

**Biologically Inspired
Plant Modelling for Synthetic Imagery**

by

Aidan Lane



Thesis

Submitted by Aidan Lane

in partial fulfillment of the Requirements for the Degree of
Bachelor of Computer Science with Honours (1608)

Supervisor: Jon McCormack

**School of Computer Science and Software Engineering
Monash University**

November, 2004

© Copyright

by

Aidan Lane

2004

Contents

List of Figures	v
Abstract	vii
1 Introduction	1
2 Plant Modelling	3
2.1 L-systems	3
2.2 Evolution-Based Approaches	4
2.2.1 Aesthetic Evolution	5
2.2.2 Autonomous Evolution	6
2.2.3 Hybrid Evolution	7
2.2.4 Summary	7
2.3 Data-Oriented Approaches	8
2.3.1 Geometric Parameters	8
2.3.2 Observational Data	9
2.4 Graph-Oriented Approaches	10
2.4.1 L-system Representations	10
2.4.2 P-Graphs / XFROG	11
2.4.3 Generic Directed Graphs	12
2.5 Summary	13
3 Methodology	15
3.1 Objectives	15
3.2 Modelling Method	16
3.2.1 Branching Points	16
3.2.2 Plant Topology	17
3.3 Summary	19
4 System Design	21
4.1 Modelling Process	21
4.2 User Interface	21
4.2.1 Project Palette	22
4.2.2 Branching Point Editor	23
4.2.3 Plant Topology Editor	24
4.2.4 Node Inspector	25
4.2.5 Arc Editor	27
4.2.6 Material Editor	28
4.2.7 Project Preview	30

5	System Implementation	31
5.1	Source Code	31
5.2	Libraries Used	32
5.3	Limitations	32
5.4	Future Extensions	32
5.4.1	Branch Contours and Curves	32
5.4.2	Animation	32
5.4.3	Scripting	33
6	Results and Discussion	35
6.1	Arc Conditions	35
6.2	Branching Point Variations	36
6.3	Model Comparisons	37
6.3.1	L-systems	37
6.3.2	XFROG	38
6.4	Summary	38
7	Conclusion	41
7.1	Review	41
7.2	Future Research	41
	Appendix A Revised Specification of Deliverables	43
	Appendix B Clarification of Original Contribution	45
	Appendix C Branching Point Variation Algorithm	47
	Appendix D Class Hierarchy	49

List of Figures

2.1	Turtle interpretation of a simple DOL-system grammar: (a) The grammar; (b) iteration 0; (c) iteration 1; (d) iteration 2; (e) iteration 3.	3
2.2	Curry's user interface for evolution experiments.	6
2.3	One of Jacob's plant- <i>like</i> L-systems.	7
2.4	An example of a junction (or branching) point. (Zhi et al.; 2001)	9
2.5	A green ash tree, represented as an L-system, renderings of three stages of the simulated development and the corresponding Petri net. (Prusinkiewicz and Remphrey; 2000)	10
2.6	Modelling a <i>rhododendron</i> : (a) Leaf with texture; (b) a tiny twig with blossom; (c) a main twig that branches in leaves and in two tiny twigs; (d) the whole bush, and (e) p-graphs of the <i>rhododendron</i> (the numbers of the dashed regions indicate the figures corresponding to the subgraphs). (Lintermann and Deussen; 1999)	11
2.7	Designed examples of genotype graphs and corresponding creature morphologies. (Sims; 1994)	12
3.1	A branching point that could be represented in this method.	17
3.2	A plant topology that could be described using this method.	19
4.1	The three main dialogs: (a) Plant topology editor; (b) project palette; (c) node inspector.	22
4.2	The project palette dialog.	22
4.3	The branching point editor dialog.	23
4.4	The topology editor dialog.	24
4.5	The node inspector dialog.	25
4.6	The branching arcs panel.	26
4.7	The branching points panel.	26
4.8	The arc editor dialog.	28
4.9	The material editor dialog.	29
4.10	The project preview dialog (expanded view).	30
5.1	About EverGreen, the system implementation.	31
6.1	Arc conditions: (a) Original tree; (b) branching limited to when the diameter is greater than 10mm; (c) > 20mm; (d) > 30mm; (e) > 40mm; (f) the topology; (g) the branching point.	35
6.2	Branching point variations. The original tree is located in the centre, with variants surrounding it, along with the random seed used to create them.	36
6.3	For the example shown in Figure 6.2: (a) the topology; (b) the branching point used.	37
6.4	L-System: (a) Geometric model; (b) plant grammar. (see Prusinkiewicz and Lindenmayer; 1990, page 26)	38

6.5	EverGreen: (a) Geometric model; (b) plant topology and branching points.	38
6.6	XFROG: (a) Geometric model; (b) plant structure.	39
6.7	EverGreen: (a) Geometric model; (b) plant topology and branching points.	39

Biologically Inspired Plant Modelling for Synthetic Imagery

Aidan Lane
alane@csse.monash.edu.au
Monash University, 2004

Supervisor: Jon McCormack
jonmc@csse.monash.edu.au

Abstract

The desire to create models of plants has increasingly become a cross-disciplinary process, shared by artists, designers, biologists and computer scientists. While various methods exist to assist with this modelling, they have been overly complex to use or inaccurate in their results. Current methods include L-systems and applications of evolution-based computation and visual graph-based languages. Thus, the aim of this study is to develop a new method that focuses on the way that branching structures would be observed in nature. In addition, a new method must support visualisation of the resulting model, in order to produce the synthetic imagery.

This thesis presents a new, biologically inspired method for modelling plants. It allows plant architectures to be described concisely, using existing concepts, such as *branching point* types and a graph that is used to specify the topology. This form of model enables realistic branching structures to be modelled and should appeal to a broader audience of modellers, in comparison to previous systems. Furthermore, these branching structures can include stochastic variation of individual components, which enables entire forests of plants to be created from a single model. The implementation allows the models to be exported into a geometric form, suitable for use in professional rendering packages.

Biologically Inspired Plant Modelling for Synthetic Imagery

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Aidan Lane
November 3, 2004

Chapter 1

Introduction

Plant modelling is an activity undertaken across multiple disciplines. These would include artists, designers, biologists and computer scientists, all of whom are interested in creating both factual and fictional models of plants. The types of plants in this context includes woody and herbaceous (non-woody) types, which encapsulate trees, shrubs and flowers. We are interested in the type of models that can represent the architecture of plants, primarily their branching structure.

Of interest is the process by which plant modelling is performed on a computer with the aim of producing synthetic imagery. For this, the models must support visualisation in some form or another. Once created, the models can be used as props in films, video games, documents, and contemporary art. Traditional methods use geometric primitives as building-blocks to model natural shapes. This process is quite difficult, due to the complexity and diversity found in nature. Hence, many methods have been developed to assist in this process. Methods range from L-systems, which allow the modeller to write in formal grammars (Lindenmayer; 1968), applications of evolution-based computation (Jacob; 1994) (Ochoa; 1998) (Curry; 1999), those that use geometric parameters derived from plant metrics (Honda; 1971) (Weber and Penn; 1995) through to the use of graphs (Deussen and Lintermann; 1997).

Although each of the aforementioned methods have been successful in producing images that at least resemble plant-*like* structures, they are either overly complex to use or do not generate visually realistic results. Indeed, creating a model of a plant on a computer that is visually realistic in terms of its structure, given observations or prior knowledge, is still a difficult process, restricted primarily to experts. Hence, these factors have reduced the practicality of this activity.

This thesis investigates and proposes a new method for modelling plants, which can be used to create images that contain visually realistic branching structures. Furthermore, the proposed method should help make the activity accessible to a broader audience, particularly non-experts. Existing research and methods that relate to plant modelling are discussed in Chapter 2. Chapter 3 presents the new method, based on research of the previous chapter. Following this, Chapter 4 details the system design, with special attention given to the user interface. System implementation details are listed in Chapter 5. Results and a discussion of them will be presented in Chapter 6, followed by conclusions and further research in Chapter 7.

Chapter 2

Plant Modelling

In this chapter, existing research and methods that have provided assistance to process of modelling plants will be discussed. This will assist in outlining common and possibly essential properties of the existing methods, while also exposing gaps. This information is used to form a new method, as presented in Chapter 3.

2.1 L-systems

Aristid Lindenmayer (1968) devised the Lindenmayer-system (L-system) as a mathematical formalism for modelling and simulating the development of multicellular organisms. This form of parallel rewrite grammar has been put to use by biologists, computer scientists, artists and designers alike. The application of L-systems has primarily been targeted toward producing models of plants. The reason for this is that they can be used to generalise the topological branching structure of plants and exploit their features, such as symmetry, self-similarity, phyllotaxis¹ and much more. In this context, the term ‘plant’ refers to both woody and herbaceous types, including trees, shrubs and flowers.

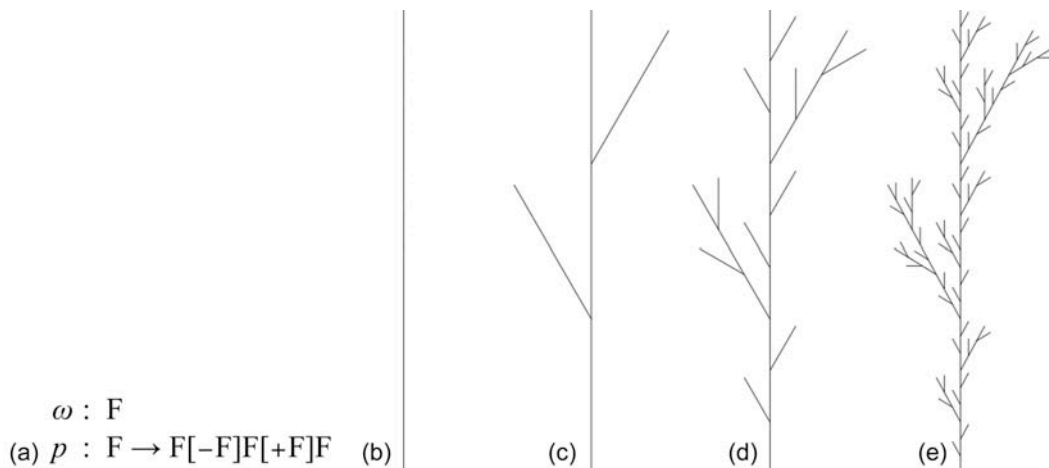


Figure 2.1: Turtle interpretation of a simple DOL-system grammar: (a) The grammar; (b) iteration 0; (c) iteration 1; (d) iteration 2; (e) iteration 3.

In his final years, Lindenmayer partnered with Prusinkiewicz, where they conducted research and published together. They focused on the generation of visually realistic plant models (Prusinkiewicz et al.; 1988) and published “The Algorithmic Beauty of Plants”

¹Phyllotaxis refers to the arrangement of plant organs on the plant body — e.g. Sunflower seeds.

(Prusinkiewicz and Lindenmayer; 1990). This work made use of many extensions to the basic L-system form. Most important to the field of plant model visualisation, was the use of turtle geometry (Papert; 1971). Figure 2.1 illustrates turtle interpretation of a simple DOL-system² grammar. The symbols used are:

- F Move forward one unit, drawing a line.
- Turn left by a predefined angle (e.g. 30 degrees).
- + Turn right by a predefined angle (e.g. 30 degrees).
- [Push the current position and orientation state onto the push-down stack.
-] Pop a position and orientation state from the stack and set it as current

Prusinkiewicz has continued research with colleges into the modelling of plants. This includes: smooth animation of plant development, improving and extending upon the base L-system language constructs, and also developed methods to help increase visual realism (Prusinkiewicz et al.; 1993) (Prusinkiewicz et al.; 2000) (Prusinkiewicz et al.; 2001). Recently, Karwowski and Prusinkiewicz (2003) introduced a new language, known as L+C. Based on the C++ programming language, L+C is more flexible than L-systems, in terms of how a grammar may be structured and encoded. A change of language also provided an opportunity to include some of the many L-system extensions as core components, for example, the parametric³ form. However, L+C is still largely based on L-systems, thus, the plant modelling process has on the whole stayed the same.

Ironically, due to the intended flexible and recursive manner of L-systems (and its derivatives), they can easily become cryptic and the result of small changes to a grammar may not be obvious. Hence, manual construction of L-systems has proven to be very tedious, even for experts. In particular, is the case of writing grammars to produce *realistic* geometric models of plants. Therefore, it would be preferable to have a method to assist with this task. Lindenmayer and Prusinkiewicz acknowledged this and drew attention to the *inference problem* in relation to L-systems and the difficulty in such an activity, noting the need for further research into this area (see Prusinkiewicz and Lindenmayer; 1990, page 11).

In the following sections, methods that have provided assistance to the plant modelling process will be covered. To aid analysis, the methods have been divided up into three broad categories; evolution-based, data-oriented and graph-oriented. However, these titles are not known to be part of any formal terminology.

2.2 Evolution-Based Approaches

Evolutionary methods, such as genetic algorithms (GAs), have long been used to search for solutions to a wide variety of problems (Holland; 1975) (Mitchell; 1996). GAs have been applied to tasks ranging from the optimal arrangement of gas-pipe networks (Goldberg; 1989), to evolving virtual creatures that can swim, jump and walk in three-dimensional worlds (Sims; 1994). Hence, it was inevitable that evolutionary approaches be used to help overcome the complexities of plant modelling. Thus, they have formed the basis of many current methods. At a lower-level of abstraction, the methods that have employed evolutionary techniques could be divided into aesthetic⁴ and autonomous approaches.

²Deterministic with no context.

³Essentially, parametric L-systems provide control constructs that can be used to dictate how and when production-rules are applied.

⁴Also known as interactive or artificial, although aesthetic is commonly used when dealing with graphics.

Both of these approaches and examples of the methods they encapsulate will be covered below. Following them, a hybrid method will be discussed.

2.2.1 Aesthetic Evolution

The practical demonstrations of interactive evolutionary methods, such as those by Dawkins (1986), sparked an interest in aesthetic evolution as a novel search technique. Although Dawkins' work was not specifically designed for 'aesthetic' purposes, the principles can be used to do so. For artists and designers, it offers the potential to aid the generation of complex forms. Thus, aesthetic evolution could be used as the basis of a plant modelling method.

