PARAMETRIC L-SYSTEMS AND THEIR APPLICATION TO THE MODELLING AND VISUALIZATION OF PLANTS

A Dissertation
Submitted to the Faculty of Graduate Studies and Research
In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Computer Science
University of Regina

By
James Scott Hanan
Regina, Saskatchewan
June, 1992

Abstract

In this dissertation, parametric L-systems are presented as the foundation of a computer graphics tool for simulating and visualizing the development of plants.

L-systems were introduced in 1968 by Aristid Lindenmayer as a mathematical model of multicellular organisms. They employ a parallel string-rewriting mechanism to describe the development of branching structures. The resulting strings can be interpreted geometrically and visualized using computer graphics techniques to create both realistic and schematic images of the modelled structures. The formalism can be applied for a variety of scientific, educational, and commercial purposes.

Parametric L-systems extend the original concept of L-systems by associating numerical parameters with the symbols representing plant components. This allows easy quantification of geometric attributes of a model, and provides a simple means for the expression of continuous processes, such as diffusion of hormones and the resulting distribution of concentrations. Formal definitions are proposed for context-free and context-sensitive parametric L-systems with either deterministic or stochastic application of production rules.

The practical value of parametric L-systems is demonstrated in this dissertation by examples that include models of plants ranging from algae to trees. Model development is controlled by lineage mechanisms, with information passed from parent to child module. This mechanism is combined in some models with endogenous interaction, where information flows through a growing structure. Selected models are suitable for simulating time-lapse photography through computer animation.

Extensions to the formalism of parametric L-systems incorporate useful features of other programming languages and provide techniques for creating hierarchical models.

Acknowledgements

First and foremost, I'd like to give heartfelt thanks to my advisor and friend, Przemek Prusinkiewicz, for the many hours we've shared in this work; it's been great fun and I hope it continues.

Inspiring discussions with the late Professor Lindenmayer helped formalize the notion of parametric L-systems, and gave me valuable insight into the biological aspects of my research.

I also appreciated the opportunity to work with Gavin Miller and the rest of the graphics research crew in the Advanced Technology Group at Apple Computer, Inc.

Thanks to the Regina members of my committee: Dr. Maguire, Dr. Symes, Dr. Law, and Dr. Ashton (and his defense stand-in, Dr. Raju). Special thanks to Dr. Wyvill of the University of Calgary for his detailed comments on my dissertation; they helped a lot. The extra-committee comments on my introduction by Jules Bloomenthal, Dr. De Boer, and Dr. Hunter were also much appreciated. Thanks to my external examiner, Dr. Alain Fournier from UBC, for his insightful questions, and for being able to be here on short notice.

My years here in graduate studies wouldn't have been the same without the companionship of my fellow graduate students, in particular the Computer Graphics Groupies, and of the computer science staff. Thanks to my best e-friend, Debbie Fowler, for never being further away than the nearest terminal, and for having the artistic eye in our collaborative image making. Thanks to Lynn Mercer, both for the good times and for the Virtual Laboratory software; it made my life a lot easier. Special thanks to Pauline Van Havere for her help in putting my dissertation through the final stages of the publishing process. Mark Haidl also deserves mention for always being there with the answers for my systems problems.

This research was made possible by equipment grants and a graduate scholarship from the Natural Sciences and Engineering Research Council of Canada, research funding from Apple Computer, Inc., and a Graduate Student Research Scholarship from the Faculty of Graduate Studies and Research at the University of Regina. Facilities of the Departments of Computer Science at both the University of Regina and the University of Calgary were also essential. All support is gratefully acknowledged.

And finally I'd like to thank all my friends and family for being there when I mostly wasn't. The biggest thank you, and all my love, goes to Pam Taylor for her love and support through the seemingly endless hours of work.

It is impossible for anyone to study, even for a short period only, the structure, forms, and colours of plants, and benefits derived from the vegetable creation, without an elevation of thought, a refinement of taste, and an increased love of nature. B. S. Williams (1868)

The central task of a natural science is to make the wonderful commonplace, to show that complexity correctly viewed is only a mask for simplicity, to find pattern hidden in an apparent chaos. H. S. Simon (1969)

Contents

List of Figures

Chapter One

INTRODUCTION

The beauty of the patterns observed in nature has attracted the attention of researchers for many years. Computer simulation, and computer graphics in particular, can play an important role in the understanding of the formation and structure of these patterns. The research presented in this dissertation focuses on the modelling and visualization of plants. The proposed formalism is designed to capture important aspects of the developmental process in a computer model. The resulting images may be either schematic, abstracting from irrelevant details, or realistic, reflecting the beauty which inspired the research in the first place. These representations can have both scientific and aesthetic value, as discussed in the following section.

1.1 Motivation for plant modelling

Plant modelling and the visualizations that result can be applied for a variety of purposes. In the area of biology, a researcher can make hypotheses about the mechanisms controlling the flowering sequences of plants and can then use a computer to visualize the models as a part of their validation. While discovery of a mechanism that will simulate the desired effects does not guarantee that an analogous mechanism is used in nature, it may suggest a possibility worth exploring either experimentally or theoretically. In addition, it may be easier to estimate the values of parameters through simulation than to measure them in the real world.

The modelling process can provide the opportunity to obtain valuable insights

into the nature of the object being studied. The researcher conceptualizes a model based on previous research and observation, then collects data that are considered important. As the model is built, inaccuracies revealed by the visualization may indicate invalid assumptions or previously overlooked factors that should be included in an improved model. A succession of improvements may follow, until the model produces satisfactory results.

Valid models can be used as tools in many areas. Exploration of the parameter space of a model can help in the basic understanding of developmental mechanisms and their effect on plant morphology [13]. The models may be applied for teaching and research in taxonomy; for instance, computer simulations have been used in paleobiology to study evolutionary trends in plant architecture [110]. Models may also provide a vehicle for making predictions involving real plants. The incorporation of information such as seed production and viability may lead to models suitable for evaluating various cropping possibilities; these models can also be used to explore measures of plant fitness for research in natural selection [138]. In the domain of remote sensing, plant models have been used to study the transport of light in vegetative canopies [19, 56]. Applications can also be found in ecology, where the interaction between plants and their environment is the focus of interest [131].

Animations based on developmental models may simulate time-lapse photography of plant growth, revealing and clarifying important processes. For example, computer modelling makes it possible to abstract from unwanted phenomena; these can range from events of a disruptive nature, such as unforeseen plant sickness or death, to daily changes in leaf position which may obscure long-term developmental processes. Animations can also reveal otherwise invisible or difficult to observe phenomena, such as the flow of hormones in a growing plant structure, or the complete life cycle of an oak tree in a compressed time scale. This is useful for educational as well as research purposes.

The realism of synthetic images has an important role to play. In research applications, an image can be difficult to interpret if it contains artifacts introduced by the underlying modelling technique. For instance, if a model of leaf venation is defined on a grid, the privileged directions imposed on the lines drawn in the resulting image

by the grid's structure may obscure the relationship between the model and the real object. The more realistic the visualization, the more confidence the researcher can have that his understanding and assumptions regarding the underlying model are valid. The images themselves may also be important; for instance, in a simulation of pruning effects, the resulting tree shapes may be the primary goal of a horticulturalist.

Realism can also be an end in itself, with plant images incorporated in animations used for commercial or entertainment purposes. Gardens and forests can be used in flight simulators and other virtual worlds [87, 109]. Landscape architects can produce realistic embodiments of their plans, allowing prospective clients to walk through the gardens and preview their design in different seasons.

Applications such as these will eventually demand that computer models be built for more members of the plant kingdom. Many have been developed to date. For reviews, see Waller and Steingraeber [152], Fisher [39], Bell [13], and Lindenmayer and Prusinkiewicz [94].

1.2 Models of branching structures

The focus of my research on the visualization of the branching structure of plants places an emphasis on those models described as *spatial* by Waller and Steingraeber [152]. These models provide the topological and geometric information necessary to produce an image. In contrast, non-spatial models concentrate on global characteristics, such as total biomass or number of flowers.

A categorization of spatial computer models is proposed in Figure 1.1. Note that this discussion is limited to plant models, and does not include other generative models used in computer graphics, such as shape grammars [55, 140]. The term *impressionistic* is used to describe models that abstract from the details of plant structure and seek to create a graphical impression of the plant being modelled (*cf.* Fournier [46]). *Architectural* models, on the other hand, attempt to recreate the structure of the plant. They often exhibit the property of *data base amplification* [136], meaning that the overall structure of a plant is captured by a relatively small set of rules applied repetitively.

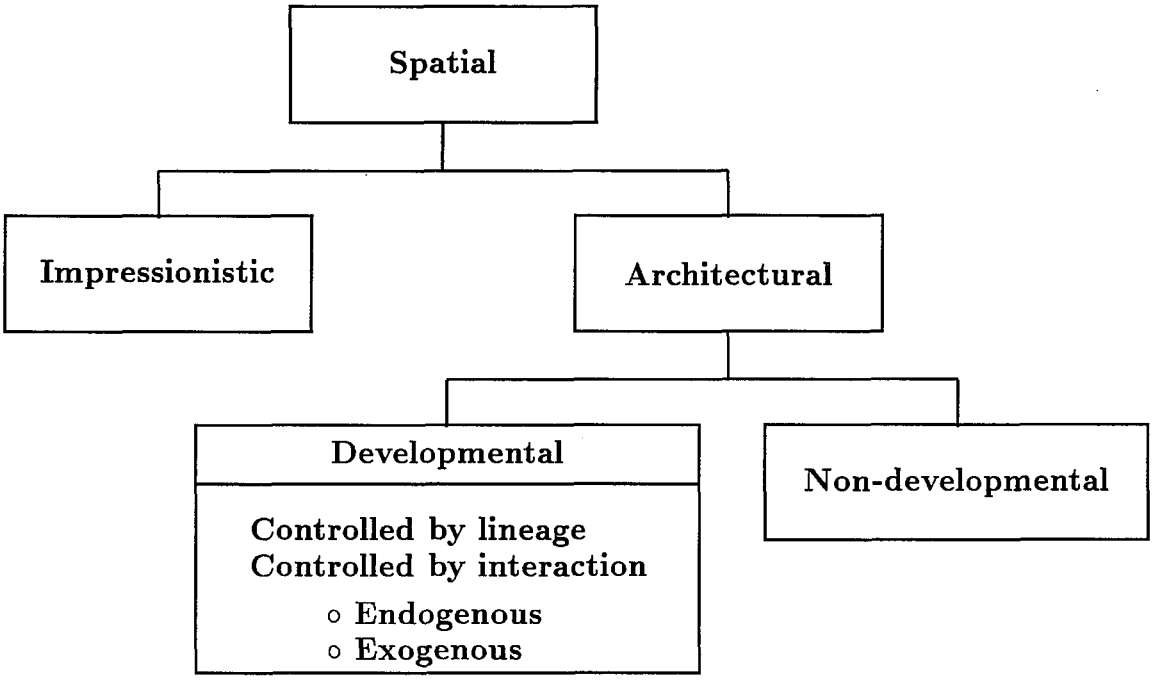


Figure 1.1: A categorization of computer models of branching structures

Architectural models can be further categorized as *developmental* or *non-developmental.* Non-developmental models attempt to capture the essence of a plant's structure in a static way, while developmental models use generative techniques to simulate the growth processes at work in the building blocks, or *modules*, of a plant. This usage of the term "module" conforms to Harper's broad definition [63]: "a repeated unit of multicellular structure, normally arranged in a branch system." In any given model, the modules to be used are chosen for convenience, depending on the desired level of abstraction.

The developmental category, on which this dissertation focuses, may be further characterized by the type of information flow that controls the branching process in the model. *Lineage* mechanisms are employed where an individual module determines its own fate using information passed from parent to child module. An *interactive*

mechanism is at work when development is controlled using information that comes from outside the module. The information flow may be either *endogenous*, with information being passed between adjacent modules within the structure, or *exogenous*, with information being transferred through the medium or environment in which the structure is growing. The simulation of the flow of hormones in a plant is an example of an endogenous mechanism, whereas the reaction of a plant model to self-shadowing is an example of an exogenous mechanism. The distinction between structure and medium is not always obvious. For instance, in a model of a venation pattern in a growing leaf, information passed through the veins would definitely be considered endogenous. On the other hand, information passed through the remaining leaf tissue could be considered either endogenous or exogenous, depending on whether the blade is considered as a part of the structure being modelled or as the growth medium. Note that this usage of the terms endogenous and exogenous is different from the usual biological sense of the words, but follows their usage in the computer simulation literature [8].

Several control mechanisms may operate concurrently in a particular plant model. For example, a lineage mechanism may capture the vegetative growth of a herbaceous plant, while an endogenous mechanism determines the flowering sequence. In the following review, spatial plant models previously described in the literature are classified according to their dominant features.

1.2.1 Impressionistic models

Plant models in this category are useful as features of large-scale terrain representations which need to be drawn quickly, as in flight simulators. In the approach proposed by Gardner [54], quadric surfaces were placed at the desired locations in a landscape and then texture mapped to create the impression of trees. Fournier and Grindal [47] used stochastic texturing on convex polyhedra for the same purpose.

