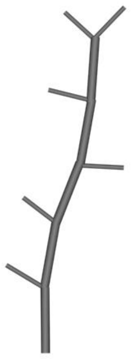
This section will focus on the design of a graphical user interface based system for the modelling of plants based on the ramification matrix concept as introduced in section 2.2. Our aim in this system is to produce a very easy-to-learn, intuitive modelling system that relies on only a handful of core modelling properties in the generation of plant geometry. We begin this section by defining these core properties and ascertaining their relationship with the actual ramification matrix. This is followed by the design of a graphical user interface for the modelling of these properties. Finally, we discuss the criteria for the plant properties that are automatically generated.

3.3.1 Core plant properties and their relation to the ramification matrix

We introduce a core set of plant properties, and discuss the effect such properties have on the values of the ramification matrix. Some properties do not actually affect the matrix itself, instead affecting other plant properties. The set of core plant properties that will be modelled interactively in this system are (along with their effect on the actual ramification matrix):

- Depth
  - The maximum depth of the tree
  - The effect of maximum depth on the ramification matrix is simply to add or remove rows and columns from the matrix
- Twist
  - The maximum angle about the axis of a parent branch through which a child branch can rotate
  - The twist property does not actually affect the matrix itself, but is a separate property of the plant
- Balance
  - A measure of the "bushiness" of the tree – a tree with low balance is just a tall trunk, while a tree with high balance should bear a strong resemblance to a perfect tree[1]
  - The balance property affects the probabilities within the matrix. A low balance indicates high probabilities on the left-hand (low branch order) side of each row, while a high balance indicates high probabilities on the right-hand (high branch order) side of each row. This is demonstrated in Figures 14 and 15.

[1] A perfect tree is one in which every branch splits equally into two identical branches

 Figure 13 – A tree with a high balance value	 Figure 14 – A tree with a low balance value

3.3.2 A graphical user interface

A graphical user interface can play an important role in practical modelling. Allowing the user to modify plant parameters interactively can often make the modelling process a lot easier, and while manually entering parameter values allows for greater accuracy of modelling, the manipulation of intuitive controls to achieve a desired "look" is often more desirable. An over-complicated user interface, however, can be a liability. In this section we discuss the design of an effective graphical user interface for the modelling of plant parameters within a ramification matrix-based system.

Each of the properties introduced in the preceding section could be represented by one of any number of graphical components. Of all the available components, the most logical choice is the slider, due to its ranged nature. All the above properties are best modelled by allowing the modeller to select values from within a specified range. The user can therefore simply adjust each of the sliders until the desired "look" is achieved. The graphical user interface for this system is shown in Figure 16 below.

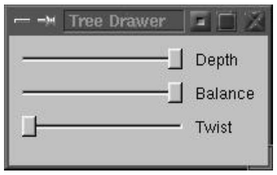


Figure 15 – A set of graphical components for the modelling of ramification matrices

3.3.3 Automatic calculation of branch properties

Properties of branches, such as length, width and branching angle, need not be directly specified by the user, but can be automatically calculated based purely on the order of the branch. The reason for this automated technique is that in nature, branch lengths, widths and branching angles are often directly related to the order of the branch. Through automatic manipulation of these properties, the modelling process becomes even easier for the user. Each of these properties is introduced here in more detail.

3.3.3.1 Branch width and length

The width and length of branches, in actual plants, is highly dependent on the order of the branch. Branches of high order (close to the trunk) are usually longer and thicker than the twigs at the edge of the tree. This can be taken advantage of to automatically create branches whose width and length are functions of their order.

3.3.3.2 Branching angles

Another property that, in nature, is dependent on branch order is the branching angle (angle between a branch and the parent axis). In this case, however, the dependency is not simply on the order of a single branch but rather the on the order of the two branches involved in the bifurcation[2]. Generally, the bifurcation of two branches of equal order results in both branches having an equal branching angle. In the bifurcation of two branches with different order, the larger

[2] Bifurcation refers to branching or forking

(higher order) branch tends to have a smaller branching angle than the smaller branch.

3.4 An elementary L-System parser

We introduce here a framework for the modelling of elementary parametric L-Systems. Rather than improve on the practicality, realism and rendering speed of other L-System parsers, this system is purely intended to demonstrate the design considerations and implementation issues involved in the construction of an L-System parser.

3.4.1 Modules

As mentioned in section 2.1.1.1, an L-System module consists of a symbol and a set of parameters. There are a vast number of potential modules, but as this is an elementary parser, only a core set of modules are selected for implementation. Other modules are, however, easily added to the system. The selected modules, along with their parameters are specified in Table 3.1 below.

Symbol	**Params.**	**Action**
F	*S*	Causes the turtle (as described in section 2.1.4) to move forward by *s* units, generating branch geometry along its path
-	*A*	Causes the turtle to rotate by *a* degrees in a clockwise motion about an axis perpendicular to the parent branch
+	*A*	As above, except that motion is counter-clockwise
/	*A*	Causes the turtle to rotate by *a* degrees in a clockwise motion about the axis of the parent branch
\	*A*	As above, except that motion is counter-clockwise
[		Causes the turtle-state to be pushed onto the stack
]		Causes the turtle-state to be popped from the stack

Table 3.1 *- Modules selected for implementation*

3.4.2 The parallel rewriting process

In the definition of an L-System, the user provides an initial L-System string, as well as a set of productions. Upon receiving an instruction to rewrite the given string, the system performs the following actions iteratively:

- The next module, *m*, is selected from the given string
- *m* is compared with the predecessor modules of the productions
- If a match is found
  - o *m* is replaced with the successor string of modules from the production
- If a match is not found
  - o *m* remains unchanged in the string

3.4.3 Parsing the string

Once the L-System has been "grown" (rewritten) to the user's satisfaction, the user may request that the string produced be parsed. This involves applying the actions associated with each module in the string, as described in Table 3.1 to generate the actually geometry of the plant..