The first published example of aesthetic evolution with L-systems was by McCormack (1993), where it was used for a practical application. Using this interactive approach, the modeller had a large amount of control over the resulting phenotype, which ensured that it was both novel and aesthetically pleasing. Thus, the method enabled a form of creative (or artistic) exploration. Although this implementation did apply aesthetic evolution to L-systems and plant models did feature in it, it was not designed to be a solution to our problem of making plant modelling easier. Therefore, as McCormack's technique did not place an emphasis on ease-of-use, it could safely make the requirement of pre-constructed L-system germinal genotypes⁵ prior to generation. These germinal L-systems were used to ensure an interesting path of exploration. Although this requirement restricts the accessibility of this method, there are other and possibly more critical limitations, which are inherent to this evolutionary approach. This includes such things as the need for constant human input at each generative step, the necessary capping of the population size⁶ and the fact that the user can have trouble remembering the offerings of previous generations, (especially after many cycles), all of which impair the selection process. Together, these latter problems would not only have made the process quite time consuming, but also have limited the depth of exploration. Therefore, although McCormack's technique was successful given its intended purpose, it is insufficient for our needs (specifically, it would be overwhelming for non-experts).

Curry (1999) extended McCormack's theme of aesthetic evolution with L-systems. The main difference was that the germinal genotype was encapsulated, with the L-system grammar itself remaining static throughout the process of evolution. Instead of mutating the grammar, Curry applied mutation and crossover operations to seven floating-point parameters. These parameters were derived from Honda's criteria, however, only partial satisfaction (of the criteria) was provided (Honda; 1971). Surprisingly, this method was able to produce a relatively large variety of branching structures (Figure 2.2). Thus, the method enabled users with no experience with (or even knowledge of) L-systems to generate models of plants. However, this only solved the requirement of a user-specified germinal L-system, as opposed to McCormack's method. Without this requirement and without the ability to mutate the encapsulated L-system, the search space is dramatically reduced. It also removes control of the initial path of exploration. Since this method uses aesthetic evolution, it still faces the other limitations as noted above for McCormack's work. On the whole, Curry definitely made progress in making plant modelling (with L-systems) more accessible.

⁵This is the first parent from which mutants are generated.

⁶The set of choices for each generation should be kept around 16-24, otherwise the user can have trouble remembering differences between them and the screen will become very cluttered.

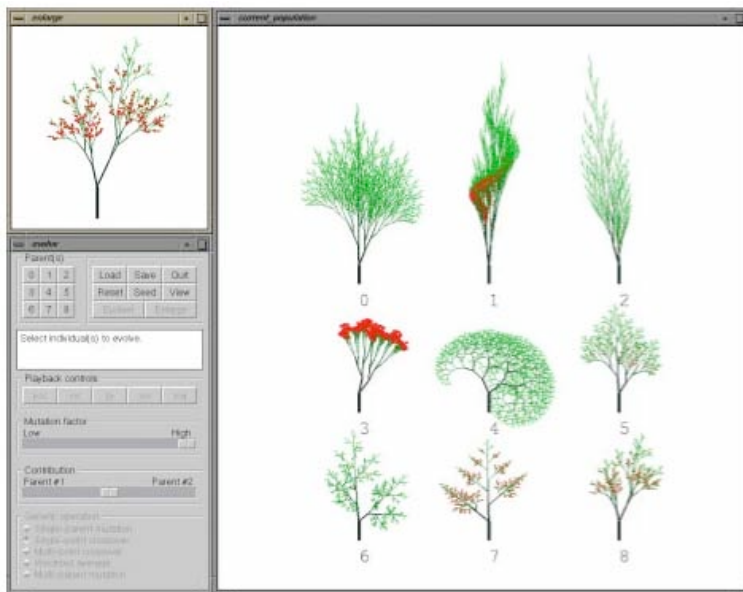


Figure 2.2: Curry's user interface for evolution experiments.

2.2.2 Autonomous Evolution

In contrast to aesthetic evolution, an autonomous system uses a fitness function, which guides the evolutionary process automatically, without the need for human intervention. The fitness function is typically a composition of set of goals or rules, which provides a quantitative *fitness* value for a given genotype (or *solution*) in a given problem domain.

Jacob applied Koza's Genetic Programming technique (Jacob; 1994) to the problem of constructing L-systems autonomously (Koza; 1992). He proposed the Genetic L-system Programming (GLP) paradigm, in which higher-order building blocks, known as *patterns*, are used for L-system generation and modification. A hierarchy of patterns provides many benefits. For instance, they enable the generation of complex grammars more easily and more efficiently, in comparison to working directly with the symbols. In addition, they can also be used to enforce fundamental rules (or laws) for both L-systems (such as prevention of unmatched/dangling brackets⁷) and the specific problem domain (such as connectivity in trees). This type of enforcement provides a streamline and elegant solution to the potential problems that may arise during the evolutionary process. For instance, after either crossover and/or mutation is applied, a grammar can become syntactically incorrect⁸ or even totally destroyed. The example described in Jacob's paper was relatively simple, given the power of GLP. The fitness function used in it was designed to simply evolve an L-system that would fill the space of a cube, yet discourage leaves from extending past a larger enveloping one. However, this was simply a proof of concept for a very promising method for applying GA operators to L-systems.

Jacob continued to build on his previous work with the application of breeding and evolving artificial flowers (Jacob; 1995). This work is quite specific, where it uses templates (possibly re-encodings of existing L-systems) and a fitness function to generate variations of a particular species of flower. His implementation has a deep reliance on the Mathematica application (Wolfram Research, Inc.; 2004), which would be overwhelming for non-experts, placing it out of reach for most artists and designers. Mathematica's graphic capabilities are also quite limited. Furthermore, from a technical standpoint, Jacob's results have been

⁷ Example of an unmatched (or dangling) bracket: $[+F] - F]$

⁸ Although, this can be also be avoided with appropriate crossover and mutation rules.

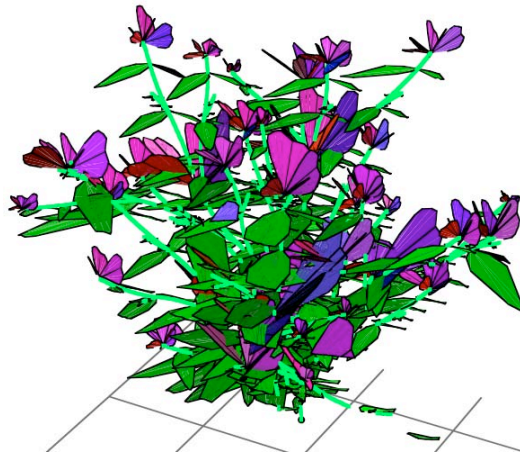


Figure 2.3: One of Jacob’s plant-*like* L-systems.

limited to plant-*like* L-systems only (Figure 2.3). By the term ‘plant-*like*’, it is implied that they are general illustrations of plants, nothing specific nor realistic. This highlights the problem with autonomous solutions, in that it is very difficult to describe and ultimately write a fitness function that will produce a model for a *real* plant. Hence, in the next section (2.3), the topic of plant feature extraction will be investigated.

2.2.3 Hybrid Evolution

Ochoa (1998) acknowledged that both of the above-mentioned approaches to evolutionary computation had significant merits. On one hand there is aesthetic evolution, which enables the modeller to dictate what the resulting phenotype will be with a great deal of ease, while placing no real requirement on them to understand the underlying plant model. On the other hand, there is autonomous evolution, which can search using a large population size and can do so in a short period of time. Just as important is the fact that the latter method can explicitly ensure that successful phenotype models will be appropriate to the given problem domain. Of course, this argument assumes a suitable fitness function. Therefore, Ochoa devised a method that allowed the modeller to make decisions based on aesthetics, while also enforcing that predefined fundamental rules were obeyed. This was to ensure that only *relevant* phenotype models were available for selection. For example, she used features such as bilateral symmetry, light gathering ability, structural stability and the proportion of branching points. However, the overall problem with this method, is that like Jacob’s work, it can only produce plant-*like* L-systems. Hence, the modelling of real plants using real observational data or expert knowledge (of plants, in general) was still difficult.

2.2.4 Summary

Recently, McCormack (2004) reviewed the developments of generating and mutating L-system grammars with evolutionary processes. He concluded that “Aesthetic evolution is yet to find a worthy successor”. For he notes that a lot of work still needs to be done in the research and development of autonomous L-system generators, of which can satisfy the requirements for practical use by artists and designers.

As noted in the discussion of Jacob's work with autonomous systems, plant feature extraction is of paramount importance to ensuring the generation of more realistic plant models. Therefore, in the following section, existing research into this area will be discussed.

2.3 Data-Oriented Approaches

As discussed in the previous section, interactive generation of plant models has been popular among artists and designers. This was primarily due to the fact that they provide the modeller with a large amount of control over the resulting phenotype and typically do so via an intuitive graphical interface. The inherent problems with this form of method was the potentially time-wasting 'generate-pick-generate...' cycle and the reduction of the *plant-space*. Thus, 'data-oriented' methods have been developed. These methods, (which could also be labelled 'parameter-oriented') keep the modeller in charge of the modelling process. They can potentially employ autonomous evolutionary techniques, as to search a for plausible solution for a given set of data. The data could be virtually any form of plant description. However, for the purposes of practicality, the use of geometric parameters, observational data or expert knowledge (of biology) appear to be essential. Therefore, the following (sub-)sections contain analysis of existing methods and further research that fall into the categories for the first two: geometric parameters and observational data.

2.3.1 Geometric Parameters

As discussed in section 2.2.1, Curry (1999) created an interactive method for plant modelling based upon the work of Honda (1971). Hence, the trees in Figure 2.2 provide insight into the types of trees that Honda's method supports. However, used directly, Honda's method is actually data-oriented. This method provides for far simpler modelling of plants, especially in comparison to L-systems. It focus on the few parameters that have the greatest effect on the form of the tree body. This meant that a novice could model a tree, simply using a handful of parameters such as branching angles and relative ratios of branch lengths. The parameter values could be derived from real trees, however they could also be imaginary. The simplicity of this method is owed to its many assumptions. For example, all branches are assumed to be straight and all branching is applied to all terminal points concurrently. In addition, each branching point produces exactly two daughter branches and all branching points use the same angles and branch length reduction values. Another problem is that, like L-systems, the resulting form produced by modifications to the parameters may not be obvious. Therefore, although Honda's method provides ease of use, it is inadequate for practical purposes.

Weber and Penn (1995) continued many of Honda's themes, with the use of geometric parameters to describe the form of a tree body and emphasised the important role of ease-of-use. In doing so, it also forfeited biological correctness, which was fine given that it was designed solely for graphic purposes. However, the method uses a larger and more formal set of parameters than those used by Honda. The method also demonstrated the need and merits of positional information and post-processing operations. This includes: stem (or branch) shapes, leaf orientation (toward sunlight) and pruning. Although the method promotes user-friendliness and does not require a background in biology, the required list of parameters can still be overwhelming. Lastly, the parameter values are provided to the system as lists of textual data, hence there is no visual feedback to their modification, which could slow the modelling process.

2.3.2 Observational Data

Runqiang et al. (2002) addressed the topic of how to automate the derivation of L-systems that represent branching structures using genetic algorithms, given real observational data. The observational data was defined in terms of an axial-tree database, which contained fields such as position, length, angle and sub-axes. Hence, a fitness function could be implemented, which evaluated the L-system genotypes by comparing the branching structure they generated with observational data. The smaller the distance between these, the greater fitness. This function was combined with the usual genetic algorithm operators of crossover, mutation and selection along with a special symbol moving operator. Together, these operators were used to steer the search for an L-system that best fits the observational data. Given their proposed method and observational data, the researchers successfully found ‘optimal’ L-systems. These models were then rendered (in two-dimensions), for which could then be compared visually in an analytical manner. Runqiang, et al. proved that L-systems can indeed be derived using an autonomous system, given a real database, not just generic rules that for example have been used in the past to generate plant-*like* models. Although this study was only used to produce simple DOL-systems, the method opens the door for solutions to the same problem for realistic models of more complex biological objects, using parametric, context-sensitive and stochastic L-systems.

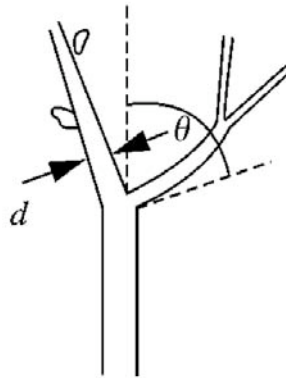


Figure 2.4: An example of a junction (or branching) point. (Zhi et al.; 2001)

The concept of a junction (or branching) point is far from new to the fields of art and science (Da Vinci; 1970)⁹. Unfortunately, junction points do not appear to have been exploited and used as a primary element of any formal modelling method. Zhi et al. (2001) addressed the topic of describing branching structures of plants. In particular, they focus on which of the four previously proposed (in biology) ‘cost’ models; minimising total surface area, total volume, total drag and total power losses at a junction point best fits observational data of branching structures of plants (see Figure 2.4). The approach that they took to answer the above question was to measure branching angles and diameters of 243 junctions from four species of plants. From this data, log-log plots were created for each cost model, where the expected cost was plotted against the observed cost. One sided t-tests were also used to compare the fitness of the four models. Both the log-log plots and the t-tests indicated that the volume minimisation model fits the data better than the other cost models. This indication most prominently seen in the expected cost vs. observed cost plot, where the data was almost an exact mapping. They have demonstrated that the volume minimisation model is the better of the four models. The authors noted that this

⁹This is a reprint date of Leonardo Da Vinci’s work on painting, originating from the turn of the 16th century.

outcome should also spark further research based on this knowledge. For example, the model could be extended or combined with other new or existing models, as to derive more advanced models that support other factors that effect the branching structure, such as the effect of light (phototropism). Therefore, it is plausible that this concept of a junction point be employed as part of a formal modelling method. In this case, a modeller could take measurements of a real plant, which could then be fed into a system as a form of observational data.

2.4 Graph-Oriented Approaches

This section discusses a number of plant modelling methods that use a graph representation as their core construct. Formed from nodes and arcs, graphs can provide the modeller with a visual design interface. L-systems have met many graph-style re-representations and will be treated first. These will be followed by a method that uses a ‘p-graph’ to represent the plant structure and another that uses a generic directed graph.