1.2.2 Non-developmental architectural models

Non-developmental architectural models are exemplified by the iterated function systems presented by Demko *et al* [32], and Barnsley [9]. This approach exploits self-similar patterns in the modelled structures, recreating them by repeated application of the corresponding affine geometric transformations. In another approach, intended to produce quick renderings of trees and forests, Marshall *et al* [99] encapsulated the structural details of a variety of tree species in procedures with parameters, such as leaf shape and density, that could be manipulated by the user. Marshall's technique, and many other models in this category, rely on a recursive definition of branching structures. Further examples include the two-dimensional branching structures of Wyvill [157], the models of plant-like forms described by Kawaguchi [86], the particle system models of trees and grass by Reeves and Blau [125], and the highly realistic model of a maple tree by Bloomenthal [16]. The recursive models of Eyrolles [38] and Viennot *et al* [147] were based on a Horton-Strahler analysis of branching structures. Recursive techniques were also used to generate models expressed in terms of fractal geometry, as introduced by Mandelbrot [98]. These include models by Oppenheimer [112, 113], Kaandorp [84], and Berger [15].

The models discussed to this point were inspired by a desire to recreate natural patterns on a computer screen. Biologists pursued a parallel line of research, focusing on the use of simulation to achieve a better understanding of natural forms. Early work in this area was conducted by Honda [74], who created generic tree models in order to study the impact of branching angle and branch length on the shape of trees. His later work with Fisher and Tomlinson produced simulations of selected tree species [40, 78] and employed the resulting models in studies of optimal leaf distribution [41, 75, 76]. Aono and Kunii [3] applied Honda's results for computer graphics purposes, by introducing new geometric models and producing images which simulated the effects of wind and gravity. Waite [151] incorporated a generalization of these recursive branching models into an interactive system.

1.2.3 Developmental models controlled by lineage mechanisms

During plant development, some control is exercised within a plant module; information is passed directly from parent to child module as the plant grows. This lineage mechanism can be thought of as direct "genetic" control of development. In 1968, Lindenmayer [88] developed a mathematical formalism based on language-theoretic principles for modelling developmental processes in multicellular organisms. The version which employs strictly lineage mechanisms is known as zero-interaction Lindenmayer systems or 0L-systems [89]. Building on the modelling work of Frijters [49] and Rozenberg and Lindenmayer [132], Frijters and Lindenmayer [53] employed this formalism to characterize and simulate compound development in branching structures. Pursuing a geometric interpretation of 0L-systems first proposed by Szilard and Quinton [142], Prusinkiewicz [116] developed L-systems with turtle interpretation and presented examples of plant-like structures generated for computer graphics purposes. 0L-systems have also been employed in a variety of plant simulations, for example, Nishida [111] applied stochastic 0L-systems to model variation in Japanese cypress, Morelli *et al* [108] studied speciation in the red algal genus *Dipterosiphonia*, and Corbit and Garbary [26] modelled the morphology and development of several species of the red algal genus *Antithamnion*. Lück and Lück [96] proposed the use of a model closely related to 0L-systems as a possible aid for formalizing plant taxonomy. Shebell [135] developed a generalized set of productions for a stochastic 0L-system, incorporating specific attributes for modelling the 23 tree architectures described by Hallé *et al* [59]. In independent research that parallels my own, but is restricted to deterministic 0L-systems, Chien and Jürgensen [22, 23] proposed an alternative approach for associating numerical values with 0L-system symbols, and provided a corresponding formal definition of graphical interpretation.

In contrast to the mainly deterministic modelling pursued above, other researchers have focused on stochastic methods of simulating lineage mechanisms. This approach often serves as an abstraction from the more complex interactive mechanisms described later, and is most suitable for representing plants that have a high degree of

variability in both topology and geometric attributes such as module length. The random elements also add an important measure of realism to the images synthesized from the models, particularly if a number of plants of the same species are to be included in the same image. Bell [11] developed a stochastic model to simulate the growth of rhizomatous plant populations by careful examination of the probable actions of the meristem. This work was later extended to a variety of species and presented as a basis for further study in plant population ecology by Bell *et al* [14] and Harper and Bell [64]. Cochrane and Ford [24] used a more complex stochastic process based on annual growth increments to model Sitka spruce, while Henderson and Renshaw [67] modelled the root systems of the same tree. In the case of root systems, growth processes were inferred from data obtained at a single point in the tree's life. This was necessitated by the difficulty in obtaining data over time, since the root systems had to be excavated in order to be measured. Further improvements of the root models were reported by Henderson *et al* [65, 66] and Renshaw[130]. Remphrey and Powell [127, 128] also employed stochastic models in their studies of tamarack. De Reffye took a similar approach, using the coffee plant as an example [29, 30]. Subsequent work with various co-workers (for example, [27, 34, 82]) led to the development of a computer system which has been successful in modelling a wide range of tree architectures [31].

1.2.4 Developmental models controlled by endogenous interaction

Lineage mechanisms alone cannot describe all phenomena observed in the development of plants. Some features require the assumption of *endogenous* information flow, corresponding to the movement of hormones or other chemicals of a regulatory nature from module to module within the plant structure.

Interactive Lindenmayer systems (IL-systems) [88] were developed as a discrete formalization of endogenous processes. Early simulations include that of the inhibitor controlled development of blue-green algae by Baker and Herman [6], in which the

modelled topology was displayed as a list of cell states. In 1974, Hogeweg and Hesper [73] employed a computer to explore the range of branching structures produced by a set of simple IL-systems, while Frijters and Lindenmayer [52] modelled the growth and flowering of asters using this formalism. Subsequent models of inflorescences based on IL-systems were investigated by Frijters [50, 51] and Janssen and Lindenmayer [83]. Inspired by the work of Hogeweg and Hesper, Smith [136, 137] demonstrated the potential of L-systems for realistic image synthesis using computer graphics techniques. This potential was tapped by Prusinkiewicz [116, 117] leading to the creation of realistic developmental sequences for various plants which employ endogenous control, as presented by Prusinkiewicz *et al* [123] and Hanan [61].

Building on their earlier models, Honda *et al* [77] incorporated a "growth flux" as an endogenous mechanism determining the growth potential of sibling branches. Borchert and Honda [17], and Borchert and Tomlinson [18] applied this technique to model a specific tropical tree, *Tabebuia rosea*.

On a more detailed level, Ford and Ford [44] described a simulator for growth of *Pinaceae* branches, based on carbon balance and the availability of photosynthate in the branch. Subsequent simulations by Ford *et al* [43] were used to analyse the contribution of a branch to the accumulation of biomass in the remainder of a tree's structure.

Low level models of endogenous information transfer were first studied by Turing [144], who applied continuous mathematics to model the diffusion and reaction of two chemicals in a homogeneous, ring-shaped environment and found that stationary wave patterns could be formed by the resulting concentrations. Meinhardt [102, page 32] proposed the use of this approach at the cellular level to describe phyllotaxis in branching structures, while Brière and Buis [21] applied a similar technique to create an activation-inhibition model of branching growth in a moss. These continuous methods can be computationally expensive for large structures, and their formulation can be difficult if a non-homogeneous system is modelled [91].

1.2.5 Developmental models controlled by exogenous interaction

Exogenous control of development occurs in models where information is transferred through the environment in which the branching structure is embedded. This information may represent regulatory chemicals that flow through a growth medium, physical interactions due to crowding, or the availability of resources such as sunlight and nutrients. One of the most difficult problems for simulation of these control mechanisms is selecting a representation for the environment which minimizes the computational complexity of a given modelling problem. In general, the environment can either be treated as a continuous geometric space, or discretized and represented by a grid.

Early work in the continuous domain was carried out by Cohen [25], who modified growth direction and controlled branching of apices based on a two-dimensional "density field". This field was determined by sampling the space around an apex and calculating the density of branches in the neighbourhood using distance measures to the existing pattern. Many other models employ similar geometric approaches to simulate a variety of environmental factors. Honda *et al* [77] extended their models to include environmental control attributed to physical branch interactions and shading. The fate of a branching point was determined based on information about the vigour of neighbouring branches. Bell used a count of branching points in a neighbourhood for the same purpose [12] and considered shading effects on the production of the photosynthate used to control meristem growth via an endogenous mechanism [13]. Shading effects were also considered by McConnell [100], who created an interactive plant modelling tool using parallel graph grammars that incorporated specific attributes for geometry, age, and the availability of light. A reduction in branch initiation caused by crowding was simulated using branch intersection testing in the stochastic model of bear-berry developed by Remphrey *et al* [126, 129], as well as in an extension of the primarily endogenous model of *Pinaceae* branch growth due to Ford *et al* [43]. A similar approach was taken by Kaandorp [85] in his models of the radiate accretive growth of sponges. Additionally, his models incorporated

comparisons with existing local components to determine the direction and length of new growth. Gottlieb [57] modelled the development of vascular networks in growing tissue by testing whether cells in the tissue were outside of a minimum distance from vessels in the net; if so the closest point to the cell on the net was considered to have received a sufficient concentration of growth factor to sprout a new vessel.

In the exogenous models described to this point, simple geometric notions were applied to capture the information coming from the environment. However, when chemical flow and interaction in the growth medium are considered, modelling becomes more complicated. Reaction-diffusion equations were employed by Meinhardt [102] to control the branching growth of venation patterns in leaves. The equations were discretized over a grid representing the leaf to make the computations more tractable. In this case, the venation pattern was the object being modelled, and the leaf blade was considered the environment in which it grew.

In contrast to the discretization of a continuous process, the cellular automata approach suggested by Ulam and elaborated by von Neumann [150] employed a strictly discrete mechanism. Space was represented as a grid of cells, each being in one of a finite number of states. A uniform set of rules which take into account the state of neighbouring cells was applied to determine the fate of each cell in the space. A variety of complex branching patterns were produced using very simple rules [145, 146].

Eden [35] proposed an approach that employs probabilities to determine growth on a grid. In his model, the next cell to appear was picked at random from those on the periphery of the growing structure. Meakin [101] combined this idea with the diffusion limited aggregation models introduced by Witten and Sander [154]. The resulting diffusion limited growth models of branching structures produced new cells using probabilities that depended on the local concentration of a nutrient diffusing through the environment from an external source.

The plant models of Arvo and Kirk [4] extended the particle system approach to create environment-sensitive automata, suitable for modelling clinging vines and simulating effects such as heliotropism. Each particle was able to estimate the availability of light and the the distance to other objects in the environment, and determined its actions using this information. In order to provide a simple means of self-intersection

testing and to increase rendering efficiency, Greene [58] extended this approach by having the particles move through a discrete voxel space. In his models, possible directions for new growth were generated stochastically and the "best" position was chosen using rules based on both local detection of obstacles and sampling of light conditions.

1.3 Motivation and scope of work

As a major stream in plant modelling research, L-systems have proven to be successful from a practical perspective. The L-system formalism provides a notation in which models utilizing both lineage and endogenous control mechanisms can be specified and then used as input data for modelling programs. As a result, these programs do not have to be recompiled each time a new plant is simulated. The concept of turtle interpretation provides a straightforward means for including geometric information in the string. The interpretation process can provide either schematic or realistic images of the modelled structures. The formalism also provides a clear sequence of time steps in which development occurs. By interpreting the model after each step, a sequence of images illustrating the plant's growth can be produced.

Building on these strengths, I propose parametric L-systems with turtle interpretation as a tool for scientific visualization of development in modular branching organisms. This extension of the L-system formalism provides a convenient means for incorporating real-valued parameters into L-system based models, creating a system capable of mixed discrete-continuous simulation. For instance, at the cellular level the genomic state can be modelled using the discrete properties of the formalism, while continuous concentrations of developmentally important chemicals can be captured using parameter values. Details of growth functions can be expressed using algebraic expressions, avoiding the convoluted approximation of a particular growth function using standard L-systems [122, Section 1.9]. A number of examples are presented to document the usefulness of parametric L-systems for these and other practical modelling purposes.

My research builds on the experience I gained in the development of the Plantworks software system [61] and through its usage for plant modelling [119, 123]. The central contribution of the research reported in this dissertation is the formal definition of parametric L-systems, developed in collaboration with Dr. Lindenmayer and Dr. Prusinkiewicz, and first presented in 1990 along with examples revealing its modelling power [120]. Based on this concept, I designed and implemented a program called the continuous plant and fractal generator or cpfg. This program was used to illustrate the practical usefulness of the concept of parametric L-systems on a wide range of applications. Most of them were described in *The Algorithmic Beauty of Plants* [122], where parametric L-systems served as a notation for expressing ideas as well as a tool for creating images. Based on this experience, the need and potential for further extensions was found; some were previously outlined [62, 121], others are described here for the first time.

1.4 Organization of the dissertation

This section describes the organization of the remainder of this dissertation. Formal definitions of L-systems and details of turtle interpretation are reviewed in Chapters 2 and 3, respectively. Chapter 4 provides motivation for the extension to parametric L-systems and defines them formally. A variety of applications, presented in Chapter 5, illustrate the usefulness of the proposed formalism. Further extensions, based on a view of L-systems as a mini-language for plant modelling, are included in Chapter 6. Chapter 7 summarizes the results of my research and presents problems open for further study. Appendix A describes the plant modelling program cpfg from a user's perspective. Appendix B provides a bridge between theory and practice by giving an overview of implementation considerations. Finally, colour plates are collected in Appendix C.

Chapter Two

L-SYSTEMS

Lindenmayer systems were conceived as a theoretical framework within which the development of multicellular organisms, such as plants, could be modelled [88]. Particular emphasis was given to the mechanisms controlling that development. A plant is composed of a variety of modules, from single cells to plant organs such as leaves or flowers, as defined in Section 1.2 on page 4. In an L-system, each plant module is represented by a letter, different letters being used for modules of different types or in different states. A sequence of letters forms a word which represents the entire plant.