3.5 Using billboards to obtain high rendering speeds

We introduce here a method of plant visual representation that allows for very high rendering speeds. A billboard is a single texture-mapped polygon; the texture, in this case, is an image of a plant or tree. The high speeds of rendering obtained are due to the fact that a single polygon (usually consisting of only two triangles) is simple to render, and the graphics hardware is often optimised for rendering just such polygons.

In this section we look at some of the ways in which billboards can be arranged for different effects, followed by ways in which billboard textures can be created.

3.5.1 Billboard-arrangement

Various methods of billboard-arrangement exist for the representation of a tree, some of which are introduced here.

3.5.1.1 Single billboards

Single billboards are extensively used in vehicle simulations to provide, for example, forests along the side of a road. These billboards are static (their position and orientation does not change) and are primarily used in view obstruction. Using a closed loop of single billboards can give the impression of a cluster of trees. Using a single billboard to represent a single tree on a landscape is not very effective due to its two-dimensional nature, which becomes very clear to the user upon observing the billboard from different positions.

Directional billboards provide a solution to the last point mentioned above. They are identical to the billboards introduced above, except that they are not static – they re-orientate themselves so as to face the viewer at all times. Thus, the user sees the same image no matter from which direction he/she faces the billboard. While eliminating the obvious two-dimensional nature of non-directional billboards, these billboards are still not ideal in the representation of individual trees; always seeing the same side of a tree is not realistic. Directional billboards prove most useful when intended only to be viewed from a fair distance, where the user cannot rapidly circle the tree and observe the defects.

3.5.1.2 Double billboards

In an attempt to solve the problems introduced above, double billboards are introduced. Double billboards consist of two perpendicularly orientated single billboards, and are static in nature. They provide the user with a model that appears more solid, giving a more three-dimensional appearance to the tree. If the user chooses to circle the tree, the observed result is far superior to that obtained with single billboards.

3.5.1.3 Triple billboards

An extension to double billboards, triple billboards include a third single billboard perpendicularly orientated to the other two (facing upwards). Triple billboards are used in applications where the user may be positioned above the tree looking down on it (for example, a flight simulator). In this case, the essentially two-dimensional nature of double billboards would be highly noticeable.

3.5.2 Creating the texture

3.5.2.1 The problem at hand

The billboard texture is not quite as trivial as often assumed. If an image of a tree is simply pasted onto a billboard, the result will be a rectangular, coloured shape in the environment containing an image of a tree in its midst. The edges of the tree need to be somehow removed so as to make it appear less solid.

3.5.2.2 A solution – the stencil

A solution to the above problem is to create not only the image of the tree, but also a stencil, or mask, that marks the parts of the image to be “cut out”. The stencil is provided as a monochrome (black and white) image where white represents the part of the image that is to remain visible. A sample image and corresponding stencil is shown in Figure 17.

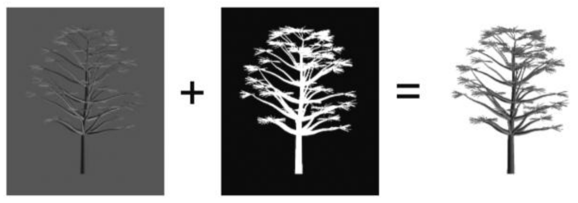


Figure 16 – Applying a stencil to an image to produce a “cut-out” texture for a billboard

3.6 Summary

In this section, we have discussed the design of four systems for the modelling of plants. Each system has had its strengths and weaknesses identified, and each system design is modelled around this. We have identified a powerful method of parameter definition in the use of splines, as well as a powerful method of branch geometry representation in the generalised cylinder. The implementation of these designs is discussed in the following section.

4 Implementation

4.1 Component Modelling

4.1.1 A spline-modelling tool

As splines are extensively used within the component modelling system implemented here, we introduce an object, Spline, which allows the creation of, and interaction with, two-dimensional multi-control-point splines.

4.1.1.1 Points on the curve (SplinePoint)

All points on the spline curve have a coordinate position, as well as an orientation, or direction, vector. These properties are encapsulated within the SplinePoint object, which is used both in the definition of the curve and in the retrieval of interpolated points. A brief description of these properties is provided below.

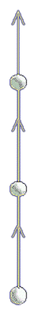
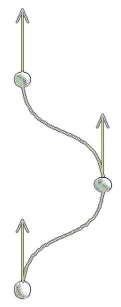
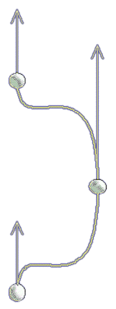
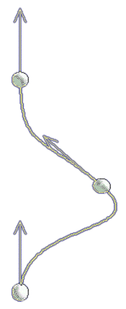
- Position
  - Consists of an x- and y-coordinate
  - Marks the exact position of the point in the coordinate space
- Orientation vector
  - Consists of an angle (direction) and magnitude
  - Direction is equal to that of the tangent to the curve at that point
  - Magnitude used only in the definition of control points

4.1.1.2 Definition of the curve

The spline curve is defined by a set of *c* control points, modelled as SplinePoints. The position and orientation of these control points is set, and the point is then passed to the Spline object. The order in which the control points are passed to the Spline objects is assumed to be the order in which they are subsequently interpolated. The effect of the control point properties on the resultant curve is discussed below.

- Position
    - o The spline curve passes through all the control points
- Orientation angle
    - o The spline curve enters and exists a control point at the orientation angle
- Orientation magnitude
    - o Defines the distance on either side of the control point within which the spline curve conforms to the orientation angle

These effects are demonstrated clearly in Figures 18 to 21.

 **Figure 17**	 **Figure 18**
 **Figure 19**	 **Figure 20**

4.1.1.3 Fractional distance definition

Because a spline can be composed of any number of control points, the length and shape of the spline can be anything. We need a way of accessing the position of any interpolated spline point without having to refer to the actual spline coordinates.