2.4.1 L-system Representations

Along with their traditional mathematical rule style notation, L-systems can be represented as graphs. McConnell (1988) used graph grammars to model plants in a similar way to context-free L-systems, where production rules are applied in parallel. In this case, each node represented a branching point of a plant and the connecting arcs represented the stems. Production rules consisted of three parts, which contained node modifications, qualified link modifications and specifications for the creation of new nodes. Although this method provided a mechanism for constructing and analysing L-systems visually, they still work at a too low a level to be of any great benefit.

In what is known formally as “language-restricted iterated function systems” (LRIFS), Prusinkiewicz and Hammel (1994) were able to represent parametric L-systems in a form of finite state automaton (FSA). In this case, nodes are used to represent states and arcs to represent transitions. Transitions between the states map directly to compositions of rotations, translations and scalings. Using their method, they could then convert a LRIFS model into a L-system model, which could then be visualised. Although this method provides parametric capabilities over McConnell’s work, it too suffers from the same problem of working too close to L-system and its intricacies for our purposes.

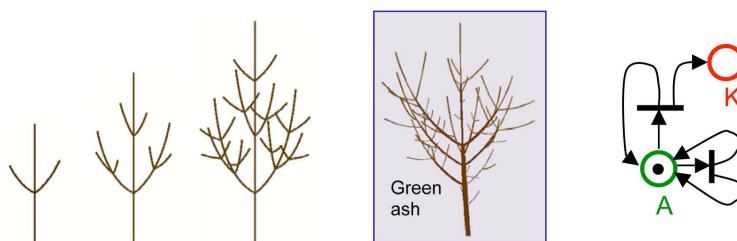


Figure 2.5: A green ash tree, represented as an L-system, renderings of three stages of the simulated development and the corresponding Petri net. (Prusinkiewicz and Remphrey; 2000)

Later, Prusinkiewicz and Remphrey (2000) used Petri nets to graphically describe architectural tree models. Petri net graphs differ quite considerably when compared to traditional graphs, such as FSAs, as used by earlier methods. In particular, a single graph can represent a plant in its entirety, including application of production rules in parallel.

This is enabled by the concept of 'tokens', which 'hop' from state¹⁰ to state along the arcs, applying production rules. The tokens can be 'cloned' (or 'split') to produce more tokens, which they themselves can branch off into different states. Each token performs one hop at a time, but does so simultaneously with all other tokens, which enables the parallelisation. Figure 2.5 illustrates an example of how Petri nets can be used for this task. Yet again, as this use of Petri nets has been directed as a representation of L-systems, it has inherited their (L-systems) above-mentioned shortcomings.

2.4.2 P-Graphs / XFROG



Figure 2.6: Modelling a *rhododendron*: (a) Leaf with texture; (b) a tiny twig with blossom; (c) a main twig that branches in leaves and in two tiny twigs; (d) the whole bush, and (e) p-graphs of the *rhododendron* (the numbers of the dashed regions indicate the figures corresponding to the subgraphs). (Lintermann and Deussen; 1999)

In comparison to the L-system representations, such as those listed above, the method proposed by Deussen and Lintermann (1997) was quite radical. The method is aimed at the artist or designer; it distanced itself from biological correctness in favour of geometry and ease-of-use (Lintermann and Deussen; 1999). In particular, it does not use a simulation of plant development. However, simplistic animations of such can be created, primarily via the use of scaling. The method uses a prototype graph (or *p-graph*) to describe the plant (see Figure 2.6). The nodes are used to represent the parts of the plants, with the arcs between them indicating creation dependencies. Although this method has greatly simplified the plant modelling process, it has its fair share of drawbacks. A general directed graph usually allows for the use of recursion, where cycles can lead from a single node directly back to itself, or it may pass through a number of other nodes first. In the latest official implementation, XFROG 4.1, the graph concept seems to be reduced down to a tree (Lintermann and Deussen; 2004). This removes the ability for recursive (or fractal) modelling, a property that had gained much popularity among L-system modellers. To model more complex plants, especially those with different types of variations within their structures, the intuitiveness of the simple p-graph and its components breaks down. Since the system is not very extensible, the modeller is sometimes left to use 'ugly' and crude 'hacks', such as putting detail into textures instead of forming geometry, in order to complete their work. Thus, the expressiveness of the method is questionable.

Genotype: directed graph. **Phenotype:** hierarchy of 3D parts.

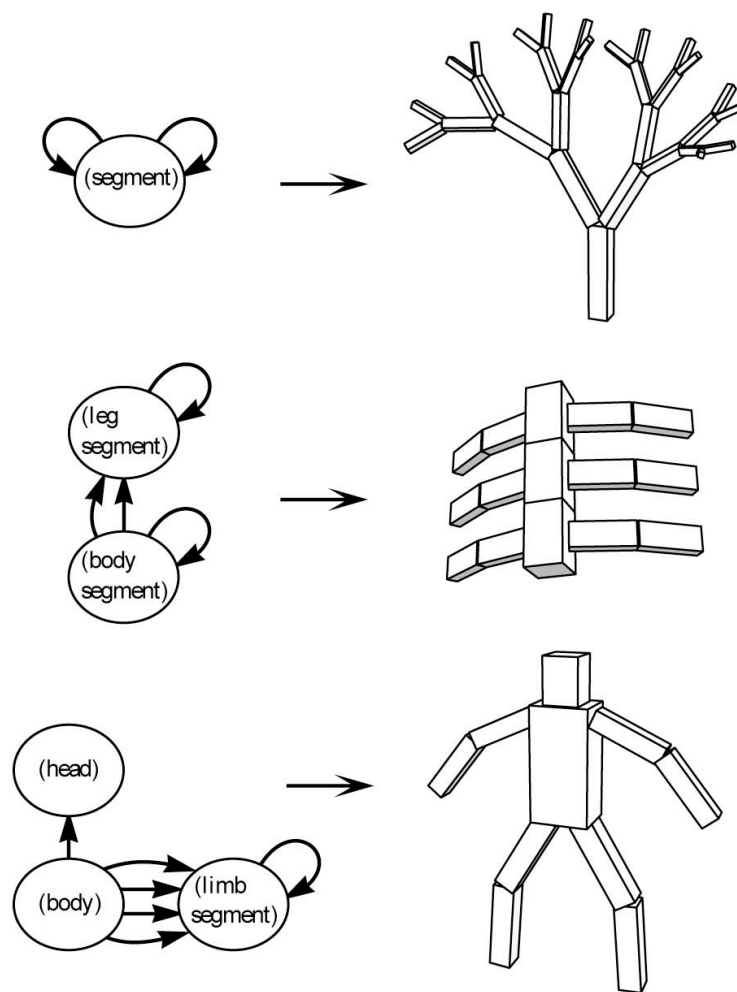


Figure 2.7: Designed examples of genotype graphs and corresponding creature morphologies. (Sims; 1994)

2.4.3 Generic Directed Graphs

As noted in section 2.2, Sims evolved virtual creatures that could swim, jump and walk in three-dimensional worlds (Sims; 1994). Although the creatures were ‘evolved’, they had to abide by predefined architectures. These architectures could be prescribed via generic directed graphs (see figure 2.7). The nodes represent components of the creature (drawn as boxes) and the arcs represent a form of branching. Using this simple framework, potentially complex creatures could be evolved, which could exhibit recursive (or fractal) features. Although this was not used for the modelling of trees, it provides the potential of doing so, as illustrated in the top example in the illustration. Nodes could represent branches and their associated properties, and arcs could again represent branching. Hence, this simple yet potentially powerful method for architectural design, could possibly be used as part of a new plant modelling method.

¹⁰The ‘states’ are represented by nodes in the graph.

2.5 Summary

Still to this day, plant modelling is difficult. Existing methods either offer flexibility and extensibility *or* ease-of-use (exclusively). As outlined in the introduction (section 1), this project aims to research and design a new modelling method. In particular, the resulting models of interest here will be those that can be used for the synthesis of tree and plant imagery.

The resulting model for the new method need not be in the form of an L-system. Instead, a specialised data-structure could be incorporated, given the method's specifics. However, it should be possible to generate a string of L-system symbols (or modules) from it. An L-system grammar (possibly of minimal length) that can produce this string could then be evolved. This could be the basis for another research project, for example "Evolution of Minimal Generative Grammars".

Chapter 3

Methodology

The way in which the modeller is able to describe their plant subject is perhaps the most significant factor in the success of any given method. It is important to understand that it is virtually impossible for any one method to satisfy all audiences. Biologists tend to emphasise biological correctness, in comparison to artists and designers, which are concerned primarily with the visual appearance. In addition, computer scientists usually prefer a formal language that provides complete control over the subject. We are investigating a method that allows realistic branching structures to be modelled that will also appeal to non-experts. Hence, the methods covered in the previous chapter (2) were assessed according to this goal.

Methods that allow a large amount of expressiveness and extensibility, such as L-systems (covered in section 2.1), have unfortunately proven to be too difficult for non-experts. On the other hand, methods that retract any form of formal language and instead rely on few and simple parameters, tend to significantly reduce the range of plants that can be modelled; this includes Honda's criteria and its derivatives (see sections 2.3.1 and 2.2.1). These are extremes of the plant modelling spectrum. In between them are mixtures and hybrids, where XFROG perhaps best fits (see section 2.4.2).

3.1 Objectives

The following list outlines the broad objectives that influenced the design of the method:

- Expressiveness: allows the modelling of a large variety of *real* branching structures.
- Maximise Reuse: allows the re-use of components, such as branches, within a plant.
- Accessibility: the method must be appeal and be accessible to non-experts.
- Artistic Exploration: support the modelling of non-realistic shapes (McCormack; 1993).

Although encapsulated by the above, the following explicitly lists important design goals:

- Have minimal, yet unambiguous concepts.
- Enable recursion, to simplify plant descriptions and promote reuse of primitives (building-blocks).
- Enable stochastic variation, which will allow an entire forest of subtly unique plants to be created from a single model and also helps to produce branching structures that appear to be more natural.

In particular, there is a large focus on the branching structure of plants, which are a prominent part of ‘woody’ plants, such as trees and shrubs. This is an area that existing methods have failed to produce realistic results for without the modeller spending much time ‘tweaking’ the model. Although constrained by explicit feature domains within a single plant variety, there is usually still an amount of variation in its components. This notion of variation is a key element in branching structures, as they are usually quite prominent in this type of plant.

3.2 Modelling Method

This section describes the components that are essential to this plant modelling method. The methods, models and theories proposed in the previous chapter (2) have provided insight into the design of a new method. Given the success of the ‘data-oriented’ and ‘graph-oriented’ methods, it is plausible that a coupling of the two could be used as the basis of a new method. This raises two questions: “what data should be used?” and “how could the data be incorporated into a graph?”. Answers to these questions will be discussed in the following sections.

The observational data provides the method with primitives (building-blocks), which describe the features of a given plant. It is important that someone without any previous experience with plant modelling, could categorically examine a plant and be able to input this information directly into an implementation of the method.

Although the term ‘observational data’ is being used, it need not actually be *real* observations of a *real* plant. Instead, the data could be prior knowledge or even imaginary, as part of artistic (or creative) exploration (of branching structures).

3.2.1 Branching Points

The notion of a *branching point* is key concept to this method (Zhi et al.; 2001). The term ‘branching point’ has been favoured over ‘junction point’, as more people tend to know what it means, without prior prompting.

In addition, the term ‘branch’ is used in a very broad sense, well beyond its definition in biology (Room et al.; 1994). It has been used to represent *any axis*¹ *including* the main stem (or trunk).

Branching point set:

$$\langle D, B_1 \dots B_n, C \rangle$$

D	Diameter of the parent branch leading into the branching point.
$B_1 \dots B_n$	Set of daughter branches, leading out of branching point.
C	Daughter branch to use to continue with when multiple branching points are placed along a given branch. This is optional.

Daughter branch set:

$$\langle DB, DA, T \rangle$$

DB	Diameter of the branch [mm].
DA	Divergence angle of the branch from its parent’s axis (0-180) [degrees].
T	Twist of the branch around its parents axis (0-360) [degrees].

¹An axis is “a sequence of units of growth in the same general direction from one (monopodial) or more (sympodial) meristems” (Room et al.; 1994)

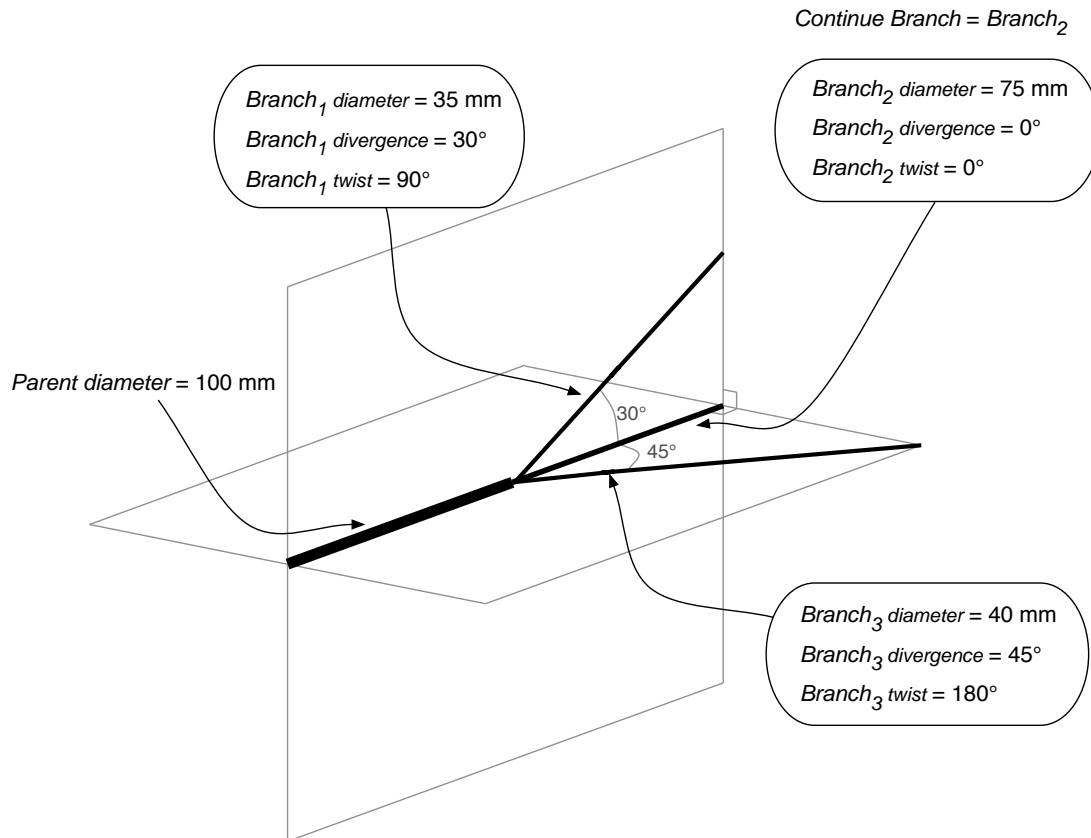


Figure 3.1: A branching point that could be represented in this method.