Development is simulated by a process of *rewriting*; a rewriting rule or *production* is applied to a letter, resulting in its replacement by a new letter if the state of the module is to be changed, or by a group of letters if the module divides. Lineage mechanisms are modelled by considering only the current state of a module in choosing the production to be applied, while endogenous control mechanisms are simulated by considering the state of neighbouring cells as well [91]. The productions are applied in parallel, simultaneously replacing all letters in the current word. This reflects the biological motivation of L-systems; many cell divisions may occur at the same time in a plant. The total effect of these changes over time represents the development of the plant.

2.1 0L-systems

This section presents the simplest class of L-systems, in which only lineage mechanisms are used to control development. Since no flow of information between coexisting cells is considered, this class is called *context-free*, *interactionless*, or *zero-sided* L-systems, and is noted 0L-systems [89]. Formal definitions describing 0L-systems and their operation are given below, following the presentation in [70, 133].

Definition 2.1 Let V denote an alphabet, V^* the set of all words over V, and V^+ the set of all nonempty words over V. A *0L-system* is an ordered triplet $G = \langle V, \omega, P \rangle$ where V is the *alphabet* of the system, $\omega \in V^+$ is a nonempty word called the *axiom* and $P \subset V \times V^*$ is a finite *set of productions*. A production $(a, \chi) \in P$ is usually written as $a \rightarrow \chi$. The letter a and the word χ are called the *predecessor* and the *successor* of this production, respectively. It is assumed that for any letter $a \in V$ there is at least one word $\chi \in V^*$ such that $a \rightarrow \chi$. If no production is explicitly specified for a given predecessor $a \in V$, the *identity production* $a \rightarrow a$ is assumed to belong to the set of productions P.

Definition 2.2 A 0L-system is *deterministic* (noted *D0L-system*) if and only if for each $a \in V$ there is exactly one $\chi \in V^*$ such that $a \rightarrow \chi$.

Definition 2.3 Let $\mu = a_1 a_2 \ldots a_m$ be an arbitrary word over V. A production $p : a \rightarrow \chi$ *matches* a letter a_i, $1 \leq i \leq m$, if $a = a_i$. The matching production p can be *applied* to the letter a_i, *producing* the word χ. If a letter a_i produces a word χ as a result of a production application, it is noted $a_i \mapsto \chi$. The word $\nu = \chi_1 \ldots \chi_m \in V^*$ is *directly derived* from (or *generated* by) μ, noted $\mu \Rightarrow \nu$, if and only if $a_i \mapsto \chi_i$ for all $i = 1, \ldots, m$. A word ν is generated by G in a *derivation of length* n if there exists a *developmental sequence* of words $\mu_0, \mu_1, \ldots, \mu_n$ such that $\mu_0 = \omega$, $\mu_n = \nu$ and $\mu_0 \Rightarrow \mu_1 \Rightarrow \ldots \Rightarrow \mu_n$.

The following example provides an illustration of the operation of D0L-systems. The formalism is used to simulate the development of a vegetative fragment of a multicellular filament such as that found in the blue-green alga *Anabaena catenula*

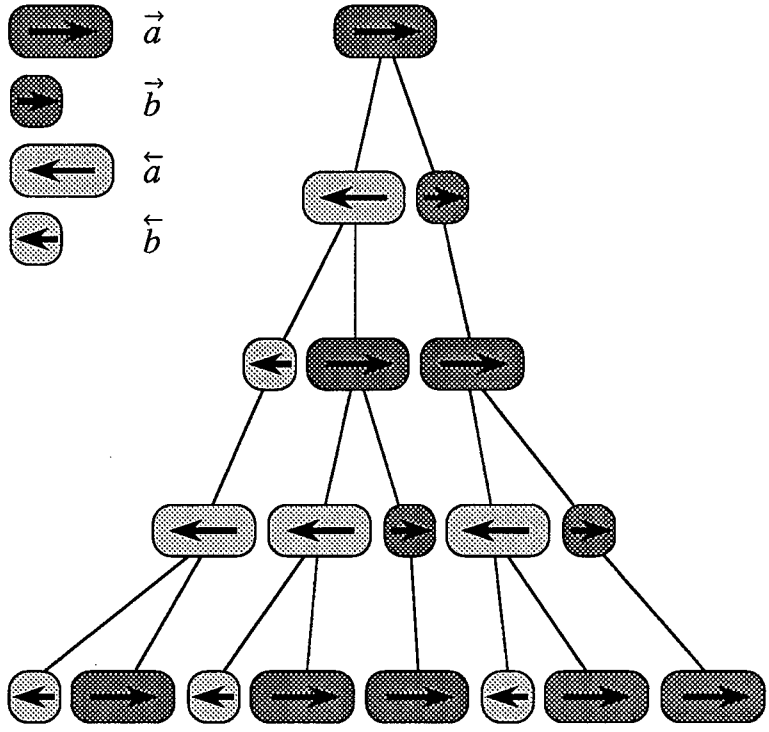


Figure 2.1: Development of a filament of *Anabaena catenula*, simulated using a D0L-system

and various other algae [28, 91, 107]. The symbols a and b represent cytological states of the cells (their size and readiness to divide). The arrows above the symbols indicate cell polarity, specifying the positions in which daughter cells of type a and b will be produced. The development is described by the following L-system:

$$\begin{aligned} \omega &: \vec{\mathtt{a}} \\ p_1 &: \vec{\mathtt{a}} \rightarrow \vec{\mathtt{a}}\,\overleftarrow{\mathtt{b}} \\ p_2 &: \overleftarrow{\mathtt{a}} \rightarrow \overleftarrow{\mathtt{b}}\,\vec{\mathtt{a}} \\ p_3 &: \vec{\mathtt{b}} \rightarrow \vec{\mathtt{a}} \\ p_4 &: \overleftarrow{\mathtt{b}} \rightarrow \overleftarrow{\mathtt{a}} \end{aligned} \tag{2.1}$$

Starting from a single cell $\vec{a}$ (the axiom), the following sequence of words is generated:

$\vec{a}$

$\overleftarrow{a}\vec{b}$

$\overleftarrow{b}\vec{a}\vec{a}$

$\overleftarrow{a}\overrightarrow{a}\overleftarrow{b}\overrightarrow{a}\overleftarrow{b}$

$\overleftarrow{b}\overrightarrow{a}\overleftarrow{b}\overrightarrow{a}\overleftarrow{a}\overrightarrow{b}\overleftarrow{a}\overrightarrow{a}$

...

Under a microscope, the filaments appear as a sequence of cylinders of various lengths, with a-type cells longer than b-type cells. By interpreting the letters in the words in a similar fashion, a sequence of images representing a developing filament can be created as shown in Figure 2.1.

2.2 IL-systems

In the L-system formalism, endogenous control of development is modelled using the context of a letter to determine the applicable production, thus simulating interaction between neighbouring cells. Various types of context-sensitive L-systems have been proposed and studied thoroughly in the past [70, 88, 95, 134]. *2L-systems* use productions of the form $a_l < a > a_r \rightarrow \chi$, where the letter a (called the *strict predecessor*) can produce word χ if and only if a is preceded by letter a_l and followed by a_r. Thus, letters a_l and a_r are called the left and the right context of a in this production. Productions in *1L-systems* have one-sided context only; consequently, they are either of the form $a_l < a \rightarrow \chi$ or $a > a_r \rightarrow \chi$. 0L-systems, 1L-systems and 2L-systems belong to a wider class of *IL-systems*, also called (m,n)*L-systems*. In an (m,n) L-system, the left context is a word of length m and the right context is a word of length n. For standard definitions, see Chapter 6 in the book by Herman and Rozenberg [70] or Chapter 6 in the book by Rozenberg and Salomaa [133]. A strict adherence to these formal definitions leads to long lists of productions, as every combination of letters of length m or n must be specified as a possible context for each letter in the alphabet. For simulation and image generation purposes it is convenient

to generalize the standard definition of IL-systems by allowing contexts of different lengths to coexist in the same L-system. I will note these as I'L-systems, and they are defined as follows.

Definition 2.4 An *I'L-system* is an ordered triplet $G = \langle V, \omega, P \rangle$, where V is the *alphabet* of the system, $\omega \in V^+$ is a nonempty word called the *axiom*, and $P \subset (V^* \times V \times V^*) \times V^*$ is a *finite set of productions*. A production $(\eta_l, a, \eta_r, \chi)$ is represented as $\eta_l < a > \eta_r \rightarrow \chi$. The triplet (η_l, a, η_r) and the word χ are called the *predecessor* and the *successor* of this production, respectively.

The above notation indicates that the *strict predecessor* a can be substituted by χ if and only if a is preceded by the word η_l and followed by the word η_r. Consequently, the words η_l and η_r are called the *left context* and the *right context* of a in the production p_i. A production with an empty left context can be written as $a > \eta_r \rightarrow \chi$. An analogous notation applies if the right context is empty.

Definition 2.5 Production $\eta_l < a > \eta_r \rightarrow \chi$ *matches* word $\mu = a_1 \ldots a_m$ at position $s, 1 \leq s \leq m$, if and only if the word μ can be represented as

$$a_1 \ldots a_{s-l'-1} \eta_l a \eta_r a_{s+r'+1} \ldots a_m$$

where l' and r' denote the lengths of the words η_l and η_r, respectively. If no production is found to match μ at position s, an identity production $a \rightarrow a$, where $a = a_s$, is assumed to match.

An I'L-system is deterministic if and only if no two productions can match the same letter in a string. Since there is no length restriction on the context strings in an I'L-system, two productions with different contexts may match the same letter. This is the case in the following I'L-system:

ω : ABA
p_1 : A > BA $\rightarrow$ C
p_2 : A > B $\rightarrow$ D

Both productions p_1 and p_2 match the first A in the axiom ω. For practical purposes, this form of non-determinism can be avoided by ordering the productions and defining a match in such a way that the first production meeting the appropriate conditions will be chosen, as in the following definition of deterministic I'L-systems.

Definition 2.6 A *DI'L-system* is an ordered triplet $G = \langle V, \omega, P \rangle$, where the alphabet V and the axiom ω are defined as in Definition 2.4, and $P : \{1, \ldots, N\} \rightarrow (V^* \times V \times V^*) \times V^*$ is a finite, *ordered* set of productions. A production p_i will *match* a letter in a string if the conditions of Definition 2.5 apply and there is no production $p_j \in P$ with $j < i$ that also meets these conditions.

The definition of derivation for I'L-systems is the same as Definition 2.3 for 0L-systems. The following sample 1L-system makes use of context to simulate signal propagation through a string:

ω : BAAAA

p_1 : B< A $\rightarrow$ B

p_2 : B >A $\rightarrow$ A

p_3 : B $\rightarrow$ C

The axiom and the first five words generated by this L-system are given below:

BAAAA

ABAAA

AABAA

AAABA

AAAAB

AAAAC

The letter B represents information that moves from the left side to the right side of the string. Note that in the first four derivation steps, production p_2 is chosen to match the module B in the word, since it appears in the list before production p_3, which would also match by Definition 2.5. In the last step, p_2 no longer matches the module B and production p_3 is applied.

2.3 Stochastic L-systems

All plants generated by the same deterministic L-system are identical. This predictability is one of the properties that makes L-systems a useful modelling tool. The user knows that changes in the visualization are a direct result of changes made in the model rather than some random fluctuation, which simplifies the analysis of a particular parameter's contribution to morphogenesis. However, it may not always be desirable or possible to discern an underlying deterministic mechanism of growth. The use of stochastic techniques can provide a convenient abstraction. Data can be collected from a number of specimens and analysed statistically to determine the appropriate probabilities for branching and growth processes. This may also be of great value for image synthesis purposes, where an attempt to combine identical plant models in the same picture can produce a striking, artificial regularity. In order to prevent this effect, it is necessary to introduce specimen-to-specimen variation that will preserve the general aspects of a plant, but will modify its details.

Variation can be achieved by randomizing the interpretation, the L-system, or both. Randomization of the interpretation alone has a limited effect. While the geometric aspects of a plant — such as the stem lengths and branching angles — vary in a random way, the underlying topology remains unchanged. In contrast, stochastic application of productions may affect both the topology and the geometry of the plant. The following definition of stochastic I'L-systems is based on the definitions of stochastic 0L-systems presented by Prusinkiewicz [117], Yokomori [158], and Eichhorst and Savitch [36].

Definition 2.7 A *stochastic I'L-system* is an ordered quadruplet $G_\pi = \langle V, \omega, P, \pi \rangle$. The alphabet V, the axiom ω and the set of productions P are defined as in an I'L-system (Definition 2.4 on page 18). Function $\pi : P \rightarrow \Re$ maps the set of productions into a set of non-negative real numbers called *probability factors*.

Definition 2.8 Let $\hat{P}(\mu, s) \subset P$ denote the subset of productions from P which match word μ at position s. If no production in P matches at a given position, then $\hat{P}(\mu, s)$ is assumed to contain an identity production for the letter at position s with

probability factor 1. The derivation $\mu \Rightarrow \nu$ is a *stochastic derivation* in G_π if for each position s in the word μ the probability of applying production $p_i \in \hat{P}(\mu, s)$ is equal to

$$prob(p_i) = \frac{\pi(p_i)}{\sum_{p_k \in \hat{P}(\mu,s)} \pi(p_k)}. \tag{2.2}$$

An important consequence of this definition is that different productions with the same strict predecessor can be applied to various occurrences of the same letter in one derivation step. Also note that static specification of production probabilities would not be sufficient, since the set of matching productions found for a letter at a given position in the word is not only dependent on the strict predecessor but also on the context. Thus, the number of productions that match a given letter may vary from position to position in the string. A simple example of a stochastic I'L-system is given below.