We suggest a method whereby a point on a spline is indicated by the fractional distance along the spline curve. This allows us to refer to points on a spline curve in the same way, no matter what shape, length, or number of control points the spline contains.

4.1.1.4 Retrieval of interpolated points and orientations

Once a spline has been defined by a set of control points, requests can be made to the Spline object for the position and orientation of an interpolated point. The Spline object is passed as a parameter the fractional distance of the required point from the start of the spline. The calculation of the position and orientation of the interpolated point is discussed below.

4.1.1.4.1 Calculation of an interpolated position

The interpolated position is calculated by means of cubic, Hermite-interpolated spline patches[3]. Each control point, except for the first and last control points, marks the start and end of a spline patch. In essence, the overall spline is composed of a number of spline patches linked together.

Since the function for the interpolation computation (see Appendix) uses only two control points in its calculation, we need to determine in which spline patch the requested point will appear. This patch is easily determined using the equation below:

[3] The mathematics of cubic Hermite-interpolated patches can be found in the Appendix

$$n = \left\lfloor \frac{u}{\left(\frac{1}{(c-1)} \right)} \right\rfloor$$

where *u* represents the fractional distance along the spline and *c* represents the number of control points used in the spline definition. The required point will lie between the nth and the (n+1)th control points. We now require the fractional distance of the required point within the spline patch itself. This fractional distance is calculated using the equation:

$$l = \frac{u}{1-n}$$

where *l*, represents the fractional distance of the point between the nth and the (n+1)th control points.

The actual coordinate position of the interpolated point is then calculated using the parametric spline equations described in the Appendix.

4.1.1.4.2 Calculation of an interpolated orientation

To find the orientation of an interpolated point, we first need to calculate the tangent to that point on the curve. The line between two, very closely situated, points on a curve is an approximation to the tangent. We can therefore use the above equations to determine the position of a point infinitesimally further along the spline curve, and use these two points in the determination of the tangent. The orientation of the tangent is calculated using some simple trigonometry. The angle θ shown in Figure 22 is calculated using the equation:

$$\theta = \tan^{-1}\left(\frac{\Delta y}{\Delta x} \right)$$

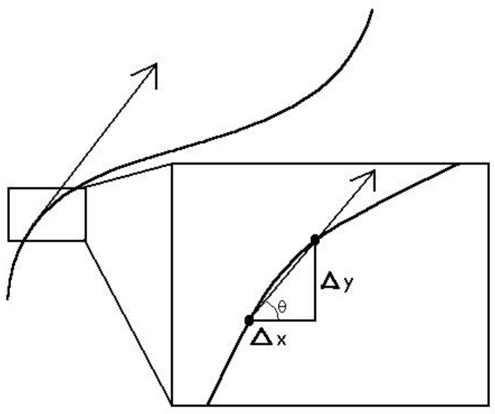


Figure 21 – Approximation of a tangent by means of two closely positioned points on the curve

4.1.2 Components

4.1.2.1 Structure of a component

All components in this system have the same basic structure. Each component consists of a set of attributes and methods relating to its specific function, as well as a Render method. In the case of a geometric component, the Render method initiates the generation and display of the component geometry. In the case of a distribution component, the Render method initiates the calculation of positions and orientations for subsequent components, and calls the Render methods of subsequent components. Thus, the invocation of the Render method of the initial component causes a “chain reaction” whereby a Render signal is essentially propagated through the component graph.

4.1.2.2 Geometric components

In this section we discuss the creation of geometry for each of the two geometric components: Stem and Leaf.

4.1.2.2.1 Stem component

The function of the Stem component is to generate geometry for plant elements such as stems, branches and twigs – geometry that is cylindrical in nature. The important attributes of the Stem component are listed in Table 4.1 below.

Attr. Name	**Description**
stemShape	A spline defining the shape of the generalised cylinder used to represent the stem geometry.
stemRadius	A spline defining the radius of the stem. Only the x-coordinates of the spline are used in calculation of the radius.
detail	The number of stemShape spline iterations used – a high detail value yields a very smooth curve.
meshDetail	The number of vertices used per spline iteration – the number of sides in the resulting cylindrical shape.

Table 4.1 *– Attributes of the Stem component*

4.1.2.2.1.1 Generation of geometry

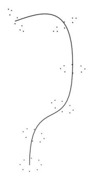
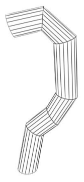
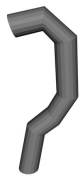
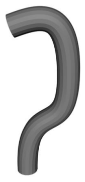
The Stem component produces as its geometry a form of generalised cylinder: a dynamic form of cylinder that follows a spline-defined path and has a radius that varies along its length. In this implementation, the radius of the cylinder is also defined by a spline.

The generalised cylinder is constructed in the following manner:

A number of equally spaced points, together with their orientations, are obtained from the stemShape spline. For every pair of points obtained:

- A circle of vertices is generated about each point
    - The circle lies in a plane perpendicular to the tangent vector of a point
    - The number of vertices is specified by the *meshDetail* attribute
- For each pair of points, quadrilateral polygons are generated linking the corresponding vertices

This process is demonstrated in Figures 23 to 26 below.

 Figure 22	 Figure 23	 Figure 24	 Figure 25

4.1.2.2.1.1.1 Calculation of generalised cylinder vertices

At each point on the stemShape spline, a ring of vertices are calculated in the x-z plane using the following equations:

$$x = r\cos\left(i * \frac{2\pi}{meshDetail}\right)$$

$$z = r\sin\left(i * \frac{2\pi}{meshDetail}\right)$$

where *r* is the radius of the generalised cylinder at that point on the stemShape spline, and *i* is an integer ranging from 0 to (*meshDetail*-1). The inclusion of the factor 2π is due to the fact that the angles are measured in radians.

Once these vertex-positions has been calculated, each vertex is rotated about the z-axis by the angle of the tangent to the stemShape spline at that point. Finally, the vertices are repositioned so as to be centred about the stemShape spline point.