Figure 3.1 illustrates a branching point that could be represented using the aforementioned structures. The figure mimics the overall form used by Hatta et al. (1999) in the analysis of branching principles governing the architecture of *Cornus kousa* (cornaceae).

3.2.2 Plant Topology

The observational data, which describes the features of the plant, must be organised in such a way to describe the complete plant. Hence, the need for a plant *topology*. To facilitate this, we can use a graph. One similar to that used by Sims (1994) has been used, while incorporating some of the ideas from p-graphs (Deussen and Lintermann; 1997).

The term ‘node’ is used in this document according to its meaning in graph theory, where it is also known as a ‘vertex’. This is in comparison to biology, where it means a type of branching point in a plant (Room et al.; 1994). Given the target audience of artists and designers, this assertion should be fairly safe, especially given that in this field a ‘vertex’ represent a point in 2D or 3D space.

The data structures presented below contain more than just the topology information. However, the data structures themselves are the essential components of the topology; namely, the node and arc sets, which themselves are brought together with the plant model set.

Plant model set:

$$\langle N_1 \dots N_n, A_1 \dots A_m, BP_1 \dots BP_p, P_1 \dots P_q, SN \rangle$$

$N_1 \dots N_n$	Set of nodes, each of which represent a single branch component.
$A_1 \dots A_m$	Set of arcs, each of which represent a branching point between two nodes. They may be multiple leading out of or into any given node.
$BP_1 \dots BP_p$	Set of branching points, as described above and used by the arcs.
$P_1 \dots P_q$	Set of properties. They can be applied to nodes in order to change their visual appearance. Materials (with textures) would need to be supported and could be extended to include contours and branch curves.
SN	The state node. The node from which all branching originates.

Node set:

$$\langle T, G, S, P_1 \dots P_n, D, N \rangle$$

T	Type of the node, either a branch or an object. The latter could be used to represent leaves, complex flowers and fruit.
G	Growth of the node. For branch type node, this only affects its length, not its diameter. [0-100]%
S	Boolean value that set whether or not the G should be interpreted as being a scaling of the inherited growth. [true / false]
$P_1 \dots P_n$	Set of properties that are to be applied to the node during geometric construction. Properties will be inherited by the descendant nodes, if there are any.

Node set - branch type specifics:

D	Maximum recursive depth for when the arcs form cycles. This can be used to form a branching pattern of a specified depth.
N	Number of branching points that are to be placed along the branch.

Arc set:

$$\langle N_{from}, N_{to}, BP, C_1 \dots C_n, V \rangle$$

N_{from}	The node the arc leads from.
N_{to}	The node the arc leads to.
BP	The branching point that is applied when the arc is traversed.
$C_1 \dots C_n$	A set of conditions that must be satisfied in order to traverse the arc.
V	The maximum percentage by which the branching point can be varied. The term “variation” will be used when referring to this parameter, in comparison to “variance”, which would otherwise suggest that it is part of some probability distribution. Please see the variation algorithm in C for more details.

Figure 3.2 provides an example of a topology that could be described using this method. The graph may seem to resemble a NFA (Non-deterministic Finite state Automata). However, it is possible to have multiple branching points lead out of a node in a single instance

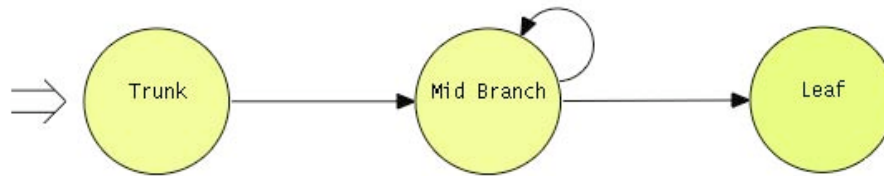


Figure 3.2: A plant topology that could be described using this method.

with this method, which cannot occur in NFAs. When node properties are incorporated, the graph also behaves similar to a *scene graph*, as used in computer graphics.

Thus, the data sets defined above provide the ability to construct *plant architectures* using branching points. Biologists should feel comfortable with this, given that they already use branching points in the description of plant structures (Hatta et al.; 1999). Furthermore, non-experts should be able to grasp the concept quite easily, given that branching points are relatively simple and readily observable in nature.

3.3 Summary

In summary, this chapter has presented the essentials of a way in which plants may be described. With this in mind, the system design will be covered in the next chapter, which builds upon this method.

Please refer to Appendix ?? for a high-level algorithm of how the plant geometry could be constructed from a model encoded using this method. Also covered is a branching point algorithm in Appendix C.

Chapter 4

System Design

With the methodology in place (Chapter 3), the implementation *design* can now be described. This chapter is primarily devoted to the design of the user interface (section 4.2), but first, the modelling process (section 4.1) will be discussed.

4.1 Modelling Process

At a high-level of abstraction, the design facilitates plant modelling for synthetic imagery via the process listed below.

Ideally, the modeller (user) will:

1. Gather observational data and the topology from the subject (plant).
2. Enter observational data into the system.
3. Enter the topology into the system, linking the observations together.
4. Make any necessary refinements to the model.
5. Have the system generate a geometric model.
6. Export the model to a file.
7. Using a software renderer, insert the model into a 3D scene, add any effects and then render it.

4.2 User Interface

The design of the GUI (Graphical User Interface) plays a critical role in how a user perceives the method, in terms of the ease-of-use and intuitiveness.

In addition to the method's own objectives (section 3.1), the following lists goals that are targeted at the GUI:

- Minimal setup: allow simple plants to be created quickly and easily.
- Visual feedback: action-and-effect, shorten the learning curve and speed exploitive modelling.
- Elegant simplicity: provide an uncluttered interface, where the user may choose to 'expand' on more advanced items.

Ideas have been taken from existing modelling packages, such as the design of the project palette (section 4.2.1), so that the primary audience of artists and designers can relate to it, as to minimise the learning curve. Furthermore, the dialogs have been designed to complement the Mac OS X interface and its native applications.

The dialog designs are presented and described in the order in which the user will typically use them, when modelling a plant. This follows the process that was previously outlined in section 4.1. The descriptions of the dialogs has been written in way that it can be used a manual. Hence, the word “you” has been used when discussing what the user can or should do.

Figure 4.1 shows the layout of the three main dialogs; the plant topology editor (section 4.2.3), the project palette (section 4.2.1) and the node inspector (section 4.2.4).

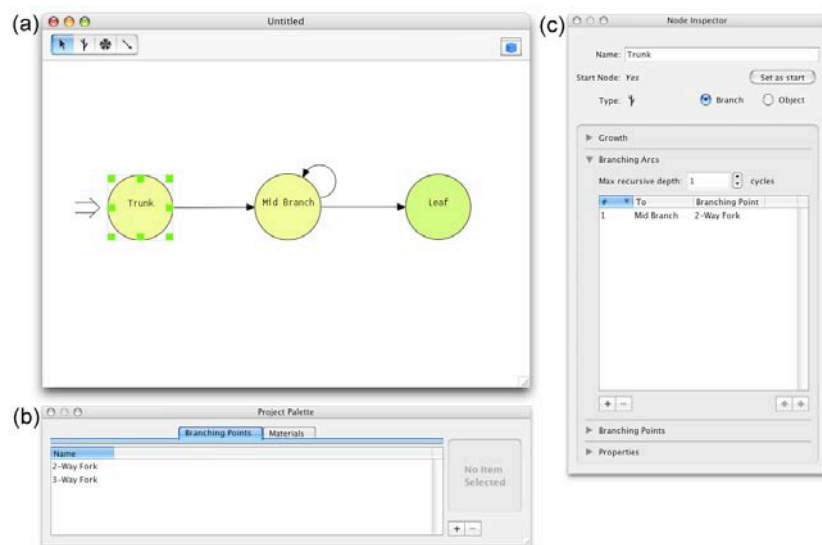


Figure 4.1: The three main dialogs: (a) Plant topology editor; (b) project palette; (c) node inspector.

4.2.1 Project Palette

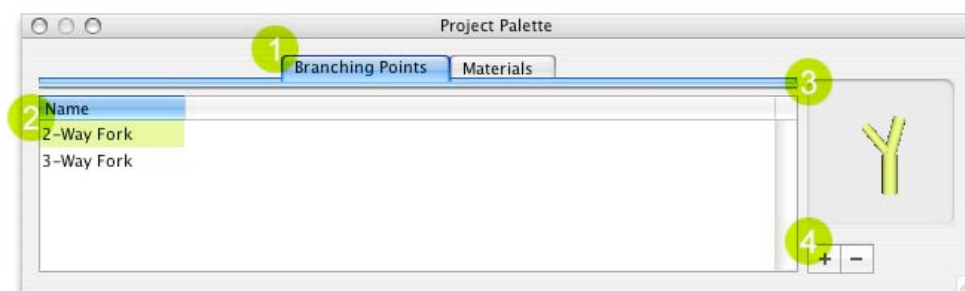


Figure 4.2: The project palette dialog.

Figure 4.2 shows a screenshot of the project palette. This is where the reusable observational data is kept. The features of the dialog have been marked numerically in the figure and align with descriptions below:

1. Click of these tabs to flip between a list of branching points and a list of materials.

2. List of either the branching points or materials, as selected by the tab bar (1). To edit an item, double click on it.
3. Item preview. If an item has been selected in (2), then a preview of it will be shown. Otherwise, “No Item Selected” will be displayed.
4. Click on these to add or remove an item (respectively). An item must be selected in list box (2) for the remove button to be active.

4.2.2 Branching Point Editor

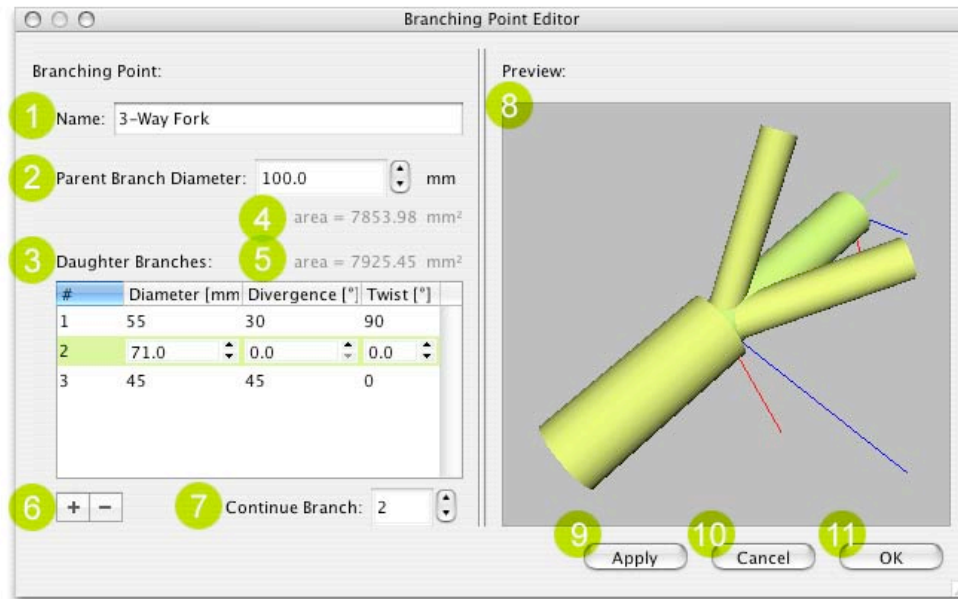


Figure 4.3: The branching point editor dialog.

Figure 4.3 shows the design of the branching point (BP) editor dialog. The dialog’s widgets have been labelled, for which their descriptions are listed below:

1. The name used to identify the particular BP. It is unique with respect to other BPs in the plant project (file).
2. Diameter of the parent branch that leads into the BP.
3. List of daughter branches. Each of the diameter, divergence and twist fields may be edited for each daughter branch item.
4. The cross-sectional area used by the end of the parent branch that is closest to the BP. This is calculated directly from the branch’s diameter. It is assumed that the area is perfectly circular. This is useful when used in conjunction with (5), given the suggestion from Da Vinci that the cross-sectional area of the parent branch is equal to the sum of the cross-sectional areas of the daughter branches (Da Vinci; 1970).
5. Sum of the cross-sectional areas of the daughter branches. This is calculated directly from their diameters. Please see (4) for more details.
6. Click these to add or remove a daughter branch (respectively). A daughter branch must be selected in list box (3) for the remove button to be active.

7. Identifies which daughter branch is to used as the *continue branch*. This is needed when multiple BPs are placed along a single branch, where one of the daughter branches will need to be aligned with it. This field is optional, hence it can be set to “none”.
8. Simple preview of the branching point. It provides an illustration of the branches, along with the diameters and angles used for them.
9. Click this to apply the changes to the plant project, without closing the dialog. This will update all sources and dialogs that refer to this BP, including the preview, if it is open.
10. Click this to close the dialog, without updating the project.
11. Click this to apply the changes (see 9) and close the dialog.

4.2.3 Plant Topology Editor

The graphical notion used here has been drawn from finite state automata theory.