ω : ABC

p_1 : A < B $\rightarrow$ A : 1

p_2 : B > C $\rightarrow$ BB : 2

The production probability factors must be listed following a colon at the end of each production. In the first and all subsequent derivation steps, the letters A and C will be replaced by themselves using identity productions, as no other productions match. In the first derivation step, both productions match the letter B at position $s = 2$. Production p_1 will be selected with the probability $1/(1+2) = 1/3$, while production p_2 will be selected with the probability $2/(1+2) = 2/3$. If production p_1 is selected, the resulting string will be AAC, and no further changes will occur. If production p_2 is selected, the resulting string will be ABBC. In the next derivation step, the B at position $s = 2$ is only matched by production p_1 which will be applied with probability $1/1 = 1$. Similarly, the B at position $s = 3$ will only be matched by production p_2, which will be applied with probability $2/2 = 1$. The resulting string will be AABBC.

2.4 Modelling branching structures

The formalism presented to this point can only capture models of organisms with a non-branching architecture, since the letters in a word are arranged in a strictly linear fashion. In the second part of his 1968 paper [88], Lindenmayer introduced the notion of *bracketed strings* to describe the branching structure of plants. Left and right brackets, "[" and "]", are added to the alphabet of an L-system. In a correctly formed bracketed string, left and right brackets must occur in matching pairs in the same way as parentheses are used in an arithmetic expression. By considering the brackets as delimiters of branches, the bracketed strings can be interpreted as branching structures. Specifically, a left bracket indicates the node of the mother branch to which the daughter branch is to be attached, while the matching right bracket terminates branch specification. The brackets may be nested, indicating higher-order branches. If more than one branch occurs at a site, the bracketed substrings representing each one are listed consecutively in an arbitrary order. An example of a bracketed string and a corresponding tree structure are shown in Figure 2.2.

In context-free L-systems, the introduction of brackets has no effect on the derivation process; the brackets are simply rewritten into themselves. However, in bracketed I'L-systems, there may be pairs of letters that cannot communicate with each other directly, since the brackets impose a branching topology on the string. All those letters that can communicate with a particular letter are distinguished as its *complete context.*

The left and right contexts have slightly different properties, as illustrated in Figure 2.2. The complete left context represents all modules that are "below" the current module in the branching structure, thus it is a sequence of all letters to the left of a given letter, possibly interrupted by one or more substrings enclosed in matching brackets. In the figure, the string `ABC` is the complete left context of the letter `F`. On the other hand, the complete right context represents all modules which are "above" the current module, thus it can be a bracketed string representing a branching structure. In the figure, the string `G[HI[JK]L]MNO` is the complete right context of the letter `F`. In general, the complete right context of a letter is defined as

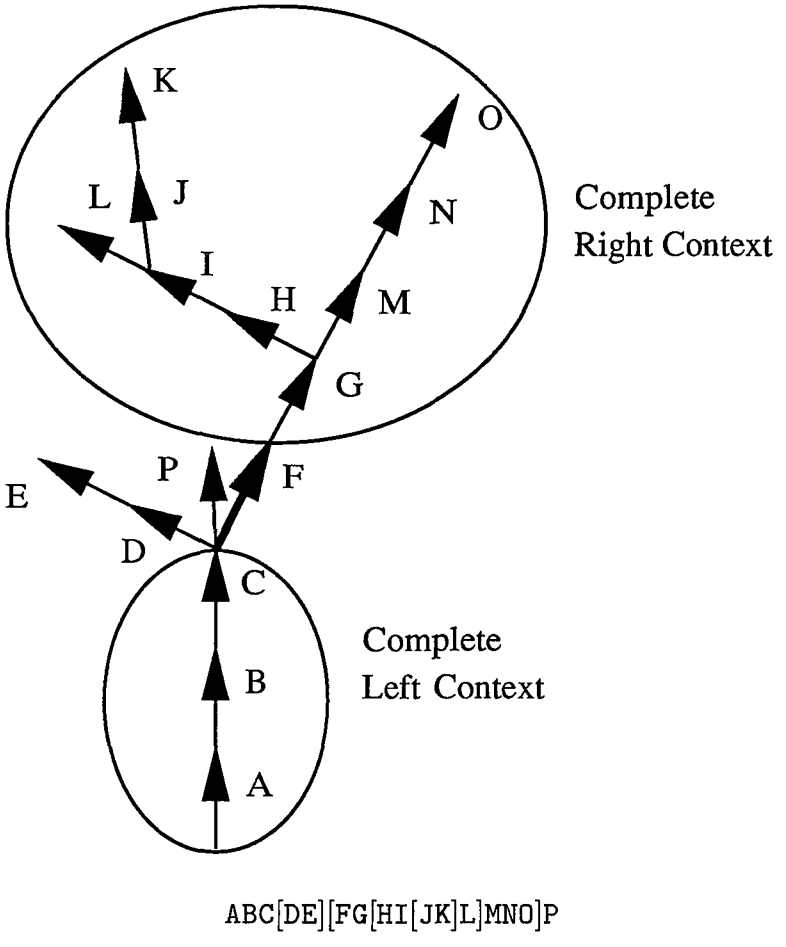


Figure 2.2: A bracketed string and a corresponding tree structure. The complete contexts for the letter F are marked.

---

all letters to its right that are encountered before an unmatched right bracket or the end of the string.

As a consequence of the asymmetry between the definitions of the left and right contexts, they are matched using different procedures. The left context matching procedure presented on page 24 is controlled by an outer while loop that stops when a mismatch has been found or if the end of the string or context has been reached. In the case where a_i == ']', the procedure skips over intervening substrings representing branches that are found in the string. In the case where a_i == '[', the procedure has found a left bracket and skips over it. Otherwise, the current string and context

Procedure	**MatchLeftContext**
Purpose	This procedure attempts to match the substring $a_1 a_2 \ldots a_{s-1}$ with the left context $\eta_l = c_1 c_2 \ldots c_{l'}$, where l' denotes the length of the word η_l. The boolean variable *match* will have value TRUE if the left context matches the string, FALSE otherwise.

```
Let i = s − 1          /* initialize string index */
Let j = l'             /* initialize context index */
Let match = TRUE       /* initialize matching flag */

While match and i > 0 and j > 0
    /* Check for symbols to be skipped */
    DoCase
        /* Skip substrings representing branches */
        Case a_i == ']'
            /* move the index i to point at the first character
               to the left of the matching left bracket */
            i = SkipLeft(i)
        End Case
        /* Skip left brackets */
        Case a_i == '['
            i = i − 1
        End Case
        /* Nothing to be skipped, check for match */
        Otherwise
            If a_i == c_j
                /* OK so far; keep scanning the string and context */
                i = i − 1 /* move left in the string */
                j = j − 1 /* move left in the context */
            Else
                /* Mismatch */
                match = FALSE
            End If
        End Otherwise
    End DoCase
    If i == 0 and j > 0
        /* the string index is past the left end and
           there's still context to match */
        match = FALSE
    End If
End While
```

Procedure	**MatchRightContext**
Purpose	This procedure attempts to match the substring a_{s+1} ... a_m with the right context $\eta_r = c_1$ ... $c_{r'}$, where r' denotes the length of the word η_r. The boolean variable *match* will have value TRUE if the right context matches the string, FALSE otherwise.

```
Let i = s + 1              /* initialize string index */
Let j = 1                  /* initialize context index */
Let match=TRUE             /* initialize matching flag */
While match and i <= m and j <= r'
    DoCase
        /* Check for substrings representing branches that
           have no match in the context */
        Case a_i == '[' and c_j != '['
            /* move the index i to point at the first character
               to the right of the matching right bracket */
            i = SkipRightPastBranch(i)
        End Case
        /* Check for branch end in the context */
        Case c_j == ']'
            /* move the index i to point at the first character to the right
               of the next unmatched right bracket in the string. */
            i = SkipRightPastEnd(i)
            j = j + 1 /* move right in the context */
        End Case
        /* Check for mismatch */
        Otherwise
            If i <= m and a_i == c_j
                /* OK so far; keep scanning the string and context */
                i = i + 1 /* move right in the string */
                j = j + 1 /* move right in the context */
            Else
                match = FALSE
            End If
        End Otherwise
    End DoCase
    If i > m and j < r'
        /* the string index is past the right end and
           there's still context to match */
        match = FALSE
    End If
End While
```

symbols are compared. The use of the DoCase construct ensures that all intervening branches or left brackets are found and skipped before a comparison occurs between a string and a context symbol. In the example presented in Figure 2.2, the left context BC matches the string BC[DE][, with the symbols [DE] and [being skipped.

The right context matching procedure presented on page 25 is controlled by an outer while loop which stops when a mismatch has been found or if the end of the string or context has been reached. In the case where a_i == '[' and c_j != '[', the procedure has found a substring enclosed in matching brackets in the string that is not in the context. The substring and matching brackets are skipped and the next symbol considered in the string is the first one after the closing bracket. In the case where c_j ==']', the procedure has found a right bracket in the context. Symbols are skipped by moving the context pointer to the first symbol after the next unmatched right bracket in the string. Otherwise, if neither of these cases apply, the string and context symbols are compared and the appropriate action taken. In the example, the right context G[H]M matches the string G[HI[JK]][L]M. The symbols I[JK] and [L] are skipped over.

Note that the bracketed string notation imposes an ordering on multiple branches that originate after the same module. For example, the strings

ABC[DE][FG[HI[JK]L]MNO]P and
ABC[FG[HI[JK]L]MNO][DE]P

represent the same structure, as the same two "branches" [FG[HI[JK]L]MNO] and [DE] appear immediately after the symbol C in both strings. If context from a specific branch is required, it is the responsibility of the user to keep track of this order. In addition, this notation does not provide a simple mechanism for collecting information from an arbitrary number of branches at a given location.

Figure 2.3 gives examples of the use of 1L-systems to simulate propagation of signals in a branching structure that does not grow. The letter B represents a segment already reached by the signal, while A represents a segment that has not yet been reached. The left context can be used to simulate control signals that propagate *acropetally*, from the root or basal leaves towards the apices of the modelled plant, while the right context represents signals that propagate *basipetally*, from the apices

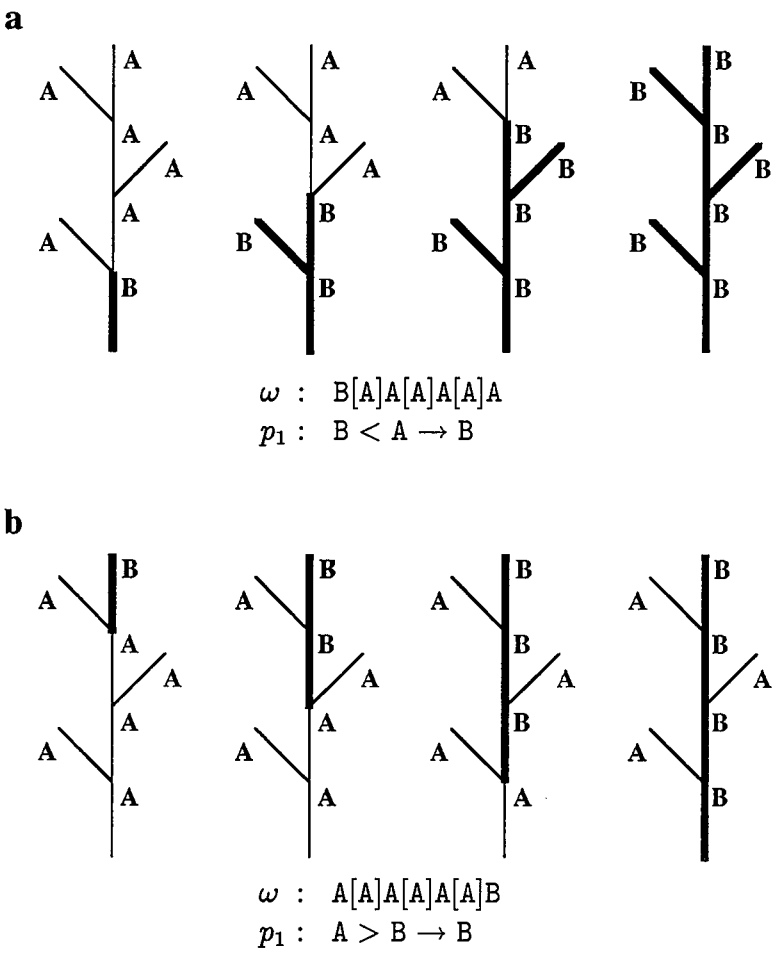


Figure 2.3: Signal propagation in a branching structure: (a) acropetal, (b) basipetal

towards the root. The images represent consecutive stages of signal propagation corresponding to consecutive words generated by the L-system under consideration.

2.5 Symbols not included in context

In order to create a visualization of the plants modelled using L-systems, it is convenient to incorporate symbols carrying information about model geometry into

the bracketed string (see Chapter 3 for more details). These symbols typically occur throughout the string and thus have to be dealt with while context matching. For example, the axiom of the L-system in Figure 2.3a might be changed to `B[GA]A[GA]A[GA]A`, where symbol `G` carries information about branching angles. The application of production p_1 would now result in only the main-stem letters `A` being converted to `B`'s. The `G`'s at the beginning of each branch would prevent the `B`'s in the main stem from being matched as the left context of the `A`'s in the branches.

One solution to this problem would be to add productions passing the signal past each geometric attribute symbol. This may increase the number of productions dramatically, and causes modelling difficulties as the time taken to pass the signal through the string will be proportional to the total number of letters in the string, rather than to the number of letters representing the structure. A better solution, which ensures correct propagation of information regardless of these symbols, is to partition the L-system alphabet into two sets, those considered during context matching and those that are ignored. Using this approach, `G` is specified as the letter to be ignored and the signal will be passed into the branches in the same way as in the original example.