4.1.2.2.2 Leaf component

The function of the LeafComponent is to generate leaf-type geometry. The primary attributes of LeafComponent are listed in Table 4.2 below.

Attr. Name	**Description**
leafShape	A spline that defines the outline of the leaf
detail	The number of leafShape iterations used

Table 4.2 – Attributes of the Leaf component

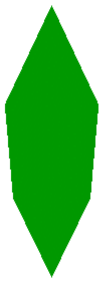
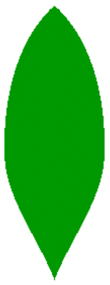
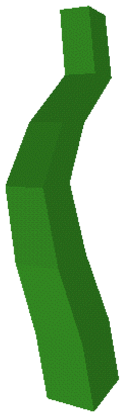
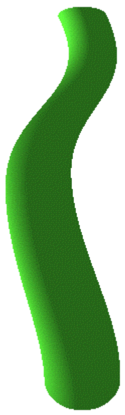
4.1.2.2.2.1 Generation of geometry

As mentioned in Table 4.2, leafShape defines the outline of the leaf. Due to the symmetrical nature of leaves, we need only define one half of the outline of the leaf. The other half can be calculated by mirroring the spline coordinates about the line x=0. The process of leaf geometry generation can be summarised as follows:

- A number (detail) of equally spaced points are extracted from the leafShape spline
- For each point, now treated as a vertex, a corresponding vertex is created with the same y-coordinate but a negated x-coordinate
- These vertices are used to describe a polygon of the colour specified by r, g and b

4.1.2.2.3 Geometric levels of detail

By using splines in the definition of the Stem and Leaf components, the geometric detail can be easily adjusted. This is accomplished by simply changing the number of interpolated points retrieved from the Spline object. This is demonstrated in Figures 27 to 30 where the only difference between the two sets of diagrams is the value of the *detail* parameter.

 **Figure 26 – Low detailed leaf**	 **Figure 27 – High detailed leaf**	 **Figure 28 – Low detailed stem**	 **Figure 29 – High detailed stem**

4.1.2.3 Distribution components

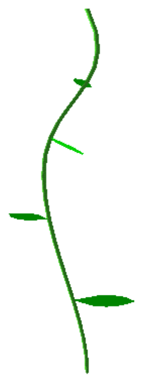
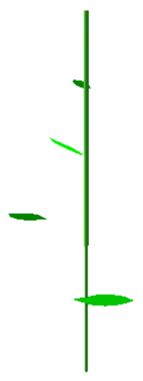
4.1.2.3.1 Tree component

The function of the Tree component is to calculate the positions and orientations of subsequent components about a stem. Some of the important attributes of the Tree component are introduced and discussed here.

4.1.2.3.1.1 The stem

The Tree component requires a Stem component, not only to generate geometry (in fact, geometry generation is not obligatory as discussed below), but to provide a central axis about which subsequent components may be distributed. A flag, *stemVisible*, can be set to false preventing the stem geometry from being rendered. This is most useful when modelling multiple sets of branching components about the same stem; the stem geometry need only be generated once.

A consideration if using low detail geometry for the stem is that the branch positions still follow the actual spline rather than the generated geometry. This may cause the branches to appear disconnected to the plant stem as demonstrated in Figures 31 and 32.

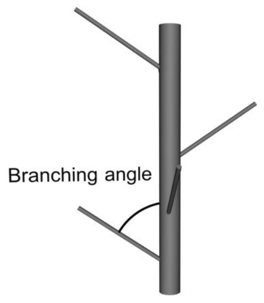
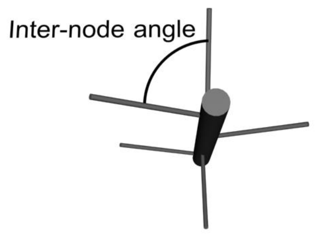
 **Figure 30**	 **Figure 31**

4.1.2.3.1.2 Scaling of branches

Because the same component is being rendered at each node position, we need a scaling function to prevent all branches being of equal length. The most obvious method of defining a scaling function, considering the tool developed above, is by means of a spline. The scaling function is defined in much the same way as the radius function for the Stem component. At each branching point along the stem, the scaling factor of that branch is taken to be the x-coordinate of the corresponding point on the *branchScale* spline. The *branchScale* spline effectively allows us to define the overall shape of the tree.

4.1.2.3.1.3 Inter-node rotation and branching angles

The orientation of branches is defined by these two attributes. Inter-node rotation specifies the angle about the stem axis between subsequent branches. The branching angle specifies the angle that the branch makes with the stem. Both of these attributes are shown below.

 **Figure 32**	 **Figure 33**

4.1.2.3.1.4 Adding variation

Real trees do not have a perfect spline outline, nor do all their branches have identical branching angles. We therefore introduce two attributes, *branchScaleVariation* and *branchAngleVariation*, which cause a specified amount of deviation from the values of their parents. The variation attributes are specified as the maximum fraction of the value of their parents by which they can deviate. The equations used in the calculation of the variations are:

$$s_v = s - 2rv_s s$$

where s_v is the varied scale, s is the unmodified scale value, v_s is the *branchScaleVariation* and r is a random number between 0 and 1.

$$a_v = a + (v_a - 2rv_a)$$

where a_v is the varied branch angle, a is the unmodified branch angle, v_a is the *branchAngleVariation* and r is a random number between 0 and 1.

The reason for the difference between the two equations it the fact that *branchScaleVariation* represents a fraction of the *branchScale* value, while *branchAngleVariation* is an angle representing the maximum deviation from the *branchAngle* value.

4.1.2.3.2 RadialComponent

The function of the RadialComponent is to distribute subsequent components in a radial fashion about a central point. The attributes that specify the distribution are:

- *interChildAngle*
    - specifies the angle between the distributed components
- *nodeAngle*
    - specifies the angle that a subsequent component makes with the parent component
- *nodeTwist*
    - specifies the angle of rotation of a component about its own axis

Variation is added in a similar fashion to that of the Tree component, except that the variation is defined in degrees rather than as a fraction of the parent value.