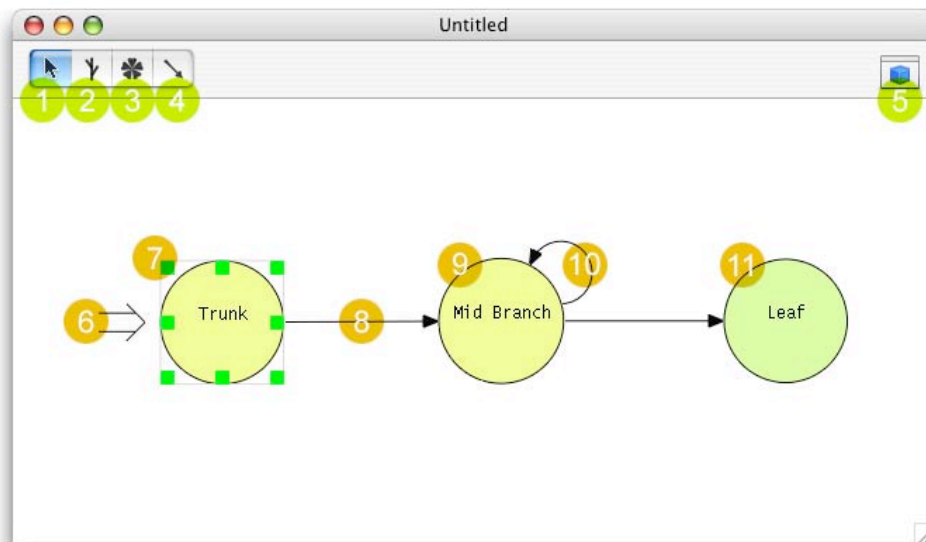


Figure 4.4: The topology editor dialog.

Figure 4.4 shows the design of the plant topology graph editor dialog. The editor’s features have been marked numerically in the figure and align with descriptions below:

1. Click to change to pointer mode, allowing nodes to be selected for editing and deletion.
2. Click to change to branch creation mode. To create a new branch node, left-click anywhere in the graph window (coloured white) to create one at that position.
3. Click to change to object creation mode. To create a new object node, left-click anywhere in the graph window (coloured white) to create one at that position.
4. Click to change to arc creation mode. Press-and-hold the left mouse button the on the *from* node and release on the *destination* node. It is safe to create an arc that links back to itself.

5. Click this to open the project preview dialog. Please see section 4.2.7 for details regarding this dialog.
6. This symbol indicates that the node to the left of it is the *start node*.
7. Node selection marker. It also allows the node to be resized (for layout purposes only).
8. An arc, which leads out of one node and into another (different) node.
9. A branch node, distinguishable given its yellow/green colour.
10. Another arc, which leads out of and then back into a single node.
11. An object node, distinguishable given its green colour.

4.2.4 Node Inspector

General

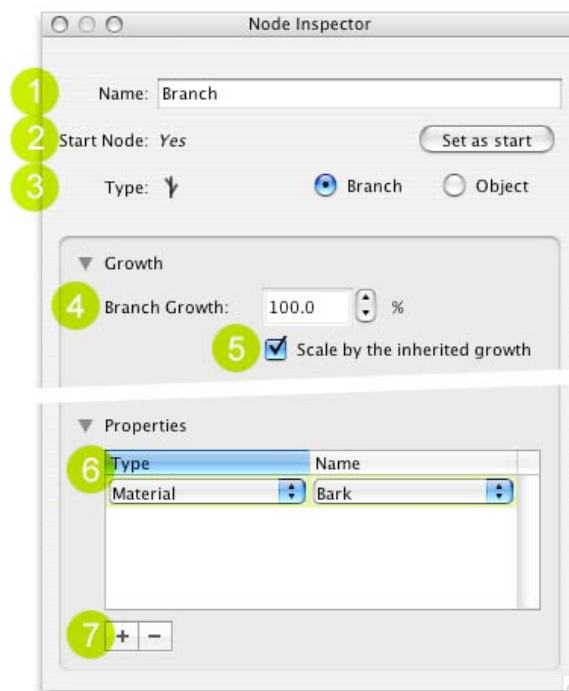


Figure 4.5: The node inspector dialog.

The contents of the node inspector dialog changes depending on the type of node being represented. Figure 4.5 shows a *sliced* view of the dialog, where only the elements common to the two types are shown.

Any changes made in this dialog are applied to the plant project immediately. Hence, this will update all sources and dialogs that refer to any this node, including the preview, if it is open.

The items indicated in the dialog are:

1. The name used to identify the particular node. It is unique with respect to the other node in the plant project (file).

2. Indicates whether or not this is the start node, either *Yes* or *No*. A start node is the node from which plant construction initiates. To make it the start node, click the “Set as start” button.
3. Indicates the type of the node, either *Branch* or *Object*. A branch is generalised to include the trunk and the roots. An object can be used to represent leaves, flowers and fruit. To change the type, click the either of the two radio-buttons. When a change of type is requested and there exists data that is specific to the old type, you will be asked if you want to continue, as it will need to be removed.
4. Growth of the node. For branches, this only affects its length, not its diameter. For objects, this affects the overall side, in all dimensions.
5. Toggle this to set whether or not the growth (4) should be interpreted as being a scaling of the inherited growth (from the node that branched into this node). For example, if it is, then with an inherited growth of 90% and its own of 80%, the actual growth would be 72%.
6. List of properties *associated* with the node. To change which property an item refers to, first select the item, choose the “Type” of the desired property and then choose the “Name” of property (which is filtered by its type).
7. Click these to add or remove a property from being *associated* with the node. A property item must be selected in list box (6) for the remove button to be active.

Branch Specific Panels

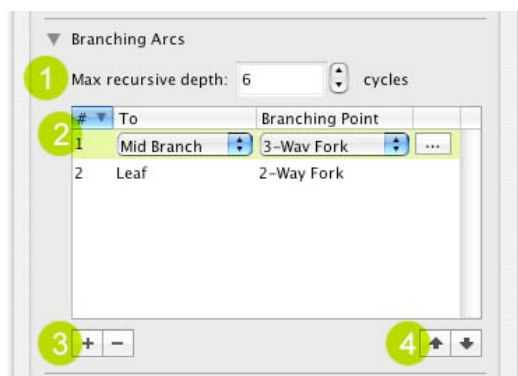


Figure 4.6: The branching arcs panel.

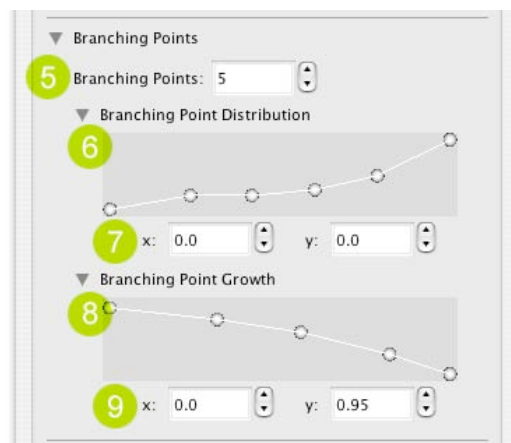


Figure 4.7: The branching points panel.

Figures 4.6 and 4.7 show the design of the branching arc and branching points panels (respectively), which are specific to branch nodes. Thus, these panels are only available in the node inspector for this node type. The point sets in the branching points panel can be categorised as *positional information*, as used in existing modelling methods (Prusinkiewicz et al.; 2001) and (Deussen and Lintermann; 1997). The features marked in the panels are as follows:

1. Maximum recursive depth for which this node can be used. This is necessary as the topology graph allows for cycles to be formed through the nodes, which could lead to infinite branching.

2. List of arcs that lead out from this node. The destination of the arc and the (default) BP (Branching Point) it uses can set directly. Although there can be multiple arcs listed here, only one can be used at a time. This is why the ordering of the arcs is important, as it sets the order in which the arcs are tried. Only when the conditions on an arc unsatisfied is the next considered. For access the rest of an arc's properties, click the button labeled “...” to open the full arc editor (section 4.2.5).
3. Click these to add or remove an arc (respectively). An arc must be selected in list box (2) for the remove button to be active.
4. Click these to change the order of the arcs, up or down (respectively). See (2) for why ordering is important.
5. Number of branching points to distribute along the branch. If the chosen BP has a *continue* set, then it will be aligned with the branch's axis. Otherwise, the BP will be positioned at the end of the branch.
6. Branching point distribution. This sets the density of BPs along the branch, and hence the way they are distributed. In the example shown, BPs will be sparse along the first half of the branch and fairly dense in the second. Values of positions between points are interpolated, allowing the distribution to be used with a variable number of BPs. To create a new point, right-click (or control key + left-click) in the graph area, followed by a left-click on “Insert point” in the pop-up menu. To remove a point, right-click (or control key + left-click) on the point, followed by a left-click on “Remove point” in the pop-up menu. You may also delete a point by first left-clicking on it (selecting it) and then pressing the delete key. The two boundary points (at 0.0 and 1.0) cannot be removed.
7. The X and Y co-ordinates of the selected point in (6). The range is [0.0, 1.0].
8. Branching point growth. This sets the growth of the BPs along the branch, and hence the growth of branches and objects that branch out of this node. In the example shown, branches and objects will have a higher growth if they branch from the first half than if they do in the second. Please see the description for (6) for editing information.
9. The X and Y co-ordinates of the selected point in (8). The range is [0.0, 1.0].

4.2.5 Arc Editor

Figure 4.8 shows the design of the arc editor dialog. The editor's features have been marked numerically in the figure and align with descriptions below.

Like the node inspector (section 4.2.4), any changes made in this dialog are applied to the plant project immediately. Hence, this will update all sources and dialogs that refer to any this arc, including the preview, if it is open.

1. The node the arc leads out from (i.e. attached to).
2. Toggle this to enable or disable conditions on the arc. The plant geometry constructor checks to see if each of an arc's conditions are satisfied before using it. If one is unsatisfied, then the arc will not be used and the next arc in the node's list will be tried (if there is one). If this is un-checked (toggled off) and there are one or more conditions associated with the arc, then you will be prompted first to see if they really do wish to remove them.

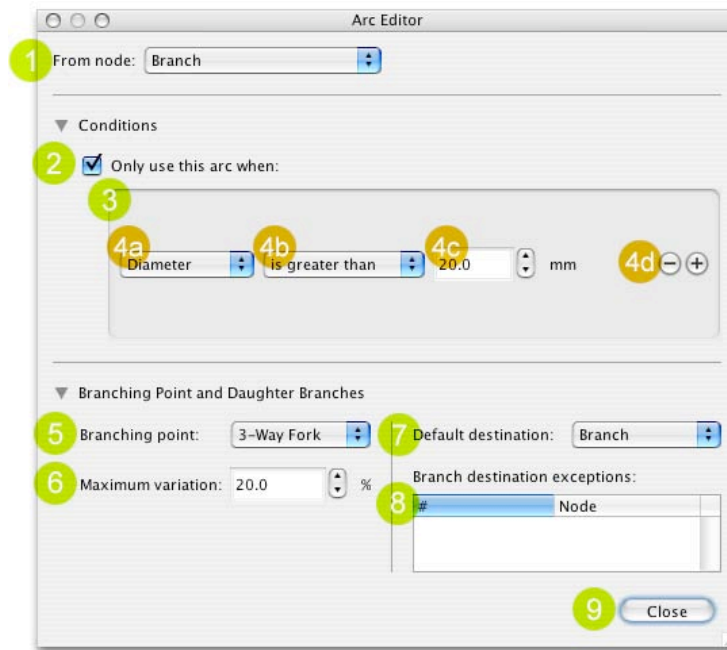


Figure 4.8: The arc editor dialog.

3. List of condition items associated with this arc.
4. An example of a condition, this one limits the use of this arc to only when the current diameter is greater than 50mm. Currently, conditions can be made based on the current *Diameter*, *Growth* or *Branch Point Number* (for when multiple branching points are placed along a branch), as selectable via combo-box (4a). The value of this field is then compared as being greater than, less than or equal (set via combo-box 4b) to a given value (in spin-box 4c). The minus and plus buttons (4d) allow conditions to be removed or new ones added (respectively).
5. Use this to choose which branching point applies when using this arc.
6. This allows for variation of the branching point, up to this specified percentage amount. The algorithm in Appendix C is used to produce the variants. Two limitations of this algorithm are that the branching point must have at least two daughter branches to work and the twist angles of daughter branches will be ignored.
7. Sets which node the daughter branches of the branching point lead to by default.
8. A list of exceptions to the default destination node (7). Exceptions can be used to have different daughter branches lead off into different nodes.
9. Click this to close the dialog.

4.2.6 Material Editor

Figure 4.9 shows the design of the material editor dialog. The dialog's widgets have been labelled, for which their descriptions are listed below:

1. The name used to identify the particular material. It is unique with respect to the other materials in the plant project (file).



Figure 4.9: The material editor dialog.

2. Ambient colour part to be used in the material. Click the coloured button to open the (system defandant) colour edit dialog.
3. Diffuse colour part to be used in the material. Click the coloured button to open the (system defandant) colour edit dialog.
4. Specular colour part to be used in the material. Click the coloured button to open the (system defandant) colour edit dialog.
5. Emissive colour part to be used in the material. Click the coloured button to open the (system defandant) colour edit dialog.
6. Preview of the material, as applied to a sphere.
7. Shininess of the material, which ranges from 0% (which gives a dim appearance), to 100% (glossy-looking surfaces). It may be adjusted via the slider or the spin-box.
8. Transparency of the material, which ranges from 0% (completely opaque) to 100% (completely transparent, i.e. invisible). It may be adjusted via the slider or the spin-box.
9. Click this to show the file open dialog, where a texture image file may be chosen.
10. This displays the texture image filename (and its path) that is to be used for the material. If it is empty, then no texture will be used. It is read-only, use (9) or (11) to change it.
11. Click this to clear the texture image filename field, removing the use of any texture by the material.

12. Click this to apply the changes to the plant project, without closing the dialog. This will update all sources and dialogs that refer to this material, including the preview, if it is open.
13. Click this to close the dialog, without updating the project.
14. Click this to apply the changes (see 12) and close the dialog.

4.2.7 Project Preview

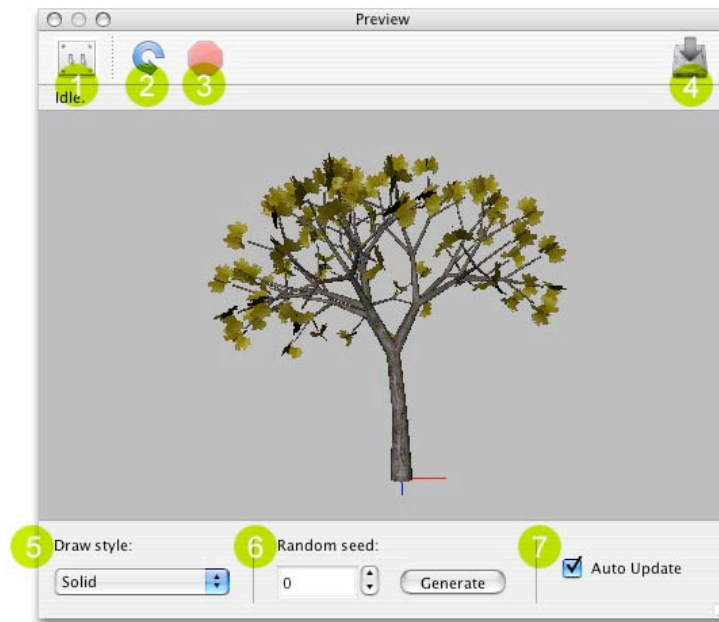


Figure 4.10: The project preview dialog (expanded view).