Chapter Three

TURTLE INTERPRETATION

Initially, the models expressed in terms of L-systems were illustrated by listing the consecutive words generated during a derivation. Further visualization relied on human interpretation of the letters as different modules, either in the mind's eye or in the form of hand-drawn images. As an example of this approach, consider the simple geometric interpretation of strings that was applied to create the schematic images of *Anabaena catenula* in Figure 2.1 on page 16. Letters of the L-system alphabet were represented graphically as shorter or longer rectangles with rounded corners. The generated structures were one-dimensional chains of rectangles, reflecting the sequence of symbols in each of the underlying strings. However, there is nothing in the model that restricts the filament to a straight line, so the draftsperson is free to create a curved representation, such as that seen in Figure 3.1, which matches microscopic images of the real organism more closely.

In order to model and visualize complex branching plants, a more sophisticated graphical interpretation of L-systems is needed. The first results in this direction were published in 1974 by Hogeweg and Hesper [73], and Frijters and Lindenmayer [52]. In both cases, bracketed L-systems were used to determine the branching topology of the modelled plants. Geometric aspects, such as branching angles, were added in a post-processing phase. The results of Hogeweg and Hesper were subsequently extended by Smith [136, 137], who demonstrated the potential of L-systems for realistic image synthesis using state-of-the-art computer graphics techniques. In the resulting *graftal* formalism, geometry was added to the topological model in a separate interpretation

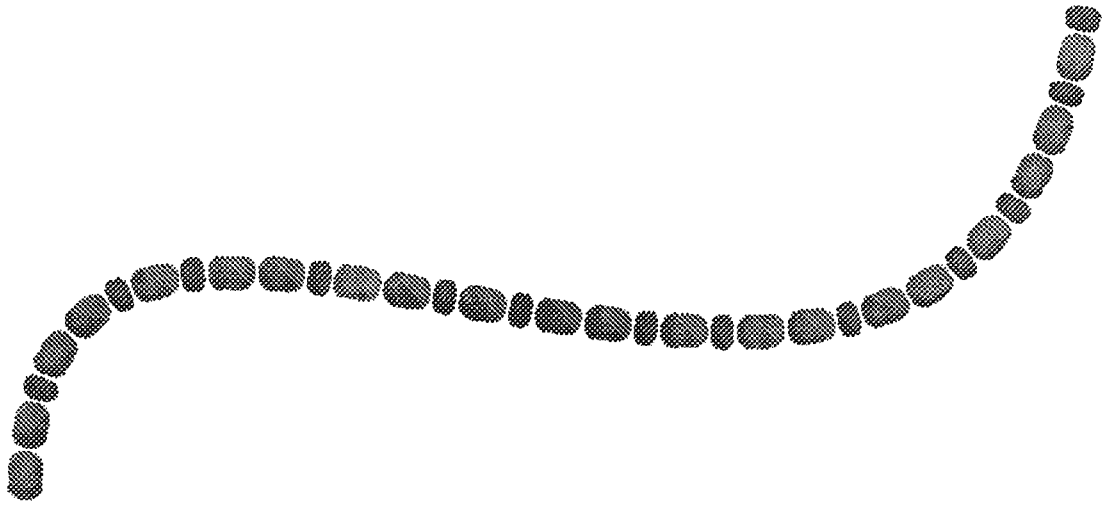


Figure 3.1: Two-dimensional representation of *Anabaena catenula*

step.

In 1979, Szilard and Quinton [142] proposed a different approach, in which each letter of the L-system alphabet was assigned an interpretation as a sequence of commands to a plotter pen, effectively making the geometry internal to the L-system. They showed that strikingly simple D0L-systems could generate the convoluted curves known today as *fractals* [98]. Noting the similarity of their approach to LOGO turtle geometry [1, 114], Prusinkiewicz incorporated the geometric commands directly within L-systems, and extended this *turtle interpretation* to include bracketed IL-systems [116] and three-dimensional models [117]. This is the approach I followed in my research.

In the original concept of LOGO turtle geometry [114], images are created by a pen-carrying "turtle" as it crawls around on a two-dimensional surface. The turtle responds to commands which originate from the user, such as pen up, pen down, turn right, and move forward. The path of the turtle in the "pen down" state is traced on paper or a computer screen.

In turtle interpretation of L-systems, the turtle builds a structure of line segments

as it moves around in a three-dimensional world. Its actions are controlled by the sequence of commands obtained by reading the L-system-generated string from left to right. For example, the commands may cause the turtle to change its orientation in space, move forward, or add a new line segment. Finally, an image is created by rendering a particular view of the turtle's world using standard computer graphics techniques.

The following discussion of turtle interpretation is derived from the presentations of Prusinkiewicz *et al* [122] and Hanan [61].

3.1 Turtle state

The turtle interpreting a string is represented by a set of attributes that constitute its *state*. The major components of the turtle's state are its *position*, represented by three Cartesian coordinates x, y, and z, and its *orientation* in space, represented by three vectors indicating the turtle's *heading* ($\vec{H}$), *left* ($\vec{L}$), and *up* ($\vec{U}$) directions. These vectors have unit length, are perpendicular to each other and satisfy the equation $\vec{H} \times \vec{L} = \vec{U}$. Using this notation, rotations of the turtle are expressed by the formula:

$$\left[\begin{array}{ccc} \vec{H}' & \vec{L}' & \vec{U}' \end{array} \right] = \left[\begin{array}{ccc} \vec{H} & \vec{L} & \vec{U} \end{array} \right] \mathbf{R}$$

where $\mathbf{R}$ is a 3×3 rotation matrix [42]. Specifically, rotations by angle α about vectors $\vec{H}, \vec{L}$ and $\vec{U}$ are represented by the following matrices.

$$\mathbf{R_U}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_L}(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

$$\mathbf{R_H}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

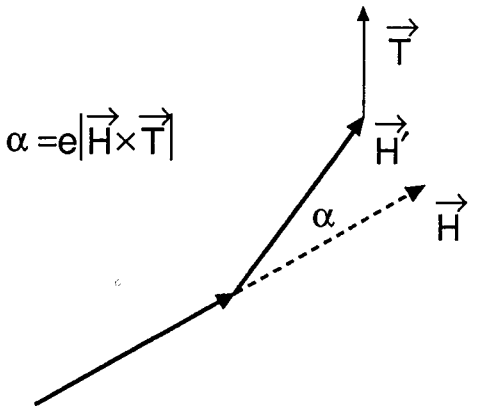


Figure 3.2: Correction α of segment heading $\vec{H}$ due to tropism $\vec{T}$

The remaining elements of the turtle state are its *drawing attributes.* The *colour index* specifies the colour of segments drawn by the turtle. The images are created using the colour map technique [42, page 132], therefore the colour represented by a particular index depends on the contents of the map. The *line width* attribute specifies the width of the segments being drawn. The final drawing attribute is called the *elasticity factor* and is used to model the bending of a plant's branches towards a source of light (phototropism), down due to gravity, or sideways due to wind. These effects are simulated by slightly rotating the turtle in the direction of a predefined *tropism vector* $\vec{T}$ after drawing each segment. Figure 3.2 illustrates the two-dimensional case. The orientation adjustment angle α is calculated from the formula $\alpha = e|\vec{H} \times \vec{T}|$, where the elasticity factor e is a parameter capturing axis susceptibility to bending. This heuristic formula has a physical motivation; if $\vec{T}$ is interpreted as a force applied to the endpoint of segment $\vec{H}$, and $\vec{H}$ can rotate around its starting point, the torque is equal to $\vec{H} \times \vec{T}$. In the three-dimensional case, similar heuristics are applied to rotate the turtle, first around its left axis and then around its up axis, thus re-aligning the turtle's heading toward the tropism direction. The effect of tropism on a branching structure is illustrated in Figure 3.3.

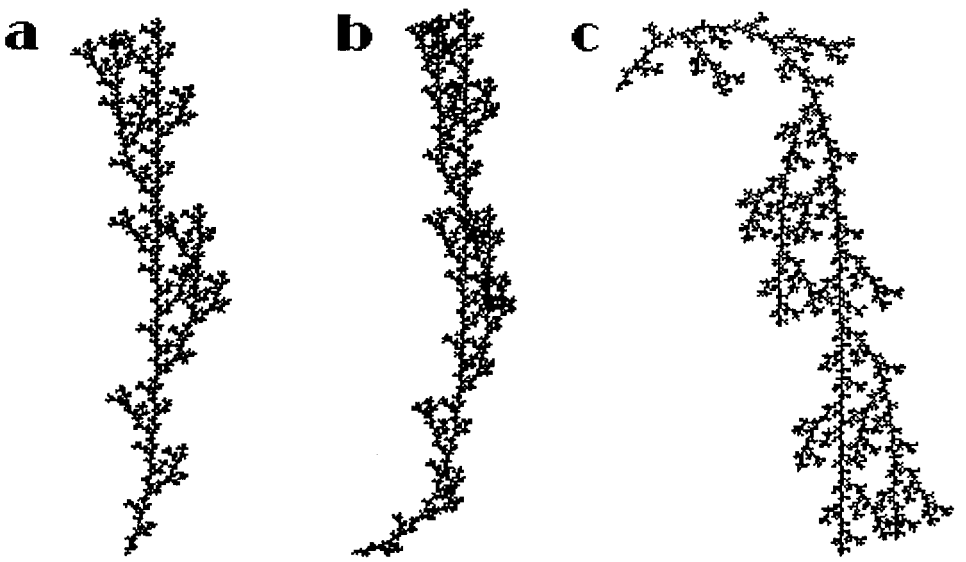


Figure 3.3: Modelling tropism. The tropism vector $\vec{T}$ points up. The coefficients e used to generate structures a–c satisfy the relation $e_b > e_a > 0 > e_c$.

3.2 Basic commands

As the turtle scans the L-system string from left to right, it encounters specific letters that have been assigned an interpretation. Typically, these *command* letters result in a modification of the turtle's state. The interpretation of a single symbol causes the appropriate state attribute to be changed by an amount specified by parameters making up the turtle's *drawing environment.* The *step size* d specifies the distance the turtle moves in the direction of its heading vector. The *angle increment* δ controls the size of the angle that the turtle is rotated. The *colour index increment*, *line width increment*, and *elasticity increment* specify the amount by which the respective state attribute is increased or decreased. The *tropism vector* $\vec{T}$, described in the previous section, is the final parameter.

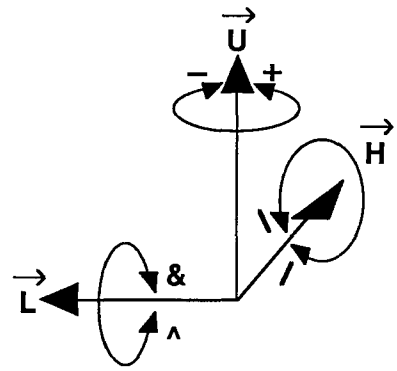


Figure 3.4: Controlling the turtle in three dimensions

Given this drawing environment, the turtle responds to a basic set of commands that can be divided into the following categories:

Moving and drawing commands

`F` Move forward a step of length d. The position of the turtle changes to (x', y', z'), where $x' = x + d\vec{H}_x$, $y' = y + d\vec{H}_y$, and $z' = z + d\vec{H}_z$. A line segment is drawn between points (x, y, z) and (x', y', z').

`f` Move forward a step of length d without drawing a line.

Orientation modifying commands (Figure 3.4)

`+` Turn left by angle δ. The rotation matrix is equal to $\mathbf{R_U}(\delta)$.

`−` Turn right by angle δ. The rotation matrix is equal to $\mathbf{R_U}(-\delta)$.

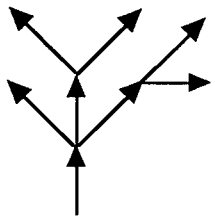
`&` Pitch down by angle δ. The rotation matrix is equal to $\mathbf{R_L}(\delta)$.

`^` Pitch up by angle δ. The rotation matrix is equal to $\mathbf{R_L}(-\delta)$.

`\` Roll left by angle δ. The rotation matrix is equal to $\mathbf{R_H}(\delta)$.

`/` Roll right by angle δ. The rotation matrix is equal to $\mathbf{R_H}(-\delta)$.

`|` Turn around. The rotation matrix is equal to $\mathbf{R_U}(180°)$.



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

Figure 3.5: Turtle interpretation of a bracketed string

Branch modelling commands (Figure 3.5)

[Push the current state of the turtle onto a pushdown stack.

] Pop a state from the stack and make it the current state of the turtle. No line is drawn, although in general the position of the turtle changes.

Commands changing drawing attributes

, Increase the value of the current index into the colour map by the colour increment.

; Decrease the value of the current index into the colour map by the colour increment.

Increase the value of the current line width by the line width increment.

! Decrease the value of the current line width by the line width increment.

" Increase the value of the current elasticity factor by the elasticity increment.

' Decrease the value of the current elasticity factor by the elasticity increment.

3.3 Surface modelling

The turtle interpretation commands presented to this point only allow the creation of structures composed of line segments. However, plants have many components, such as leaves and petals, that are more naturally modelled as surfaces.