4.1.3 Putting the pieces together

Once each component has been individually modelled, they are linked to one another by means of the component connectors common to each component. Thus the component hierarchy is formed, and a call to the “root” Render method causes all plant geometry to be generated in its correct position and orientation.

4.2 An interactive modelling system based on ramification matrices

4.2.1 Relating slider values to actual parameters

Once a slider value has been selected, we need to convert that value into something applicable to the property it represents. For discrete-valued properties such as *depth*, manipulation of the slider value is unnecessary. For dynamic, non-discrete-valued properties such as *twist* and *balance*, however, the slider value must be interpreted and manipulated to achieve a desirable output. We discuss these manipulations below.

4.2.1.1 Twist

The *twist*-slider is assigned a range of 0 (zero) to 360 degrees. We take the *twist* parameter to be a random number between zero and the value of the *twist*-slider. Providing a different twist-slider for every depth level would be impractical. We therefore treat twist as a global tree property – a new random number is chosen for every branch.

A better result is obtained if we allow the lower order branches (branches closer to the leaves) to have a greater twist than the higher order branches. This can be easily accomplished by adding an inverse-depth multiplier to the random twist value, as shown in the equation below:

$$t_f = rt_i\left(d_t + 1 - d_c\right)$$

where t_f represents the resultant twist value, r is a random number between 0 and 1, t_i is the value of the *twist*-slider, d_t is the depth of the tree, and d_c is the depth of the current branch being twisted.

4.2.1.2 Balance

The balance-slider is assigned a range from 0 (zero) to 100. Manipulation of the balance slider value poses a problem: from a single value we must extrapolate an entire ramification matrix. For a low balance value, the matrix probabilities must be weighted in favour of lower depth levels, and a high balance value in favour of higher depth levels.

It is clear that we require a function that, for a low balance, generates high values for the lower depth branching probabilities, and low values (close to zero) for the higher depth branching probabilities. The same function should be applied to all the rows (depth levels) in the matrix, each time scaled to meet the length of the row.

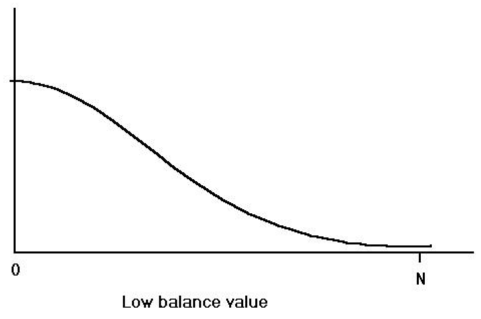
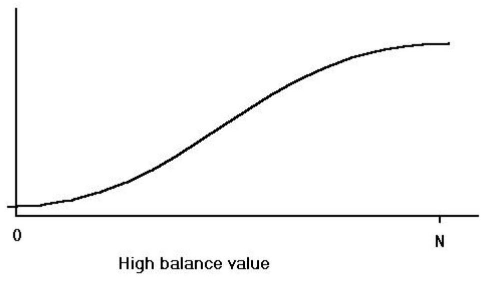
4.2.1.2.1 A sinusoidal solution

A vast number of functions could be used for the purpose of calculating ramification matrix values based on a single *balance* value. In this implementation we introduce a sinusoidal function for this purpose.

A sinusoidal function can be used to generate a peak value, with successively smaller values on either side. By including a “sliding” term in the sinusoidal function and limiting the parameter to a value between -π and +π, for example

$$y = \cos[\pi(x - s)]$$

where *s* is the sliding term and ranges from 0 (zero) to 1, and x ranges from 0 (zero) to 1, we can create a sinusoidal function whose peak varies depending on the value of *s*, which it is now clear is a representation of *balance*.

 **Figure 34**	 **Figure 35**

4.2.1.2.2 Applying the function to the ramification matrix

The sinusoidal function (as shown above) must now be tailored to generate values for the ramification matrix. As mentioned above, the values of *x* and *s* range between 0 (zero) and 1. They can therefore be calculated as fractions of a total value.

For row i and column j, the value of *x* is taken to be $\frac{j}{i}$. Thus *x* represents the relative position along a row. The value of *s* is taken to be $\frac{b}{100}$, where *b* is the value of the balance-slider. *s* therefore represents the fraction of the total balance selected.

The final equation, for row i and column j,

$$y = 1 + \cos\left[\pi\left(\frac{j}{i} - \frac{b}{100}\right)\right],$$

where *y* represents the subsequent values of the matrix cells. The unit increment shown above is to force *y* to be a positive value (the natural cosine function produces values between –1 and +1).

4.2.1.2.3 Normalisation of the matrix

Because the values within the matrix represents branching probabilities, the sum of each row much be equal to 1. The function introduced above provides no guarantee that this will be the case, so we must normalise each of the values in the matrix.

This is fairly simply accomplished by summing each row and dividing each element in the row by the value of the sum. This maintains the ratios between the elements while ensuring that the sum of each row is exactly equal to 1.

4.2.2 Generating the tree

The tree structure is generated in a recursive manner as follows:

- Generate geometry for a branch
- Choose a branch order from the ramification matrix
- Rotate through a specified angle (twist and branching angle)
    - Generate a tree of depth level specified by ramification matrix
- Rotate through a specified angle (twist and branching angle)
    - Generate a tree of depth level specified by ramification matrix

4.2.2.1 Traversing the ramification matrix

The sum of each row in the ramification matrix is 1. To select a branching order, we therefore select a random number between 0 (zero) and 1, and sequentially sum (from left to right) the probabilities in the row until the sum is greater than our random number. The column in which this occurs represents the new depth to which the current segment will branch.