Figure 4.10 shows the design of the preview dialog. The features of the dialog have been marked numerically in the figure and align with descriptions below:

1. Click this to toggle whether or not to show the settings widgets, which includes showing/hiding widgets (5), (6) and (7).
2. Click to re-build the geometric model. It will be disabled if construction is already in progress.
3. Click to stop building the model. This button will only be enabled while a model is being constructed.
4. Click this to export the model to disk, so that it can be used by an external renderer. You will be presented with a save dialog to save to a number of 3D geometric formats, including VRML 2.0 (also known as VRML97).
5. Draw style selector. You may select between *Solid*, *Wireframe* or *Solid+Wireframe*.
6. Random seed setter. Enter a different number or click on the “Generate” button to build a different variant, when one or more arcs have *variation* amount greater than 0%.
7. Auto-update toggle. If it is checked (on), then when any model influencing change is made to the project, the preview will be automatically updated.

Chapter 5

System Implementation

The system implementation is discussed in this chapter, which builds upon the methodology (Chapter 3) and conforms to the system design (Chapter 4). This proof-of-concept application is known as *Evergreen*, indicated in Figure 5.1. It was written in C++, using various libraries, as listed in section 5.2. EverGreen has been released under the GNU GPL (General Public License), as the libraries used where themselves made available under this license. It was written on an Apple Macintosh, running Mac OS X 10.3. However, given that the libraries used are cross-platform, the system could be ported to other operating systems, such as Linux or Microsoft Windows, with relative ease.



Figure 5.1: About EverGreen, the system implementation.

5.1 Source Code

To access the source code please see the accompanying CD or try the web site:

`http://www.csse.monash.edu.au/~alane`

The source code consists of more than 17,000 lines of C++ code (including comments). Please refer to Appendix D for the class hierarchy.

5.2 Libraries Used

The software libraries and APIs (Application Programming Interfaces) used by the implementation are:

Trolltech Qt 3.3.3: Backbone of the implementation. Used for the GUI (Graphical User Interface), file handling, event handling, threading and much more. (Trolltech; 2004).

SIM Coin3D 2.3.0: Systems In Motions's OpenInventor implementation, which uses OpenGL. Used for visualisation throughout the application, including the project preview, the topology graph editor, and point set editors (Systems in Motion AS; 2004a).

SIM SoQt 1.2.0: Binds Coin3D and Qt together (Systems in Motion AS; 2004b).

5.3 Limitations

Due to time constraints and prioritisation of specific goals, some of the features proposed in the system design could not be implemented. They include:

Arc destination exceptions: All daughter branches that lead out of a branching point when used by an arc, branch to the default destination node (of an arc), without exception.

Branching point distribution & growth: Although the editors have been implemented, the point-sets are not used when the geometric models are constructed.

5.4 Future Extensions

This section contains possible extensions to the current implementation that would not require any changes to the current methodology.

5.4.1 Branch Contours and Curves

If a spline editor were implemented, then branch contours and curves could be supported. These could be integrated directly into the existing node property architecture, as used by materials. They would enable individual branches to be modelled more realistically, in comparison to the tapered cylinders currently used. They would also allow leaves to be modelled directly, without the need for textures.

5.4.2 Animation

Almost any numeric field could benefit from being “animatable”. Such a field could have different values set for arbitrary time slots. To provide smooth animation, the values in between these time slots could be interpolated in some form or another. For example, the divergence of a set of petals in a flower could be set to 0.0 at time 0.0 and 190.0 at time 5.0. An animation could then be generated over the time interval [0.0, 5.0], where the flower will shown to be smoothly opening.

5.4.3 Scripting

If a particular branching feature is not included in the implementation, then the modeller would have to improvise. Therefore, it maybe useful to have a form of scripting embedded into the application. This could be used in arc conditions, where by calculations could be performed on supplied variables, in order to determine if a given arc should be used in a given instance. This could be used for setting-up geometric bounds, for which the branches of a plant must not exceed. Scripting could also be useful for nodes, in which case they could be executed every time the given node is used in a plant construction. Node scripts could be used to modify the node's own parameters, as to produce an effect. Without doubt, the addition of scripting would be targeted toward the more advanced modeller.

Chapter 6

Results and Discussion

This chapter presents and analyses various features of the new plant modelling system. In addition, comparisons are made with existing methods.

6.1 Arc Conditions

As discussed in section 4.2.5, conditions can be used to control if and when specific arcs are used during the construction of a geometric model.

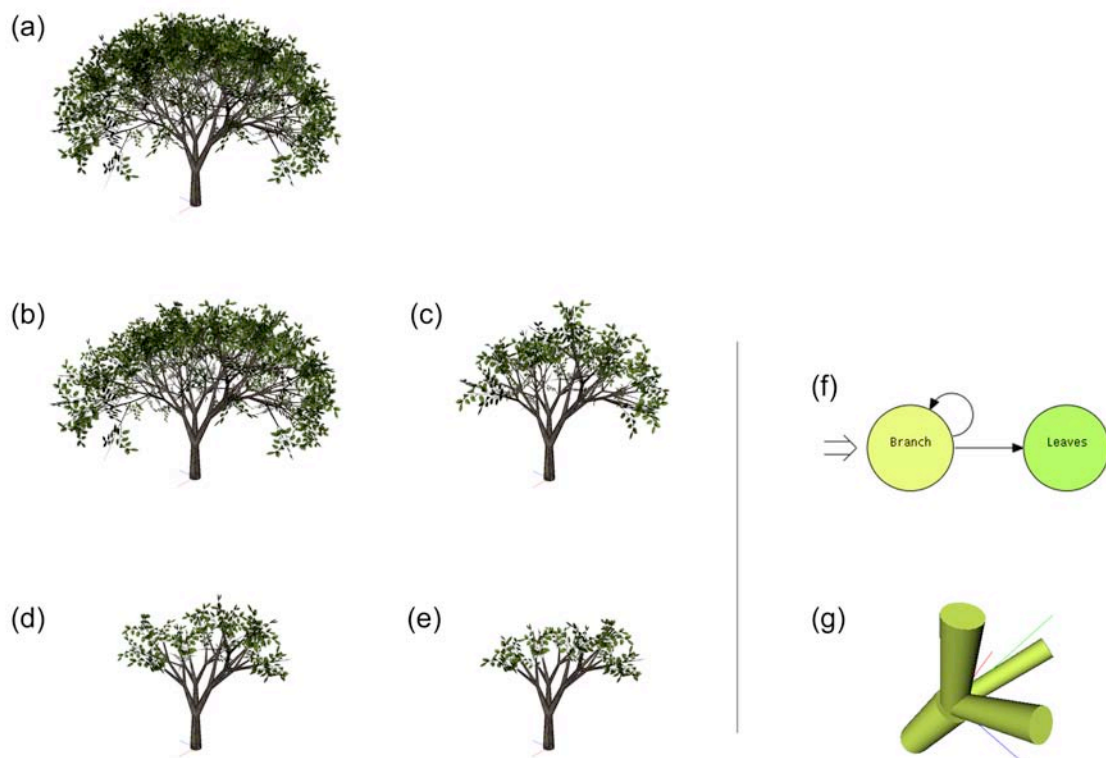


Figure 6.1: Arc conditions: (a) Original tree; (b) branching limited to when the diameter is greater than 10mm; (c) > 20mm; (d) > 30mm; (e) > 40mm; (f) the topology; (g) the branching point.

Figure 6.1 illustrates the use of arc conditions. The original tree has a fixed branching depth of 6 levels. However, in a real plant, there will be varying levels of branching.

A reason for this is that the branches at each level typically do not have the exact same diameters. Hence, some branches may be capable of supporting another level of branching, while others may not. Therefore, in this example, a condition was placed on the arc, which limited its use depending on the current branch diameter. From the models produced, we can see that the branching depth is higher on the right-hand side of the tree. This is due to right-most daughter branch of the first junction having the largest diameter in its branching point set, which kept the diameter along that path higher than others, satisfying more conditions.

As previously noted, conditions can also be placed on the value of the current growth. This is perhaps best suited to toggling parts of the plant as the growth constant on a node is varied. This could be useful animations of growth.

6.2 Branching Point Variations

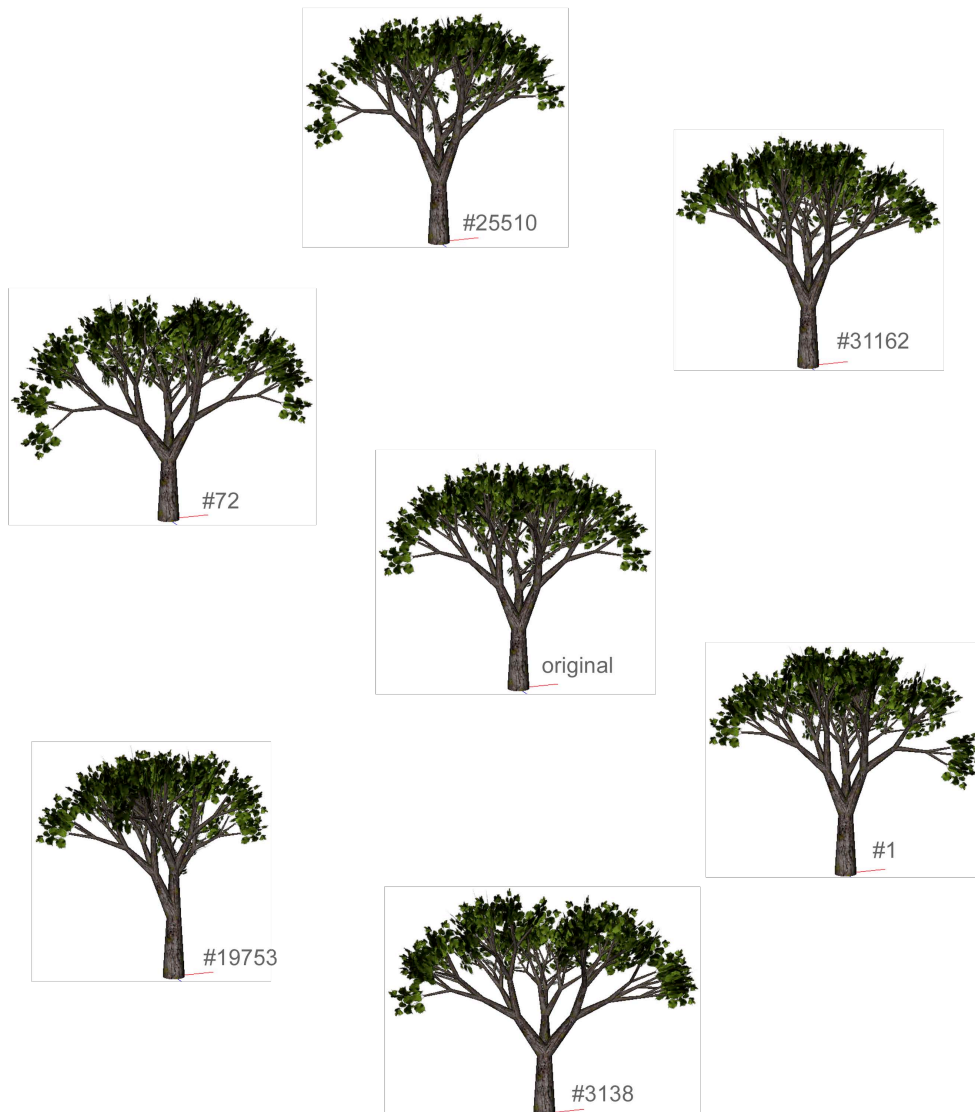


Figure 6.2: Branching point variations. The original tree is located in the centre, with variants surrounding it, along with the random seed used to create them.

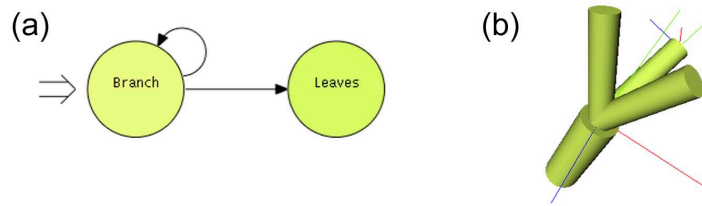


Figure 6.3: For the example shown in Figure 6.2: (a) the topology; (b) the branching point used.

Stochastic branching point variations can be used to produce entire forests of plants from a single model. The resulting variations should be subtly unique, while also remaining faithful the overall original design. Figure 6.2 illustrates the use of this concept, where the plants surrounding the original vary by up to 30%. The topology and branching point used can be seen in Figure 6.3. Please refer to Appendix C for details on how this works.

Variations may also be used to produce branching patterns that appear to be more natural. This is because in nature there are a vast array of environmental factors that affect the way in which plants grow. This includes photo-synthesis, photo-tropisms¹, soil conditions, disease, and pruning. Therefore, to account for some of these factors, stochastic variation of the branching points can be used. This can also be seen in Figure 6.2, where most of the variants look more natural than the original ridged idealistic plant.

6.3 Model Comparisons

In this section, the new method’s way of describing plants will be discussed, in comparison to L-systems and XFROG (Lindenmayer; 1968) (Deussen and Lintermann; 1997).

6.3.1 L-systems

In general, performing comparisons between L-systems and graph-based methods is unfair, given that many details can be hidden in the nodes. Therefore, the approach taken here was to create a model in EverGreen (the implementation) in a way that functions much like an L-system grammar, just without the symbol replacement effect.