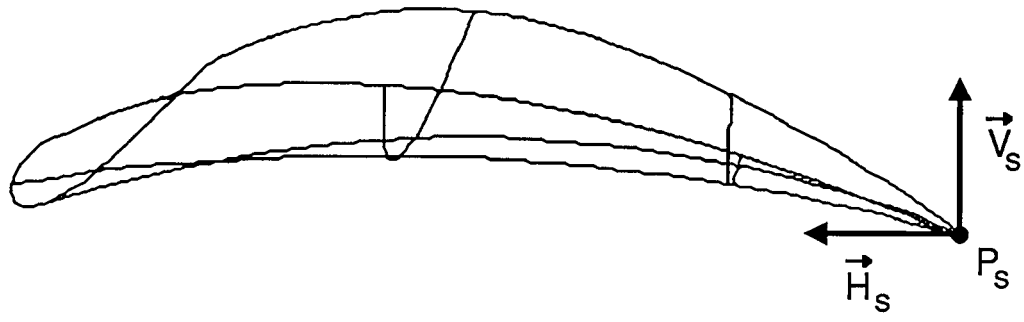


Figure 3.6: Surface specification

Two approaches to surface modelling using turtle interpretation are presented in this section.

3.3.1 Predefined surfaces

A standard computer graphics method for defining surfaces makes use of *parametric bicubic patches* [10, 42]. This technique is well suited for interactive design of arbitrary surface shapes. The control points that define an individual patch can be modified using a graphical interface [61, Section 4.2], and several patches can be combined to create a more complex surface [61, Section 3.5]. The resulting surface definition can then be stored in a file for use during turtle interpretation. Further details of the file format can be found in Section A.3.4 on page 164.

Predefined surfaces are incorporated into a plant model by extending the L-system alphabet. When the turtle encounters a symbol representing a surface preceded by a tilde ($\sim$), the corresponding surface is drawn. The exact position and orientation of a predefined surface S is determined using the user-defined *contact point* P_S, *heading vector* $\vec{H}_S$, and *up vector* $\vec{U}_S$ as references (Figure 3.6). The surface is translated in such a way that its contact point matches the current position of the turtle, and is rotated to align its heading and up vectors with the corresponding vectors of the turtle. If a surface represents an internal part of a plant's structure, the turtle is

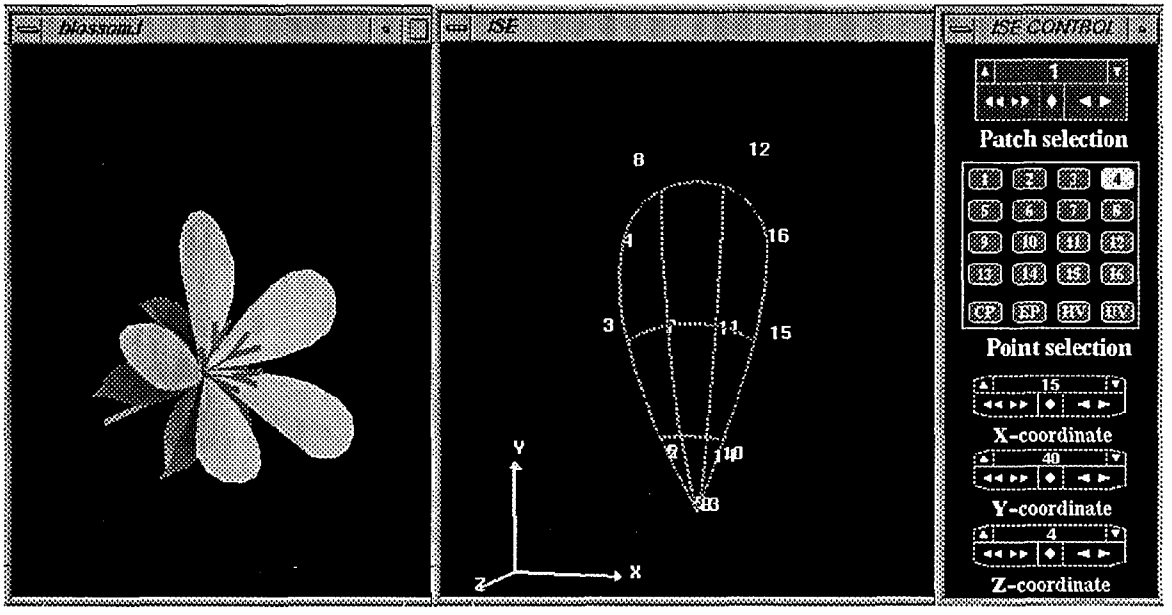


Figure 3.7: Apple blossom and interactive surface editor

positioned at a user-defined *end point* once the surface has been drawn.

The following L-system produces the apple blossom shown on the left side of Figure 3.7 in two derivation steps, given an angle increment of 18°.

$$\begin{aligned} \omega: &\quad \texttt{FFFFFB} \\ p_1: &\quad \texttt{B} \rightarrow \texttt{[S////S////S////S////S]} \\ p_2: &\quad \texttt{S} \rightarrow \texttt{[}\sim\texttt{C][}\sim\texttt{P][}\wedge\wedge\texttt{F[}-\texttt{F][}+\texttt{F]]} \end{aligned}$$

The F's in the axiom represent the blossom's stem, while the B represents a bud. In the first derivation step, production p_1 replaces the symbol B by five segments S separated by / symbols. In the second derivation step, production p_2 creates the three components of each segment, a calyx leaf [∼C], a petal [∼P], and a stamen [∧ ∧ F[−F][+F]]. During turtle interpretation, the predefined surfaces C, representing the leaf, and P, representing the petal, will be incorporated into the image. These surfaces were designed using the interactive surface editor shown on the right in Figure 3.7.

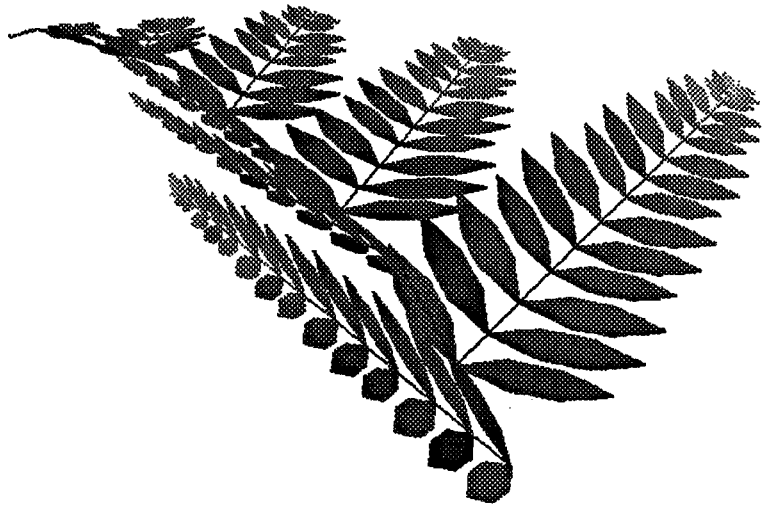


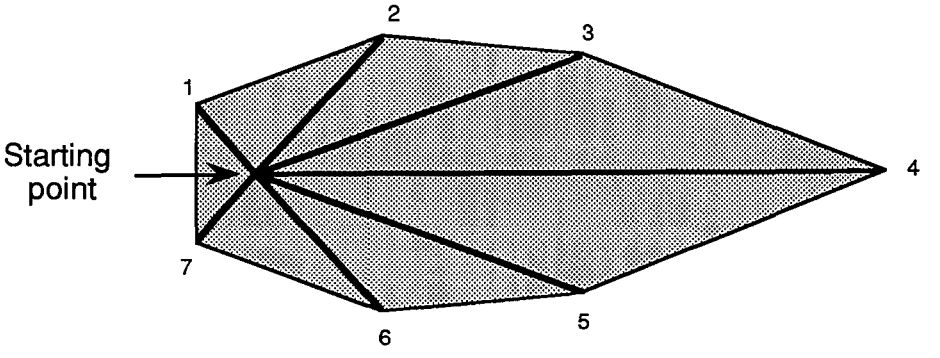
Figure 3.8: A model of a fern frond with polygonal leaflets

3.3.2 Developmental surfaces

Predefined surfaces do not "grow"; if a developmental sequence is required, surfaces representing individual stages of surface growth must be separately defined and incorporated into the model. An alternate approach is to allow the turtle to create polygons directly. The opening brace "{" and the closing brace "}" are introduced as commands that delimit the substring which determines the boundary of a polygon to be filled. When an opening brace is encountered during interpretation, an empty *list of vertices* representing the current polygon is created. Subsequently, whenever an F or f is interpreted, the resulting turtle position is appended as a vertex on the list. Interpretation of the closing brace causes the current polygon to be filled. Using this approach, L-system productions can be employed in a number of different ways to change the size and shape of a polygon over time.

The first possibility is to trace surface boundaries using the turtle and fill the resulting polygons, as in the L-system given below:

$$\begin{aligned} \omega &: \ \mathtt{L} \\ p_1 &: \ \mathtt{L} \rightarrow \{-\mathtt{FX}+\mathtt{X}-\mathtt{FX}-|-\mathtt{FX}+\mathtt{X}+\mathtt{FX}\} \\ p_2 &: \ \mathtt{X} \rightarrow \mathtt{FX} \end{aligned}$$



{[++++G .][++GG .][+GGG .][GGGGG .][-GGG .][--GG .][----G .]}
1 2 3 4 5 6 7

Figure 3.9: Surface specification using a branching structure as a framework. The numbers correspond to the order of vertex specification by the turtle.

Production p_1 defines leaf L as a closed planar polygon. Production p_2 increases the lengths of its edges linearly. This technique was used to model the leaflets on the fern branch in Figure 3.8. Leaflets appear in order of age with the youngest at the top. The branch exhibits a "phase effect" [143] in which a series of stages in a plant component's growth appear in the structure at the same time.

In practice, the tracing of polygon boundaries only produces acceptable effects for small, flat surfaces. In other cases it is more convenient to use a tree structure as a framework for a polygon. Vertices are specified by a sequence of turtle positions marked by the dot symbol (.). An example is given in Figure 3.9. The letter G has been used instead of F to indicate that the segments enclosed between the braces should not be interpreted as the edges of the constructed polygon. The numbers correspond to the order in which the turtle specifies the vertices.

In the techniques discussed so far, the turtle specifies the vertices of one polygon, then moves on to the next. Further flexibility in surface definition can be achieved by interleaving vertex specifications for different polygons. In order to accomplish this, the interpretation of braces is redefined as follows. A string containing nested braces is evaluated using two data structures, the list of vertices representing the

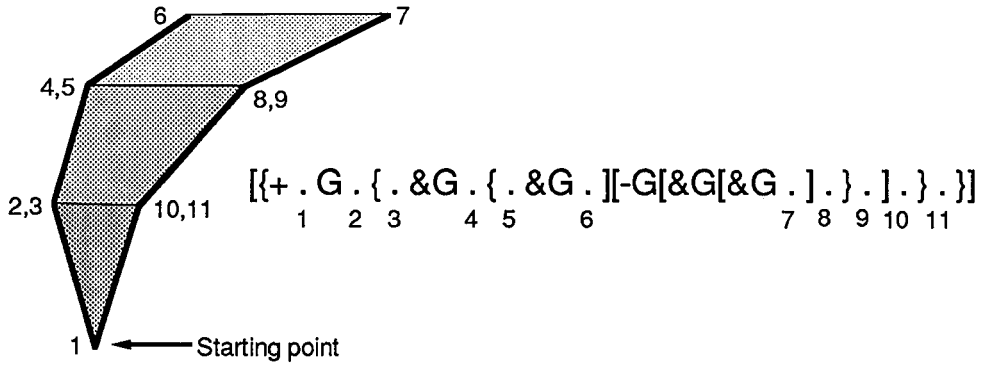


Figure 3.10: Surface specification using stacked polygons. The numbers correspond to the order of vertex specification by the turtle.

current polygon and a *polygon stack*. At the beginning of string interpretation, both structures are empty. The interpretation of an opening brace "{" initializes a new polygon list and pushes it onto the polygon stack. When the turtle encounters a closing brace "}" it pops the current polygon from the top of the stack and draws the polygon specified by its list of vertices. An example of string interpretation involving nested braces is given in Figure 3.10. This surface cannot be described using a single pair of braces, since methods for filling non-planar polygons are not well defined. Therefore, the figure is decomposed into three polygons connecting the following sets of vertices: $\{1, 2, 11\}$, $\{3, 4, 9, 10\}$, and $\{5, 6, 7, 8\}$. Note that it is necessary to have separate stacks for polygons and branches, as they operate independently. In this case, all three polygons start in one branch and are completed in another.

3.3.3 Surface specification commands

The turtle interpretation commands involved in surface creation can be summarized as follows.

$\sim$ Draw the surface identified by the letter immediately following the $\sim$ symbol at the current location of the turtle and with its orientation.

{ Create an empty current polygon and push it on the polygon stack.

`F` Move forward a step of length d and draw a line, then append the turtle's position to the current polygon.

`f` Move forward a step of length d without drawing a line, then append the turtle's position to the current polygon.

`G` Move forward a step of length d and draw a line, but do not append a vertex to the current polygon.

`g` Move forward a step of length d without drawing a line, but do not append a vertex to the current polygon.

`.` Append the turtle's position to the current polygon.

`}` Pop the current polygon from the stack and draw it using the specified vertices.

3.4 Special-purpose interpretation routines

Aside from the standard set of commands, it is often desirable to have the turtle perform special-purpose interpretation routines, which may be experimental in nature. These routines are compiled into the plant modelling program, and are typically used to increase the functionality of turtle interpretation. In order to facilitate this task, a "black-box" interface routine [61, Section 3.6] provides the link between the turtle and the new action, while isolating the user from the details of string handling and the remainder of the interpretation code.

The user specifies a new action by supplying code that performs the desired operations, along with a unique one- or two-letter identifier. In order to avoid ambiguities, one identifier cannot be a prefix of another. The turtle state attributes and drawing environment parameters are available for use and modification by the special purpose interpretation routines.