4.2.2.2 Calculation of branching angles and branch lengths and widths

In nature, there is a strong correlation between the branch order (k) and the length and width of the branch, as well as the angle made with the parent branch. We therefore wish to keep this trend in our implementation, and require that the length, width and branching angle are functions of the branch order (k). These functions are empirical by nature – a variety of values can be tried and the best ones kept. Functions suggested by Viennot et al [Viennot 89] provide realistic results:

$$L(k) = c_1 k$$

$$W(k) = c_2 L(k)^{\frac{d_t}{2}}$$

where L(k) is the length of a branch of order k, and W(k) is the width of the branch. The constants c_1 and c_2 are arbitrarily chosen while d_t is the depth of the tree. In our implementation we found that values of:

c_1 = 0.0625
c_2 = 0.5

produced the most satisfactory results.

The branching angles, as discussed in section 3.3.3.2 above, can be calculated from the relative depths of the branches. If both branches are of the same depth, the angle between them is equal and is specified by a constant value. If the branches are not of equal depth, however, the following equations (suggested by Viennot et al) can be used to calculate their respective branching angles:

$$\theta_m(k,i) = c_m * \frac{i}{k-1}$$

$$\theta_s(k,i) = c_s * \frac{k-i}{k-1}$$

where θ_m is the angle between the main (depth level k) branch and the parent axis, and θ_s is the angle between the secondary (depth level i) branch and the parent axis. These angles are demonstrated in the figure below.

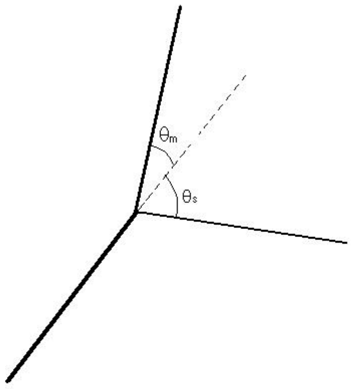


Figure 36 – Automatic calculation of branching angles based on the order of the respective branches

4.2.2.3 Generating geometry

The branch geometry can be generated in a number of ways. We implement cone-type geometry, in which the starting radius of the cone is the width of the branch, and the end radius is the width of the next (highest depth level) branch. The leaves in this implementation are polygon-defined.

4.3 An elementary L-System parser

The implementation of this system is a relatively simple task. In this section we will discuss the structure of a module, followed by a look at an effective method of storing L-System strings, and finally the processes involved in writing an L-System parser.

4.3.1 The structure of an L-System module

An L-System module, as discussed in section 2.1.1.1, consists of a symbol and a set of attributes. This is conveniently represented as an object with the following structure:

Attr. Name	***Type***	***Description***
symbol	char	A single character representing the symbol
numAttr[0] │ numAttr[4]	double	An array of numerical attributes. The module may therefore contain at most five attributes.

4.3.2 L-System strings

An L-System string is a string of module objects. A useful structure for storing such a string is a Vector. A Vector is essentially an array with dynamic size. As elements are added to a Vector, so the Vector increases its maximum size and as elements are removed from a Vector, so the Vector frees the memory used by that element.

4.3.3 Rewriting the L-System string

We implement a dual string rewriting system whereby the rewritten string is stored in another Vector. In order to rewrite an L-System string all that needs to be done is to iterate through the provided L-System Vector, at each module comparing the symbol field with a set of predecessor symbols. If a match is found, a set of successor modules is placed into the new Vector. Once this process is complete, the provided L-System Vector is replaced with the new Vector. This is shown in the pseudo-code below:

```
while (there are modules in the provided L-System Vector)
     module = lSystemVector.nextElement ()
     switch (module.symbol)
```

```
          ‘F’:newVector.addElement (‘[‘)
               newVector.addElement (‘F’, 1.0)
               newVector.addElement (‘]’)
          ‘A’:newVector.addElement(‘A’,module.param[0]-0.5)
lSystemVector = newVector
```

4.3.4 Parsing the L-System

Parsing the L-System to generate the plant geometry is accomplished in a similar manner to the rewriting process described above. The symbol of each module is compared to a list of symbol, and if a match is found the corresponding set of instructions is executed. This process is demonstrated by the fragment of pseudo-code below:

```
while (there are modules in the provided L-System Vector)
     module = lSystemVector.nextElement ()
     switch (module.symbol)
          ‘F’: Draw a branch of length module.param[0]
          ‘-‘: Rotate by module.param[0] degrees clockwise
          ‘[’: Push turtle state
          ‘]’: Pop turtle state
```

4.4 Billboards

The only real technical aspect involved in the implementation of billboards is the creation of a stencil-affected texture. This process is discussed in detail here.

4.4.1 Alpha values

Every point in a texture is defined by four values: red, green, blue and alpha value. The red, green and blue values describe the colour of the point, while the alpha value describes the “visibility” of the point on a scale from 0 to 1 (where 1 is completely opaque). In the creation of a stencil-affected image, it is these alpha values that need to be affected by the corresponding points on the stencil image.

4.4.2 Hiding unwanted regions

There are three elements in this discussion: the *stencil*, the *image* and the *texture*. Our aim is to create the texture with the colours derived from the image, and the opacity derived from the stencil. Using alpha values (introduced above), this is accomplished as follows:

- for every point (x, y) in the *texture*
    - assign the (r,g,b) value of the *texture* point the (r,g,b) value of the corresponding *image* point
    - if the corresponding point (x, y) in the *stencil* is black
        - assign the *texture* point an alpha value of 0 (transparent)
    - otherwise
        - assign the *texture* point an alpha value of 1 (opaque)

4.5 Summary

In this section, we have discussed the implementation of the systems designed in chapter 3. We have introduced a spline modelling system, and shown how it can be used to model component parameters effectively to produce more realistic results. We have also discussed the creation of a graphical user interface for the interaction with a ramification matrix, and described functions that can be used to relate the graphical components to the matrix itself. The parsing of L-Systems is discussed in greater depth, with particular focus on methods and data structures that can be used in the L-System rewriting process. Finally, we introduce a method of using stencils in the generation of billboard textures, in an attempt to allow billboards to blend into the environment.