Figure 6.4 lists a seemingly-cryptic L-system grammar, along with a geometric turtle interpretation. Likewise, figure 6.5 shows a similar model using EverGreen. This model has been complicated due to the requirement of three branches starting from the base, instead of just one. This required two extra nodes: a base node with a growth of 0% (i.e. length = 0 units) that creates the initial 3-way fork and a trunk node that restores the growth. (This technique can also be used to include roots in the plant models.) Therefore, if there was only one trunk branch, then these two nodes could be removed. The rest of the model works in a way that is similar to the L-system grammar. Application of the first L-system production rule (p_1) produces a 3-way fork, along with some leaves. This is roughly the same as traversing from the “Branch” node to “Branch2” and “Leaf” (where multiple branching points are placed along the branch). The second and third production rules (p_2 and p_3) call for a twist around the current heading and for a leaf to be drawn, as reflected by traversing from “Branch2” to “Branch” and “Leaf”.

Although it was possible to create a model that functioned in a similar way to an L-system grammar, the result was not at all concise. Therefore, the way in which the new method has been used here is not advisable. Instead, its own concepts should be exploited.

¹Typically, plants will try to grow toward the direction of sunlight.

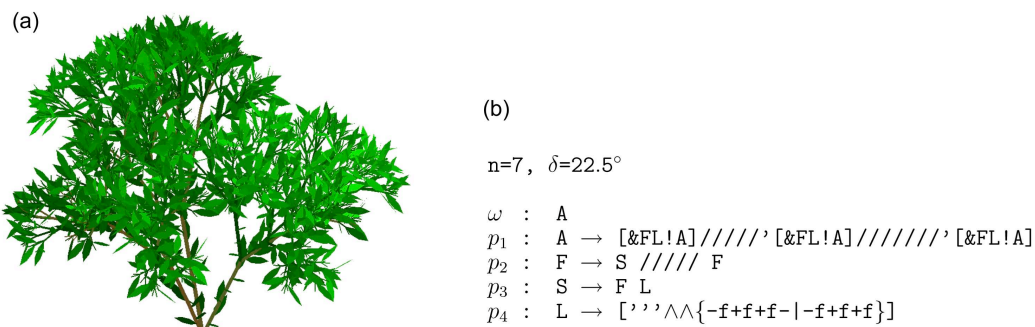


Figure 6.4: L-System: (a) Geometric model; (b) plant grammar. (see Prusinkiewicz and Lindenmayer; 1990, page 26)

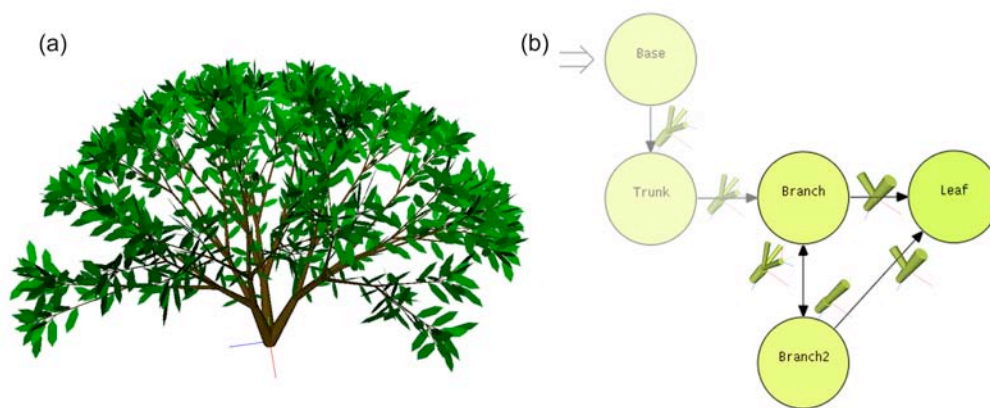


Figure 6.5: EverGreen: (a) Geometric model; (b) plant topology and branching points.

6.3.2 XFROG

Figure 6.6 presents an XFROG plant structure, along with the rendered product of its geometry. Figure 6.7 shows a similar model using EverGreen. Unlike the previous comparison, the way in which EverGreen was used to model this plant was not influenced by the description used to produce the target image (specifically, XFROG’s data structure). Without spline editors and distribution sets, EverGreen is in one respect at a disadvantage when compared to XFROG. However, arc conditions and branching point variations have been used to help produce an image that is quite realistic. Furthermore, the plant architecture is quite concise, formed from just 3 nodes and 3 branching point types (one was used twice). Most people would agree, XFROG’s plant structure is daunting in comparison.

6.4 Summary

In summary, the new method has been used successfully to produce images of plants with realistic branching structures. Arc conditions and branching point variations have been successful components.

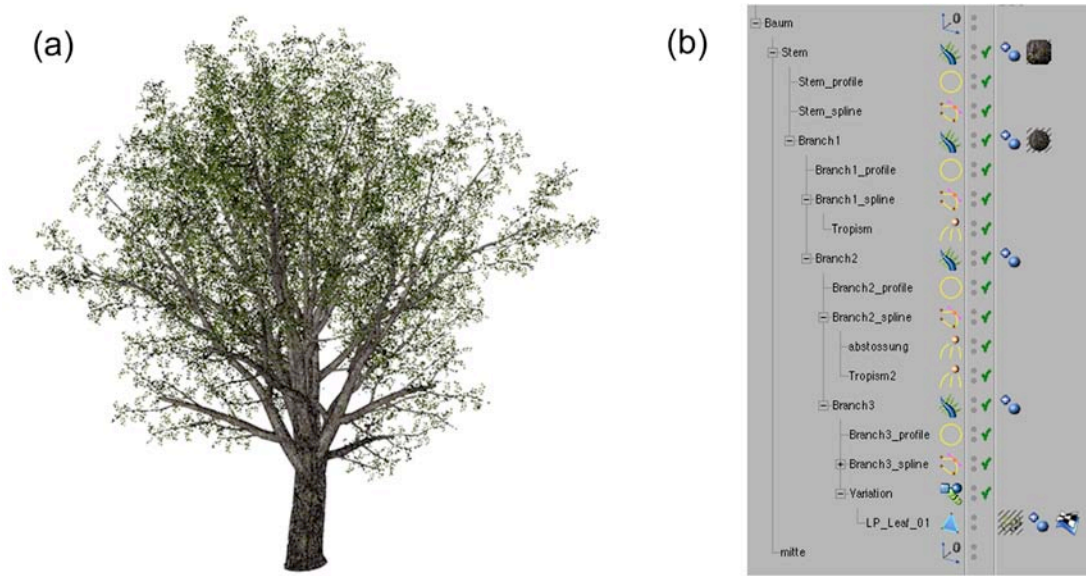


Figure 6.6: XFROG: (a) Geometric model; (b) plant structure.

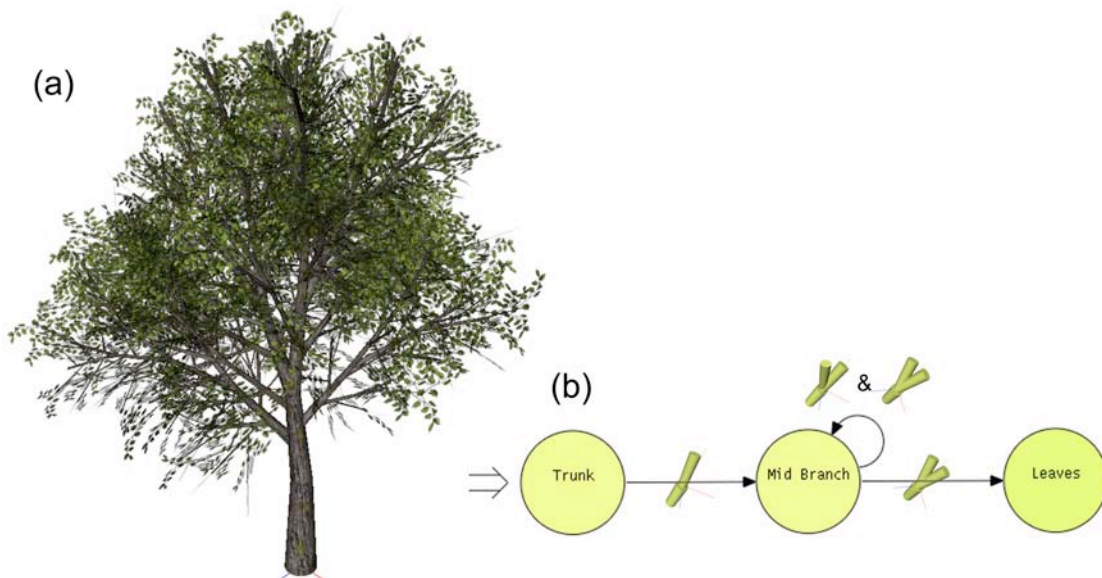


Figure 6.7: EverGreen: (a) Geometric model; (b) plant topology and branching points.

Chapter 7

Conclusion

The research in this thesis was motivated by the desire to model plants in a way that allows realistic branching structures to be created easily and concisely. Ultimately, I hoped to produce synthetic plant imagery that is visually realistic. In this chapter, we review the new modelling method and its implementation, and provide directions for further research.

7.1 Review

This thesis has presented a new, biologically inspired method for modelling plants. The method outlined in Chapter 3 allows plant architectures to be described concisely, using existing concepts. This includes *branching points*, which are made available as reusable “building-blocks”. Biologists should feel comfortable with them, given they already use them in describing branching structures of plants (Hatta et al.; 1999). Others should be able to grasp the concept quite easily, given that they are simple and readily observable in nature. Equally important is the way in which the branching topology can be specified. For this, a type of graph has been proposed that allows the branching points to be linked together, along with other properties. Thus, this method enables realistic branching structures to be created and should appeal to a broader audience of modellers, in comparison to previous systems. In particular, it should appeal to non-experts. Conditional branching can be also used to simplify the modelling process, as they enable an overall branching structure to be created, which can then be “pruned down” to conform with laws of physics. These branching structures can include stochastic variation of individual components, which enables entire forests of plants to be created from a single model.

7.2 Future Research

The new method that has been presented here works in way that is fairly unique. In addition, it is quite flexible and extensible. Therefore, the scope of the method and its applications is open-ended.

The addition of a complete set of positional information should both improve the ease in which the intricacies are modelled and realism of the results (Prusinkiewicz et al.; 2001). Perhaps the ability to define curves that mutate the diameter and divergence of branching points along a branch could prove beneficial. For example, in many ferns, the divergence of branches changes along the stem, from pointing downwards to upwards.

The graph representation may also be improved, so that it provides more information about the curves. This could include which branching point as used by each arc and the properties used by the nodes.

The method benefit from a better branching point variation algorithm, one that supports twists, not just the diameter and divergence.

New types of arc conditions could be introduced, which could use “rules” from biology. They could also be the means of supporting topiary.

Future research is the key to the longevity of this method. Hopefully, with this research as a base, others will be able to succeed in this endeavor.

Appendix A

Revised Specification of Deliverables

The research proposal outlined a primary goal that has remained the same throughout the year, through to the project's end. That goal was to research and design a method that would enable the synthesis of tree and plant imagery that was also relatively easy to use. However, the process by which this method works, in comparison to the one forecast does differ. The research proposal stated that the method would use autonomous evolution of L-systems. This thought was premature, as discovered when more background research was carried out.

Appendix B

Clarification of Original Contribution

In reading that branching point types could be used in the description of branching structures (Zhi et al.; 2001), I began to think about how they could be used as primitives (or building blocks) in a new plant modelling method. They provide a concept that most people can relate to, they can be measured from *real* plants and in all the research that I undertook, no existing method had used them. One of the largest problems was how to work out how to assemble these points, in order to describe a plant architecture. As I wished to include support for recursion and in general, reuse of primitives, I eventually derived a way in which graphs could be used to facilitate this. Specifically, nodes represent branches or objects (such as leaves, flowers and fruit) and the arcs represent the application of the branching points. From there, I designed the complete methodology (Chapter 3) and system design (Chapter 4), with inspiration from existing methods. The way in which variations are supported is also new, whereby branching points may be varied as a complete entity, while staying true the overall shape of the model (see Appendix C).

Appendix C

Branching Point Variation Algorithm

Branching point set:

$$\langle D, B_1 \dots B_n, C \rangle$$

Arc set:

$$\langle N_{from}, N_{to}, BP, V \rangle$$

Please see section 3.2.1 for more details.

Notes:

- Volume minimisation formula has been used from (Zhi et al.; 2001), where volume is the ‘cost’.
- For our purposes, the π constant is redundant in the calculations, therefore it has been left out.
- The twist angle is ignored (left unmodified).
- The parent branch’s diameter is ignored (left unmodified).
- It is assumed that the `cos` function works in *degrees*, as opposed to *radians*.

```
modifyBranchSet( B1 ... Bn, V )
{
  if n < 2 return # need atleast one to target and one to pick up the slack

  variation = random( -V ... V ) // where V = 0.0 ... 1.0

  branch = random( B1 ... Bn )
  varyBranch( branch, variation )

  # Randomly vary the branches by random amounts, until the slack is used.
  # Any branch may be varied multiple times, with the exception of branch.
  variationSlack = -variation
  counter = 0
}
```

```

while abs(variationSlack) > 0.0001 and counter < n
{
  b = random( {B1 ... Bn} - {branch} )
  v = random( 0.0 ... variationSlack )
  modifyBranch( b, v )
  variationSlack += v
  counter ++
}
b = random( {B1 ... Bn} - {branch} )
modifyBranch( b, variationSlack ) // use whatever is left over
}

modifyBranch( branch, variation )
{
  if variation equals 0.0 return

  origDivergence = branch.divergence

  # Modify the divergence by up to variation amount.
  divergenceVariation = random( 0.0, variation )
  v = cos(origDivergence) · (1.0 + divergenceVariation)
  # Make v safe for cos-1 by capping it to -1.0 to 1.0
  # This capping has the bonus effect of capping the divergence to 0-180 degrees.
  if v < -1.0
    v = -1.0
  else if v > 1.0
    v = 1.0
  branch.divergence = cos-1(v)

  # Let the diameter make up any remaining slack.
  origAmount = (branch.diameter/2.0)2 · cos(origDivergence)
  variedAmount = origAmount · (1.0 + variation)
  branch.diameter = sqrt( variedAmount/cos(branch.divergence) ) · 2.0
}