During interpretation, the black-box interface routine is called when an @ symbol is encountered in the string. The routine scans the string starting immediately after the @ and tests whether it matches any user-supplied identifier. If so, the appropriate

code is executed; if not, an error is reported. The black-box interface then returns control to the standard interpretation routines, where scanning is resumed at the first letter after those used to identify the black-box action. Examples of useful black-box functions are listed below.

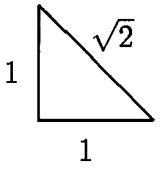
- `@C` draws a circle with radius equal to the turtle's linewidth at the turtle's current location, in the plane perpendicular to the viewing vector.
- `@S` draws a sphere with radius equal to the turtle's linewidth at the current location of the turtle.
- `@LF` decreases the line length attribute by a constant factor.
- `@V` rotates the turtle around its heading vector so that the left vector is horizontal and the y component of the up vector is positive.

Functions `@C` and `@S` extend the set of drawing primitives, while `@LF` modifies the drawing environment and `@V` modifies the turtle state.

Chapter Four

PARAMETRIC L-SYSTEMS

L-systems with turtle interpretation make it possible to generate a variety of objects, from abstract fractals to realistic images of flowering plants [61, 123, 119]. However, the discrete nature of the formalism imposes limitations on the user. One major problem can be traced to the restriction that all lines drawn by the turtle must be integer multiples of the unit segment. As a result, even such a simple figure as an isosceles right triangle



cannot be traced exactly, since the ratio of its hypotenuse length to the length of a side is the irrational number $\sqrt{2}$. Rational approximation of this line provides a limited solution, but requires the use of a large number of F commands, since the unit step must be the smallest common denominator of all line lengths in the modelled structure. This requirement can be of even greater consequence when modelling plant structures, which often exhibit a wide range of sizes, from tree trunk to tiny twig, and a large number of components in the entire organism. A similar argument applies for angles.

The constant length of the turtle step is also a problem when simulating the expansion of a structure over time. Since line segments are represented by sequences of F symbols, this expansion is modelled by the increase in their number from one

derivation step to the next. A function $f_G(n)$ associating the derivation step n with the number of symbols in a string is called a growth function. Rozenberg and Salomaa [133, pages 30–38] show that the growth function $f_G(n)$ of any D0L-system $G = \langle V, \omega, P \rangle$ is a combination of polynomial and exponential functions:

$$f_G(n) = \sum_{i=1}^{s} P_i(n)\rho_i^n \quad \text{for} \quad n \geq n_0, \tag{4.1}$$

where $P_i(n)$ denotes a polynomial with integer coefficients, ρ_i is a nonnegative integer, and n_0 is the total number of letters in the alphabet of G. Unfortunately, many growth processes observed in nature cannot be described by equation (4.1). The use of interactive L-systems extends the range of expressible growth functions, but still does not allow the specification of slower than logarithmic growth [148]. In addition, the IL-systems may require message passing schemes not found in real plants. It would be convenient for the user to be able to abstract from the details of modelling a particular growth function by simply specifying it mathematically. This form of expression could also be useful for a higher level plant design system.

Many processes that occur during plant development are continuous in nature. Examples include chemical reactions, diffusion of hormones, and the resulting distribution of concentrations. Generally, it is difficult to capture such phenomena using L-systems, since the obvious technique of discretizing continuous values may require a large number of quantization levels, yielding L-systems with hundreds of symbols and productions. To reduce this number, Lindenmayer proposed the association of numerical parameters with L-system symbols [90]. He illustrated this idea by referring to the continuous development of branching structures [52] and to the diffusion of chemical compounds in a nonbranching filament of *Anabaena catenula*. Both problems were revisited in later papers [28, 51, 50, 83].

These concepts inspired Baker, Herman, and Liu to develop a cellular iterative array simulator called CELIA [6, 7, 69, 71]. They took an ad hoc approach, in which ordinary productions were specified in a data file, while those with parameters were coded in a FORTRAN subroutine that had to be compiled and linked with the program. No formal definition was given describing the integration of the two techniques.

As a result, the data file specification was incomplete and could not serve as documentation of the underlying developmental processes.

The definitions for parametric L-systems with turtle interpretation found in the remainder of this chapter are based on the seminal ideas of Lindenmayer and are derived from the original formulation by Prusinkiewicz and Hanan [120]. They provide a practical basis for the modelling and visualization of a wide variety of organisms.

4.1 Parametric 0L-systems

Parametric L-systems operate on *parametric words*, which are strings of *modules* consisting of *letters* with associated *parameters*. The letters belong to an *alphabet* V, and the parameters belong to the set of *real numbers* $\Re$. A module with letter $A \in V$ and parameters $a_1, a_2, ..., a_n \in \Re$ is denoted by $A(a_1, a_2, ..., a_n)$. Every module belongs to the set $V \times \Re^*$, where $\Re^*$ is the set of all finite sequences of parameters.

The real-valued *actual* parameters appearing in the words correspond to *formal* parameters represented by names of variables in the specification of L-system productions. A letter with an associated sequence of formal parameters is called a *formal module*, and a sequence of formal modules is called a *formal parametric word*. If Σ is a set of formal parameters, then $C(\Sigma)$ denotes a *logical expression* with parameters from Σ, and $E(\Sigma)$ is an *arithmetic expression* with parameters from the same set. Both types of expressions consist of formal parameters and numeric constants, combined using the operators listed in Table 4.1 and parentheses, "(" and ")", for grouping. Standard rules for constructing syntactically correct expressions are observed, using the operator precedence and associativity presented in the table. Relational and logical expressions evaluate to zero for false and one for true. A logical statement specified as the empty string is assumed to have value one. The expressions can include calls to predefined functions from the following list: sine, cosine, tangent, arccos, arcsin, arctan, floor, ceiling, truncate to integer, absolute value, exponential, natural logarithm, and random. This choice of functions is dictated by their usefulness in applications. The sets of all correctly constructed logical and arithmetic expressions with parameters from Σ are noted $\mathcal{C}(\Sigma)$ and $\mathcal{E}(\Sigma)$.

Operator	Description	Associativity
$f()$	function call	left to right
$-$!	unary minus and logical negation	right to left
$\wedge$	exponentiation	right to left
* / %	multiplication, division, and remainder	left to right
$+$ $-$	addition and subtraction	left to right
< <= > >=	less than, less than or equal to, greater than, greater than or equal to	left to right
== !=	equality and inequality	left to right
&&	logical *and*	left to right
||	logical *or*	left to right

Table 4.1: Operator precedence and associativity. Operators on the same line have the same precedence and the rows are in order of decreasing precedence.

In order to model structures in which development is only controlled using lineage mechanisms, parameters can be incorporated into context-free L-systems, as in the following definition.

Definition 4.1 A *parametric 0L-system* is defined as an ordered quadruplet $G = \langle V, \Sigma, \omega, P \rangle$, where

- V is a nonempty set of letters called the *alphabet* of the system,
- Σ is the *set of formal parameters*,
- $\omega \in (V \times \Re^*)^+$ is a nonempty parametric word called the *axiom*, and
- $P \subset (V \times \Sigma^*) \times C(\Sigma) \times (V \times \mathcal{E}(\Sigma)^*)^*$ is a finite *set of productions.*

A production, $(\underline{a}, C, \underline{\chi})$ is usually noted as $\underline{a} : C \rightarrow \underline{\chi}$ where the formal module $\underline{a} \in V \times \Sigma^*$ is called the *predecessor*, the logical expression $C \in C(\Sigma)$ is called the *condition*, and the formal parametric word $\underline{\chi} \in (V \times \mathcal{E}(\Sigma)^*)^*$ is called the *successor*. If the condition is empty, the production can be noted $\underline{a} \rightarrow \underline{\chi}$. For a given production, it is assumed that a formal parameter can appear no more than once in the

predecessor, and all formal parameters in the condition and successor must appear in the predecessor.

An example of a production is given below.

$$\mathtt{A(t)} : \mathtt{t} > 5 \rightarrow \mathtt{B(t+1)CD(t} \wedge 0.5, \mathtt{t-2}) \tag{4.2}$$

It has predecessor $\mathtt{A(t)}$, condition $\mathtt{t} > 5$ and successor $\mathtt{B(t+1)CD(t} \wedge 0.5, \mathtt{t-2})$.

Definition 4.2 A production p_i *matches* a module in a parametric word if the following conditions are met:

- the letter in the module and the letter in the production predecessor are the same,
- the number of actual parameters in the module is equal to the number of formal parameters in the production predecessor, and
- the condition evaluates to *true* if the actual parameter values are substituted for (or bound to) the formal parameters according to their relative position in the module and predecessor.

A matching production can be *applied* to the module, creating a string of modules specified by the production successor with expressions being evaluated to produce actual parameters. If no production $p \in P$ matches a given module from the string, then the module is replaced by itself.

For example, production (4.2) matches the module $\mathtt{A(9)}$, since the letter $\mathtt{A}$ in the module is the same as in the production predecessor, there is one actual parameter in the module $\mathtt{A(9)}$ and one formal parameter in the predecessor $\mathtt{A(t)}$, and the logical expression $\mathtt{t} > 5$ is true for $\mathtt{t}$ equal to 9. The result of the application of this production is the parametric word $\mathtt{B(10)CD(3,7)}$.

A parametric 0L-system is deterministic if and only if no two productions can match the same module in a string. This requirement will be satisfied if for every group of productions that have predecessors with the same letter and the same

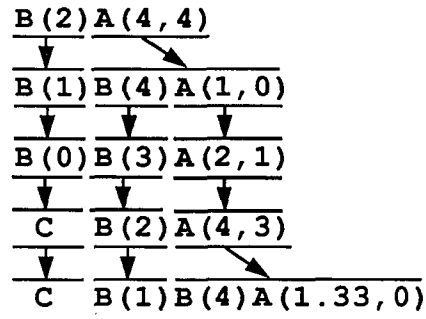


Figure 4.1: The initial sequence of strings generated by the parametric L-system specified in equation (4.3)

number of formal parameters, no two conditions evaluate to true, for every possible combination of parameter values. In practice it is often convenient to enforce deterministic operation of an L-system by ordering the set of productions and by applying the first matching production in the list, as in the following definition.

Definition 4.3 A *parametric D0L-system* is defined as an ordered quadruplet $G = \langle V, \Sigma, \omega, P \rangle$, where the alphabet V, the formal parameters Σ, and the axiom ω are defined as in Definition 4.1, and $P : \{1, 2, ..., N\} \subset (V \times \Sigma^*) \times \mathcal{C}(\Sigma) \times (V \times \mathcal{E}(\Sigma)^*)^*$ is a finite, *ordered* set of productions. A production p_i will *match* a module in a string, if the conditions of Definition 4.2 apply and there is no production $p_j \in P$ with $j < i$ that also meets those conditions.

A derivation in a parametric L-system is defined below, in the same manner as for standard L-systems.

Definition 4.4 If a module a produces a parametric word χ as the result of a production application in a parametric L-system G, we write $a \mapsto \chi$. Given a parametric word $\mu = a_1 a_2 ... a_m$, we say that the word $\nu = \chi_1 \chi_2 ... \chi_m$ is *directly derived* from (or *generated* by) μ, and write $\mu \Longrightarrow \nu$, if and only if $a_i \mapsto \chi_i$ for all $i = 1, 2, ..., m$. A parametric word ν is generated by G in a *derivation of length* n if there exists a sequence of words $\mu_0, \mu_1, ..., \mu_n$ such that $\mu_0 = \omega$, $\mu_n = \nu$ and $\mu_0 \Longrightarrow \mu_1 \Longrightarrow ... \Longrightarrow \mu_n$.

An example of a parametric D0L-system is given below.

$$\begin{array}{lllll} \omega: & \mathtt{B(2)A(4,4)} & & & \\ p_1: & \mathtt{A(x,y)} & \mathtt{:y<3} & \rightarrow & \mathtt{A(x*2,x+y)} \\ p_2: & \mathtt{A(x,y)} & \mathtt{:y>=3} & \rightarrow & \mathtt{B(x)A(x/y,0)} \\ p_3: & \mathtt{B(x)} & \mathtt{:x<1} & \rightarrow & \mathtt{C} \\ p_4: & \mathtt{B(x)} & \mathtt{:x>=1} & \rightarrow & \mathtt{B(x-1)} \end{array} \tag{4.3}$$

The words obtained in the first few derivation steps are shown in Figure 4.1.

4.2 Parametric IL-systems

Productions in parametric D0L-systems are context-free and provide a mechanism for the simulation of information flow controlled by lineage. In order to model endogenous control of development, parameters are incorporated into non-bracketed interactive L-systems. In the non-parametric case, two types of interactive L-systems were distinguished: IL-systems, where the left and right contexts were of constant lengths, and I'L-systems, where no such restriction applied. In the parametric case, this distinction is not made. The more convenient term IL-system is used to represent the general case, where contexts of different lengths coexist in the same parametric L-system.