5 Results

The results obtained from each of the four systems are discussed here in terms of the project goals: practicality of modelling, realism of models produced, and speed of rendering.

5.1 *Practicality*

Each of the systems implemented is discussed here in terms of their practicality of modelling.

5.1.1 Component modelling system

While a graphical user interface would certainly improve the practicality of the system, it was found that plants of all types could be modelled easy and quickly in the system as it stands. The ability to model each component individually (manipulating attributes closely related to actual plant properties), and then link components in a logical manner, made modelling the plants a fairly straightforward task.

The plants that were the easiest to model using this system were plants with low branching complexity. This includes plants such as flowers, palm trees and vines. Plants with high branching complexity, such as large trees and bushes, proved somewhat more difficult to model realistically.

5.1.2 Interactive ramification matrix modelling system

In some ways, the graphical user interface proved a success, and in other ways a failure. The modelling process is certainly made a lot easier through the inclusion of a graphical user interface; being able to interactively alter the plant properties saves a lot of time. However, in the interface developed in this project, we feel that the number of graphical components provided was too few. They did not allow the user enough control over the plant model. Automating some of the plant properties, such as branching angles, also detracted from the control of the user. Nevertheless, this systems was considered to be the easiest to learn.

5.1.3 Elementary L-System parser

Modelling in this framework did not prove very practical at all. The relationship between the modules and the resulting plant is not intuitive and the inability to deal with the overall plant structure makes modelling difficult, even for experienced modellers.

5.1.4 Billboarding

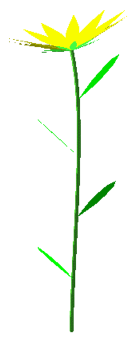
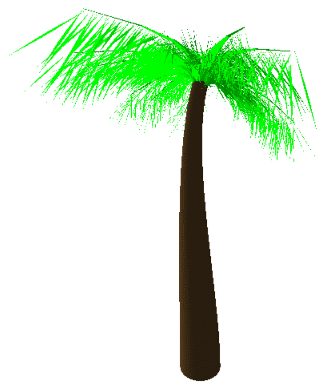
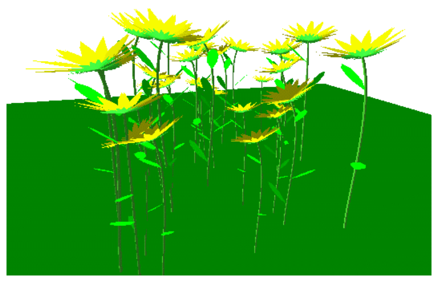
Billboarding cannot really be discussed in terms of practicality. The creation of the image can be accomplished using a variety of drawing software, which is not related to this thesis.

5.2 Realism

In this section we discuss the ability of each of the system to produce realistic looking plant models. Some models from each system are given as examples.

5.2.1 Component modelling system

Some of the plants produced in this system were superb. The degree of realism obviously depends very heavily on the user, but the fact that the system is able to produce realistic plants without too much trouble shows its competency in this category. Some of the plants modelled in this system are shown below.

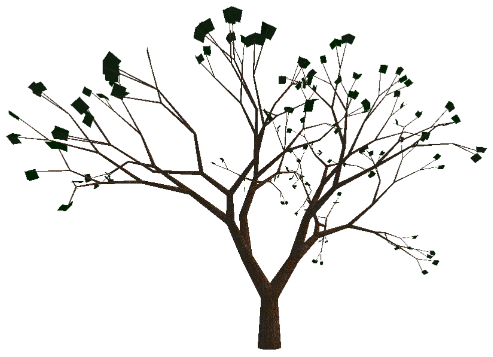
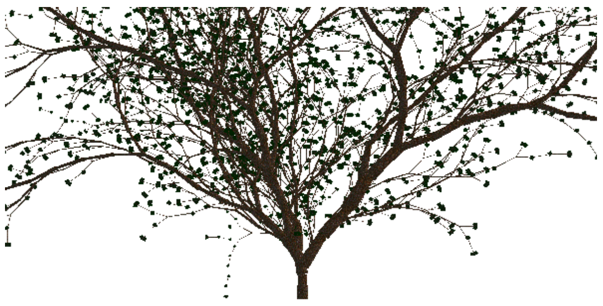
 **Figure 37**	 **Figure 38**	 **Figure 39**

5.2.2 Interactive ramification matrix modelling system

The matrices generated using the sinusoidal function introduced above do not produce the realistic models that we hoped it would. Plants modelled using the *balance*-slider are rather unnatural and have defects such as branches that extend to infinity.

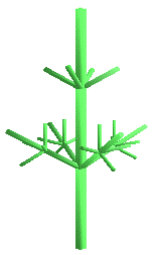
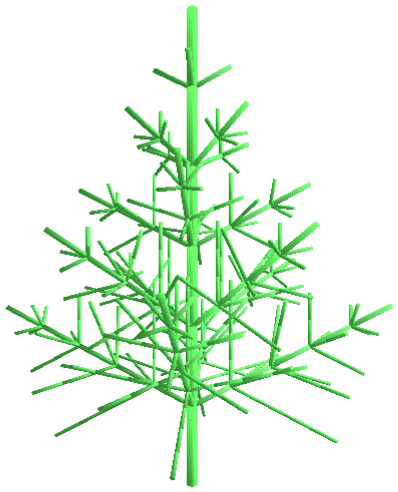
While automating the calculation of some of the plant properties allows less work to be done by the user, it severely limits the variety of plants that can be generated. It would appear that a comprise needs to be reached regarding practicality versus realism.

Interestingly, some of the best results were produced when using a perfect tree. By simply adjusting the twist-slider, some rather elegant looking trees were produced. These, and other example from this system, are shown below.

 **Figure 40**	 **Figure 41**

5.2.3 Elementary L-System parser

The models created using this technique were unsatisfactory. The limited number of modules used was one of the major factors in preventing realistic models from being generated. An example of a model generated using this system is shown below in two different stages of iteration.