```

Appendix D

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- AlertBox
- BranchListViewItem
- ConditionContainer
 - PaConditionEvent
 - PPArcConditionEvent
 - PvPArcConditionEvent
- DaughterBranch
- EvalRankContainer
 - PaEvalRankChangeEvent
 - PArcEvalRankChangeEvent
 - PvArcEvalRankChangedEvent
- GraphPArc
- GraphPNode
- GraphStartArrow
- GrowthScaleContainer
 - PnGrowthScaleChangeEvent
 - PNodeGrowthScaleChangeEvent
 - PvNodeGrowthScaleChangedEvent
- HorizPointList
- HorizPointSetBase
 - ProjectView< HorizPointSetBase >
 - HorizPointSetEditor
 - DistribPointSetEditor
 - GrowthPointSetEditor
- InheritGrowthContainer
 - PnInheritGrowthChangeEvent
 - PNodeInheritGrowthChangeEvent
 - PvNodeInheritGrowthChangedEvent
- MaxCyclesContainer
 - PnMaxCyclesChangeEvent
 - PNodeMaxCyclesChangeEvent
 - PvNodeMaxCyclesChangedEvent
- MDIChild< managerT, windowT >
- MDIManager< T >
- NameContainer
 - PNodeRenameEvent
 - PnRenameEvent

```

    PRenameEvent
    PvNodeRenamedEvent
NumBranchingPointsContainer
    PnNumBranchingPointsChangeEvent
    PNodeNumBranchingPointsChangeEvent
    PvNodeNumBranchingPointsChangedEvent
PaChangeEvent
    PaBPChangeEvent
    PaConditionEvent
    PaEvalRankChangeEvent
    PaFromChangeEvent
    PaToChangeEvent
    PaVariationChangeEvent
PArc
PArcCondition
    PArcFloatCondition
PArcContainer
    PArcEvent
        PArcBPChangeEvent
        PArcEvalRankChangeEvent
        PArcFromChangeEvent
        PArcToChangeEvent
        PArcVariationChangeEvent
        PPArcConditionEvent
    PnArcEvent
    PvArcEvent
        PvArcBPChangedEvent
        PvArcEvalRankChangedEvent
        PvArcFromChangedEvent
        PvArcToChangedEvent
        PvArcVariationChangedEvent
        PvPArcConditionEvent
PArcEditorBase
    ProjectView< PArcEditorBase >
    PArcEditor
PBranchingPoint
PBranchingPointContainer
    PaBPChangeEvent
    PArcBPChangeEvent
    PBranchingPointEvent
    PvArcBPChangedEvent
    PvBranchingPointEvent
PBranchingPointEditorBase
    ProjectView< PBranchingPointEditorBase >
    PBranchingPointEditor
PField
    PMultiField< FDType >
        PAnimField< FDType >
    PMultiField< PDouble >
        PAnimField< PDouble >
        PAFDouble

```

```

PMultiField< PPoint2d >
  PAnimField< PPoint2d >
    PAFPoint2d
PMultiField< PUInt32 >
  PAnimField< PUInt32 >
    PAFUInt32
PSingleField< FDType >
PSingleField< PBool >
  PSFBool
PSingleField< PString >
  PSFString
PSingleField< PUInt32 >
  PSFUInt32
PFieldData
  PPoint2d
  PPrimitive< Type >
  PPrimitive< bool >
    PBool
  PPrimitive< double >
    PDouble
  PPrimitive< uint32_t >
    PUInt32
  PString
PixmapButton
PixmapButtonPlugin
PMaterialEditorBase
  ProjectView< PMaterialEditorBase >
    PMaterialEditor
PnChangeEvent
  PnArcEvent
  PnBPDistribPointEvent
  PnBPGrowthPointEvent
  PnGrowthScaleChangeEvent
  PnInheritGrowthChangeEvent
  PnLayoutPosChangeEvent
  PnLayoutRadiusChangeEvent
  PnMaxCyclesChangeEvent
  PnNumBranchingPointsChangeEvent
  PnPropertyInstEvent
  PnRenameEvent
  PnTypeChangeEvent
PNode
PNodeContainer
  PaFromChangeEvent
  PArcFromChangeEvent
  PArcToChangeEvent
  PaToChangeEvent
  PLayoutSelectedNodeChangeEvent
  PNodeEvent
    PNodeBPDistribPointEvent
    PNodeBPGrowthPointEvent

```

- PNodeGrowthScaleChangeEvent
- PNodeInheritGrowthChangeEvent
- PNodeLayoutPosChangeEvent
- PNodeLayoutRadiusChangeEvent
- PNodeMaxCyclesChangeEvent
- PNodeNumBranchingPointsChangeEvent
- PNodePropertyInstEvent
- PNodeRenameEvent
- PNodeTypeChangeEvent
- PStartChangeEvent
- PvArcFromChangedEvent
- PvArcToChangedEvent
- PvLayoutSelectedNodeChangedEvent
- PvNodeEvent
 - PvNodeBPDistribPointChangedEvent
 - PvNodeBPGrowthPointChangedEvent
 - PvNodeGrowthScaleChangedEvent
 - PvNodeInheritGrowthChangedEvent
 - PvNodeLayoutPosChangedEvent
 - PvNodeLayoutRadiusChangedEvent
 - PvNodeMaxCyclesChangedEvent
 - PvNodeNumBranchingPointsChangedEvent
 - PvNodePropertyInstEvent
 - PvNodeRenamedEvent
 - PvNodeTypeChangedEvent
- PvStartChangedEvent
- PNodeInspectorBase
 - ProjectView< PNodeInspectorBase >
 - PNodeInspector
- PNodeTypeContainer
 - PNodeTypeChangeEvent
 - PnTypeChangeEvent
 - PvNodeTypeChangedEvent
- PointContainer
 - PnBPDistribPointEvent
 - PnBPGrowthPointEvent
 - PNodeBPDistribPointEvent
 - PNodeBPGrowthPointEvent
 - PvNodeBPDistribPointChangedEvent
 - PvNodeBPGrowthPointChangedEvent
- PointSetEditor
- Pos3f
- PosContainer
 - PnLayoutPosChangeEvent
 - PNodeLayoutPosChangeEvent
 - PvNodeLayoutPosChangedEvent
- PProperty
 - PMaterial
- PPropertyContainer
 - PPropertyEvent
 - PvPropertyEvent

```

PPropertyInstance
PPropertyInstContainer
  PNodePropertyInstEvent
  PnPropertyInstEvent
  PvNodePropertyInstEvent
Project
ProjectEvent
  PArcEvent
  PBranchingPointEvent
  PLayoutSelectedNodeChangeEvent
  PNodeEvent
  PPropertyEvent
  PRenameEvent
  PStartChangeEvent
ProjectGraphViewBase
  ProjectView< ProjectGraphViewBase >
    ProjectGraphView
      MDIChild< MDIProjectManager, ProjectGraphView >
        MDIProject
ProjectManagerBase
  MDIManager< ProjectManagerBase >
    MDIProjectManager
ProjectPaletteBase
  ProjectView< ProjectPaletteBase >
    ProjectPalette
ProjectPreviewBase
  ProjectView< ProjectPreviewBase >
    ProjectPreview
ProjectViewCore
  ProjectView< Type >
  ProjectView< HorizPointSetBase >
  ProjectView< PArcEditorBase >
  ProjectView< PBranchingPointEditorBase >
  ProjectView< PMaterialEditorBase >
  ProjectView< PNodeInspectorBase >
  ProjectView< ProjectGraphViewBase >
  ProjectView< ProjectPaletteBase >
  ProjectView< ProjectPreviewBase >
  ProjectView< QListViewItemWrapper >
    PArcListViewItem
    PPropertyInstListViewItem
  ProjectView< SoQtRenderAreaWrapper >
    GraphEditor
ProjectViewEvent
  PvArcEvent
  PvBranchingPointEvent
  PvLayoutSelectedNodeChangedEvent
  PvNodeEvent
  PvPropertyEvent
  PvStartChangedEvent
QListViewItemWrapper

```

- ProjectView< QListViewItemWrapper >
- RadiusContainer
 - PnLayoutRadiusChangeEvent
 - PNodeLayoutRadiusChangeEvent
 - PvNodeLayoutRadiusChangedEvent
- RankedArcList
- SceneAxesNode
- SceneBranchingPointNode
- SceneProjectPreviewNode
- SceneProjectPreviewNode::BuildState
- SceneProjectPreviewNode::CachedMaterial
- SimpleSceneViewer
- SoQtRenderAreaWrapper
 - ProjectView< SoQtRenderAreaWrapper >
- SuperListView
- SuperListViewPlugin
- SuperSpinBox
- SuperSpinBoxPlugin
- VariationContainer
 - PArcVariationChangeEvent
 - PaVariationChangeEvent
 - PvArcVariationChangedEvent
- WidgetListBox

References

- Curry, R. (1999). *On the Evolution of Parametric L-systems*, Technical Report, No. 1999-644-07, Department of Computer Science, University of Calgary, Calgary.
- Da Vinci, L. (1970). Botany for painters, in J. P. Richter (ed.), *In: The literary works of Leonardo Da Vinci*, Phaidon Publishers Ins., New York.
- Dawkins, R. (1986). *The Blind Watchmaker*, Harlow Logman.
- Deussen, O. and Lintermann, B. (1997). A modeling method and user interface for creating plants, in M. Kaufmann (ed.), *Proc. Graphics Interface 97*, pp. 189–197.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.
- Hatta, H., Honda, H. and Fisher, J. B. (1999). Branching principles governing the architecture of cornus kousa (cornaceae), *Annals of Botany* **84**: 183–193.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor MI.
- Honda, H. (1971). Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body, *Journal of Theoretical Biology* **31**: 331–338.
- Jacob, C. (1994). Genetic L-system programming, in Y. Davidor, H.-P. Schwefel and R. Männer (eds), *Parallel Problem Solving from Nature III*, Vol. 866 of *LNCS*, Springer-Verlag, Jerusalem, pp. 334–343.
- Jacob, C. (1995). Genetic L-system programming: Breeding and evolving artificial flowers with Mathematica, *IMS'95, Proc. First International Mathematica Symposium*, Computational Mechanics Publications, Southampton, UK, pp. 215–222.
- Karwowski, R. and Prusinkiewicz, P. (2003). Design and implementation of the L+C modeling language, in J.-L. Giavitto and P.-E. Moreau (eds), *Electronic Notes in Theoretical Computer Science*, Vol. 86, Elsevier.
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass.
- Lindenmayer, A. (1968). Mathematical models for cellular interactions in development, I & II, *Journal of Theoretical Biology* **18**: 280–315.
- Lintermann, B. and Deussen, O. (1999). Interactive modeling of plants, *IEEE Computer Graphics and Applications* **19**: 56–65.
- Lintermann, B. and Deussen, O. (2004). XFROG 4.1 for CINEMA 4D.
URL: <http://www.xfrog.com/> (Accessed: 25 April 2004)

- McConnell, J. J. (1988). Three dimensional tree grammars for the modeling of plants, *Proceedings of the 1988 ACM sixteenth annual conference on Computer science*, ACM Press, pp. 494–499.
- McCormack, J. (1993). Interactive evolution of L-system grammars for computer graphics modelling., in D. Green and T. Bossomaier (eds), *Complex Systems*, IOS Press, pp. 118–130.
- McCormack, J. (2004). Aesthetic evolution of L-systems revisited, in G. R. et. al. (ed.), *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, Vol. 3005 of *LNCS*, Springer Verlag, Coimbra, Portugal, pp. 475–486.
- Mitchell, M. (1996). *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, Mass.
- Ochoa, G. (1998). On genetic algorithms and lindenmayer systems, *Parallel Problem Solving from Nature IV*, Springer-Verlag, Amsterdam, pp. 335–343.
- Papert, S. (1971). A computer laboratory for elementary schools, *Logo memo 1, MIT Artificial Intelligence Lab*.
- Prusinkiewicz, P. and Hammel, M. (1994). Language-restricted iterated function systems, koch constructions, and L-systems, *New Directions for Fractal Modeling in Computer Graphics*, SIGGRAPH'94 Course Notes. ACM Press.
- Prusinkiewicz, P., Hammel, M. S. and Mjolsness, E. (1993). Animation of plant development, *Computer Graphics* **27**(Annual Conference Series): 351–360.
- Prusinkiewicz, P., Hanan, J. and Měch, R. (2000). An L-system-based plant modeling language, *Lecture Notes in Computer Science* **1779**: 395–410.
- Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants*, Springer-Verlag, New York.
- Prusinkiewicz, P., Lindenmayer, A. and Hanan, J. (1988). Developmental models of herbaceous plants for computer imagery purposes, in J. Dill (ed.), *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, pp. 141–150.
- Prusinkiewicz, P., Mündermann, L., Karwowski, R. and Lane, B. (2001). The use of positional information in the modeling of plants, in E. Fiume (ed.), *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, ACM Press / ACM SIGGRAPH, pp. 289–300.
- Prusinkiewicz, P. and Remphrey, W. R. (2000). Characterization of architectural tree models using L-systems and petri nets, in M. Labrecque (ed.), *L'arbre - The Tree 2000: Papers presented at the 4th International Symposium on the Tree*, pp. 177–186.
- Room, P., Maillette, L. and Hanan, J. (1994). Module and metamer dynamics and virtual plants, *Advances in Ecological Research*, Vol. 25, pp. 105–157.
- Runqiang, B., Chen, P., Burrage, K., Hanan, J., Room, P. and Belward, J. (2002). Derivation of L-system models from measurements of biological branching structures using genetic algorithms, *Lecture Notes in Computer Science* **2358**: 514–524.
- Sims, K. (1994). Evolving virtual creatures, *Computer Graphics* **28**(Annual Conference Series): 15–22.

Systems in Motion AS (2004a). Coin 3D 2.3.0 (GNU GPL version).

URL: <http://www.coin3d.org/> (Accessed: 28 September 2004)

Systems in Motion AS (2004b). SoQt 1.2.0 (GNU GPL version).

URL: <http://www.coin3d.org/> (Accessed: 28 September 2004)

Trolltech (2004). Qt 3.3.3 (Free Edition).

URL: <http://www.trolltech.com/> (Accessed: 28 September 2004)

Weber, J. and Penn, J. (1995). Creation and rendering of realistic trees, in R. Cook (ed.), *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 119–128. held in Los Angeles, California, 06-11 August 1995.

Wolfram Research, Inc. (2004). Mathematica.

URL: <http://www.wolfram.com/> (Accessed: 27 September 2004)

Zhi, W., Ming, Z. and Qi-Xing, Y. (2001). Modeling of branching structures of plants, *Journal of Theoretical Biology* **209**: 383–394.