 **Figure 42**	 **Figure 43**

5.2.4 Billboarding

Much of the realism of this technique is dependant on the image and stencil used. The ability to use a photograph as the texture image enables very realistic models to be generated.

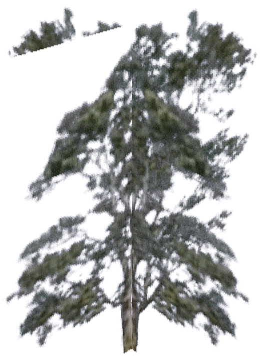


Figure 44

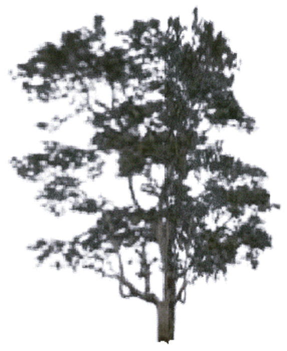


Figure 45

5.3 Rendering speed

In this section, we analyse the complexity of the models produced by each system, and thus their rendering speeds. Tables are provided for each system showing the complexity and frame-rates achieved.

5.3.1 Component modelling system

The rendering speed of the low-complexity models is excellent, while the high-complexity often generated too high a polygon count. Table 5.1 shows the frame-rates of some of the models generated. The ability of components to be assigned levels of detail proves invaluable in the speed of rendering.

Figure	Polygons	FPS
38	530	100.5

39	9000	15.8

Table 5.1 *– Complexity and rendering speed of models created using the component-modelling system.*

5.3.2 Interactive ramification matrix modelling system

The frame-rates observed when displaying trees modelled in this system were excellent, especially for the perfect trees. The relatively low complexity of the models produced ensures a low polygon count enabling high rendering speeds to be obtained. The complexity and rendering speed of the models demonstrated in section 5.2.2 are shown below.

Figure	**Polygons**	**FPS**
41	630	82.5
42	>10000	< 8

Table 5.2 *– Complexity and rendering speed of models created using the ramification matrix modelling system*

5.3.3 Elementary L-System parser

The rendering speed of the L-System models was fast only for relatively simple L-Systems. Due to the fractal nature of L-Systems, however, models rapidly become highly complex resulting in a high polygon count, and thus poor rendering speeds. The complexity and frame-rates of the models shown in section 5.2.3 are given below.

Figure	**Polygons**	**FPS**
43	510	110
44	4240	33

Table 5.3 *– Complexity and rendering speed of models created using the L-System parser*

5.3.4 Billboarding

The rendering speeds of billboards are very fast indeed. This is due to the fact that billboards are composed of a handful of polygons at most. The complexity and frame-rate of the model shown in section 5.2.4 is given below.

Figure	**Polygons**	**FPS**
45	2	>150
46	2	>150

Table 5.4 *– Complexity and rendering speed of models created using double billboarding techniques*

5.4 Summary

The results of this thesis show that the component modelling system is ideal for the creation of realistic models of any plant-type. The ramification matrix system also produced some realistic output, but the models are really limited to trees. These two systems are therefore the systems to be used if realism is the most important factor. The billboard system, as we expected, produced superbly high rendering speeds, but did not produce particularly believable models. The L-System modeller was somewhat disappointing. Further enhancements could be made to improve it, but the impractical nature of L-System modelling makes the other modelling systems a first choice.

6 Conclusion

This thesis identifies the three most important aspects of plant modelling: practicality of modelling, realism and rendering speed (complexity) of the models produced. We design and implement a set of modelling systems, each focusing on one of the three aspects mentioned above, so as to get an idea of the relationship between each. We find that, to a large extend, the three main aspects of plant modelling are mutually exclusive – a modelling system that is optimised for speed of rendering is likely to produce somewhat less realistic models than a system that focuses primarily on realism with little regard to the complexity of model produced.

Of all the systems implemented in this thesis, the component modelling system is the only one that attempts to fulfil all the above modelling criteria, and it does so fairly effectively. This system, it can be seen from some of the results produced, provides an excellent way of modelling plants, and with the development of a graphical user interface it could be even more phenomenal.

6.1 Future work

Further work could be done into the design of graphical user interfaces for the development of plant models. It became apparent, during the course of this project, that while the manual specification of plant attributes allows for more accurate modelling, the ease of modelling created through the inclusion of a graphical user interface is preferable.

7 References

[Internet cpsc]	Przemyslaw Prusinkiewicz, Mark Hammel, Radomír Měch, “Visual Models of Morphogenesis: A Guided Tour”, http://www.cpsc.ucalgary.ca/projects/bmv/vmm/
[Jakulin 00]	Aleks Jakulin, “Interactive Vegetation Rendering with Slicing and Blending”, Faculty of Computer and Information Science, University of Ljubljana, Slovenia, 2000
[JirPru]	Catherine Jirasek, Przemyslaw Prusinkiewicz, “A Biomechanical Model of Branch Shape in Plants”, Department of Computer Science, University of Calgary, Canada
[Kovács]	Ladislav Kovács, “Simultaneous Effects of the Environment and the Tropism to the Growth of a Tree”, Department of Computer Graphics and Image Processing, Comenius University, Slovakia
[LinDeu 98]	Bernd Lintermann, Oliver Deussen, “A Modelling Method and User Interface for Creating Plants”, Blackwell Publishers, Oxford, 1998
[Oxford 92]	The Oxford English Dictionary of Current English, second edition, Oxford University Press, 1992
[PruHam 95]	Przemyslaw Prusinkiewicz, Mark Hammel, Radomír Měch, “The Artificial Life of Plants”, Department of Computer Science, University of Calgary, 1995
[PruMün 01]	Przemyslaw Prusinkiewicz, Lars Mündermann, Radoslaw Karwowski, Brendan Lane, “The use of positional information in the modelling of plants”, Department of Computer Science, University of Calgary, Canada, 2001