Definition 4.5 A *parametric IL-system* is an ordered quadruplet $G = \langle V, \Sigma, \omega, P \rangle$, where

- V is a nonempty set of letters called the *alphabet* of the system,
- Σ is the *set of formal parameters*,
- $\omega \in (V \times \Re^*)^+$ is a nonempty parametric word called the *axiom*, and
- $P \subset (V \times \Sigma^*)^* \times (V \times \Sigma^*) \times (V \times \Sigma^*)^* \times \mathcal{C}(\Sigma) \times (V \times \mathcal{E}(\Sigma)^*)^*$ is a finite *set of productions.*

A production $(\underline{\eta_l}, \underline{a}, \underline{\eta_r}, C, \underline{\chi})$ is usually noted $\underline{\eta_l} < \underline{a} > \underline{\eta_r} : C \rightarrow \underline{\chi}$ where

- the formal module $\underline{a} \in V \times \Sigma^*$ is called the *strict predecessor*
- the formal parametric words $\underline{\eta_l} \in (V \times \Sigma^*)^*$ and $\underline{\eta_r} \in (V \times \Sigma^*)^*$ are called the *left context* and the *right context*, respectively,
- the triplet $(\underline{\eta_l}, \underline{a}, \underline{\eta_r})$ is called the *predecessor*,
- the logical expression $C \in \mathcal{C}(\Sigma)$ is called the *condition*, and
- the formal parametric word $\underline{\chi} \in (V \times \mathcal{E}(\Sigma)^*)^*$ is called the *successor*.

As in the context free case, a formal parameter can appear no more than once in the predecessor of any production, and all formal parameters appearing in the condition and successor must appear in the predecessor.

A production with its left context empty can be written as $\underline{a} > \underline{\eta_r} : C \rightarrow \underline{\chi}$; similarly, a production with its right context empty can be written as $\underline{\eta_l} < \underline{a} : C \rightarrow \underline{\chi}$. A production with an empty condition can be written as $\underline{\eta_l} < \underline{a} > \underline{\eta_r} \rightarrow \underline{\chi}$. Analogous notations apply for any combination of an empty condition and empty contexts.

Definition 4.6 A context-sensitive production $p_i : \underline{\eta_l} < \underline{a} > \underline{\eta_r} : C \rightarrow \underline{\chi}$ *matches* a module at position s in a parametric word μ if the following conditions are met:

- the letters and numbers of parameters in the sequence of modules $\underline{\eta_l}, \underline{a}, \underline{\eta_r}$ forming the predecessor are the same as the letters and numbers of parameters in the modules of μ at positions $s - l'$ to $s + r'$, where l' is the length of $\underline{\eta_l}$ and r' is the length of $\underline{\eta_r}$, and
- the condition C evaluates to *true* if the actual parameter values are substituted for the formal parameters according to their relative positions in the modules of μ and in the predecessor.

A matching production can be *applied* to the module at position s, creating a string of modules specified by the production successor with expressions being evaluated to produce actual parameters. If no production $p \in P$ matches a given module at position s in the string, then the module is replaced by itself.

An example of a context-sensitive production is given below:

$$\mathtt{A(w,x)} < \mathtt{B(y)} > \mathtt{C(z)} : \mathtt{x+y+z} > 10 \rightarrow \mathtt{E((x+y)/2)F((y+z)/2)}$$

It can be applied to the module B(5) that appears in a parametric word

$$\cdots \mathtt{A(3,4)B(5)C(6)} \cdots \tag{4.4}$$

since the sequence of letters A, B, C in this word and in the production are the same, the numbers of formal parameters and actual parameters in the corresponding modules coincide, and the condition $4 + 5 + 6 > 10$ is true. As a result of the production application, the module B(5) will be replaced by a pair of modules E(4.5)F(5.5). Naturally, the modules A(3, 4) and C(6) will be replaced by other productions in the same derivation step.

For the same practical reasons as outlined in the discussion preceding Definition 4.3 on page 48, the productions are ordered in the following definition of parametric deterministic IL-systems.

Definition 4.7 A *parametric DIL-system* is defined as an ordered quadruplet $G = \langle V, \Sigma, \omega, P \rangle$, where

- the alphabet V, the formal parameters Σ, and the axiom ω are defined as in Definition 4.5, and
- $P : \{1, 2, ..., N\} \rightarrow (V \times \Sigma^*)^* \times (V \times \Sigma^*) \times (V \times \Sigma^*)^* \times \mathcal{C}(\Sigma) \times (V \times \mathcal{E}(\Sigma)^*)^*$ is a finite, *ordered* set of productions.

A production p_i will *match* a module in a string, if the conditions of Definition 4.6 apply and there is no production $p_j \in P$ with $j < i$ that also meets these conditions.

A derivation in a parametric IL-system proceeds in the same manner as for parametric 0L-systems (Definition 4.4). An example of a parametric DIL-system is given below.

$$\begin{array}{lllllll}
\omega: & \mathtt{B(1)A(1,0)} & & & & & \\
p_1: & \mathtt{B(w)} < & \mathtt{A(x,y)} & : \mathtt{w <= 1} & \rightarrow & \mathtt{B(x)A(x+y,y+1)} \\
p_2: & & \mathtt{B(x)} & : \mathtt{x < 1} & \rightarrow & \mathtt{C} \\
p_3: & & \mathtt{B(x)} & & \rightarrow & \mathtt{B(x-1)}
\end{array} \tag{4.5}$$

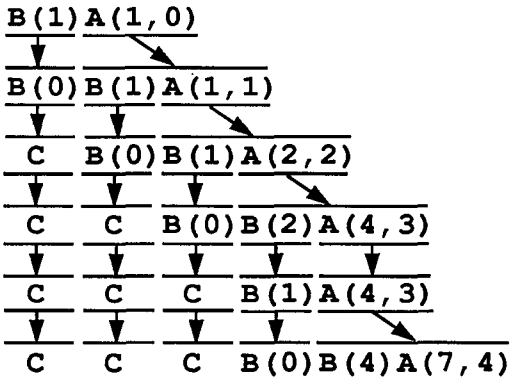


Figure 4.2: The initial sequence of strings generated by parametric DIL-system (4.5)

The words obtained in the first few derivation steps are shown in Figure 4.2. In the first derivation step, the module B(1) is matched by production p_2, and not by p_3, as a consequence of the ordering of the productions. Production p_1 is applied whenever a module with letter A has a module B to its left with a parameter value less than or equal to 1. Note that identity productions are applied to all modules C and to modules A when their left context does not match B(w) with w <= 1.

4.3 Stochastic parametric IL-systems

The introduction of stochastic capabilities to parametric IL-systems is useful for the same reasons as discussed for L-systems without parameters in Section 2.3 on page 20. Probability factors can be specified as expressions combining constants and formal parameters. This allows the probability distributions to change over time.

Definition 4.8 A *stochastic parametric IL-system* is an ordered quadruplet $G = \langle V, \Sigma, \omega, P \rangle$, where the alphabet V, the set of formal parameters Σ, and the axiom ω are defined as for a parametric IL-system (Definition 4.5) and the finite set of productions is described as $P \subset (V \times \Sigma^*)^* \times (V \times \Sigma^*) \times (V \times \Sigma^*)^* \times \mathcal{C}(\Sigma) \times (V \times \mathcal{E}(\Sigma)^*)^* \times \mathcal{E}(\Sigma)$. If a sextuple $(\underline{\eta_l}, \underline{a}, \underline{\eta_r}, C, \underline{\chi}, \psi)$ is a production, it is noted $\underline{\eta_l} < \underline{a} > \underline{\eta_r} : C \rightarrow \underline{\chi} : \psi$, where:

- the predecessor $(\underline{\eta_l}, \underline{a}, \underline{\eta_r})$, the *condition* C, and the *successor* $\underline{\chi}$ are defined as in the non-stochastic case (Definition 4.5), and
- $\psi \in \mathcal{E}(\Sigma)$ is called the *probability factor expression.*

Within a given production, a formal parameter can appear no more than once in the predecessor, and all formal parameters appearing in the condition, successor, and probability factor expression must appear in the predecessor. A production p_i will *match* a module in a string under the same conditions as in Definition 4.6.

Definition 4.9 Let $\hat{P}(\mu, s) \subset P$ denote the subset of productions from P which match word μ at position s. If no production in P matches μ at a given position, then the module at position s is replaced by itself. Otherwise, let the *probability factor* $\pi(p_i, \mu, s) \geq 0$ be the value of the probability factor expression ψ in a production $p_i \in \hat{P}(\mu, s)$, assuming that the actual parameter values from the matched string have been substituted for the formal parameters in the predecessor. The derivation $\mu \Rightarrow \nu$ is a *stochastic derivation* in G if for each position s in the word μ the probability of applying production $p_i \in \hat{P}(\mu, s)$ is equal to

$$prob(p_i) = \frac{\pi(p_i, \mu, s)}{\sum_{p_k \in \hat{P}(\mu,s)} \pi(p_k, \mu, s)} \tag{4.6}$$

assuming that at least one probability factor $\pi(p_i, \mu, s)$ is greater than zero. A simple example of a stochastic parametric L-system is given below.

$$\begin{array}{lllll} \omega: & \texttt{A(1)} & & & \\ p_1: & \texttt{A(a)} & : \texttt{a} > 3 & \rightarrow \texttt{B} & : \texttt{a}/2 \\ p_2: & \texttt{A(a)} & & \rightarrow \texttt{BA(a}+1\texttt{)} & : 1 \end{array}$$

In the first derivation step only production p_2 matches the module A(1) in the axiom. It will be selected with probability one, resulting in the string BA(2). In the next and all subsequent steps, the B modules will be replaced by themselves using an implied identity production. The modules with letter A will continue to match production p_2 and be replaced as above, until production p_2 has been applied three times. At this

point the string will be BBBA(4). In the next step both productions p_1 and p_2 will match the module A(4). The probabilities of production selection will be 2/(2+1) = 2/3 for p_1 and 1/(2+1) = 1/3 for p_2. If production p_1 is selected, the resulting string will be BBBBB, and no more changes will occur. If production p_2 is selected again, the resulting string will be BBBBA(5). In the next step the probability of applying production p_1 will be 2.5/(2.5+1) = 5/7 and the probability of applying production p_2 will be 1/(2.5+1) = 2/7. As the derivation proceeds, the probability of selecting production p_1 will increase steadily until p_1 is finally applied. No more changes will take place after this, as there will only be B modules in the string.

4.4 Modelling branching structures

Lindenmayer's notion of bracketed strings can be applied to parametric words in a straightforward way. The formalism of parametric L-systems is extended to branching structures using the branch delimiters "[" and "]" as in non-parametric bracketed L-systems (Section 2.4). During the derivation process, the procedures established in Sections 2.4 and 2.5 are applied to search for context. For example, production

$$p_1 : \mathtt{AB(w)} < \mathtt{D(x,y)} > \mathtt{F(z)} : \mathtt{w+x <= z} \rightarrow \mathtt{J(y)[K((w+x+z)/3)]}$$

matches module D(1, 2) in the parametric string AB(0.4)[C(1)][D(1, 2)[E(3)]F(4)]G. The letters and number of parameters in the left context of the production match those in the string AB(0.4)[C(1)][with the substring [C(1)][being skipped. The letters and number of parameters in the right context of the production match those in the string [E(3)]F(4) with the substring [E(3)] being skipped. The condition w + x <= z is true when the formal parameters are assigned the values w = 0.4, x = 1, y = 2, and z = 4. Application of the production will result in the parametric word J(0.4)[K(1.8)].

4.5 Turtle interpretation of parametric words

The basic concept of turtle interpretation as presented in Chapter 3 is not changed by the introduction of parameters, but the turtle can be manipulated much more flexibly. In general, the value of the first parameter quantifies the extent to which

the appropriate turtle state attribute is changed. If a symbol is not followed by any parameter, default values specified in the drawing environment are used as in the non-parametric case. The symbols interpreted by the turtle that are affected by the introduction of parameters are listed below.

`F`(a) Move forward a step of length a. The position of the turtle changes to (x', y', z'), where $x' = x + a\vec{H}_x$, $y' = y + a\vec{H}_y$ and $z' = z + a\vec{H}_z$. A line segment is drawn between points (x, y, z) and (x', y', z'). If a polygon is open, append the resulting position as a vertex of the current polygon.

`f`(a) Move forward a step of length a without drawing a line, then append the turtle's current position to the current polygon.

`G`(a) Move forward a step of length a and draw a line, but do not append a vertex to the current polygon.

`g`(a) Move forward a step of length a without drawing a line, but do not append a vertex to the current polygon.

$+(a)$ Rotate around $\vec{U}$ by an angle of a degrees. If a is positive, the turtle is turned to the left.

$-(a)$ Rotate around $\vec{U}$ by an angle of a degrees. If a is positive, the turtle is turned to the right.

$\&(a)$ Rotate around $\vec{L}$ by an angle of a degrees. If a is positive, the turtle is pitched down.

$\wedge(a)$ Rotate around $\vec{L}$ by an angle of a degrees. If a is positive, the turtle is pitched up.

$/(a)$ Rotate around $\vec{H}$ by an angle of a degrees. If a is positive, the turtle is rolled to the right.

$\backslash(a)$ Rotate around $\vec{H}$ by an angle of a degrees. If a is positive, the turtle is rolled to the left.

$,(a)$ Increase the value of the current index into the colour map by a.

$;(a)$ Decrease the value of the current index into the colour map by a.

$\#(a)$ Increase the value of the current line width by a.

$!(a)$ Decrease the value of the current line width by a.

$''(a)$ Increase the value of the current elasticity factor by a.

$'(a)$ Decrease the value of the current elasticity factor by a.

$\sim$ Draw the surface identified by the module immediately following the $\sim$ at the turtle's current location and orientation. The first parameter of the identifying module is interpreted as a scaling factor if it is present, otherwise no scaling is applied.

@ Pass the string to the black-box interface routine. Modules identifying the desired function are read from the string and the appropriate special-purpose routine is executed. The routines may use parameters in any way. For the circle $@\mathtt{C}(r)$ and sphere $@\mathtt{S}(r)$, the parameter r is interpreted as the radius.

Symbols from Chapter 3 not listed above are interpreted as outlined in that chapter; parameters have no effect on their interpretation. It should be noted that symbols $+$, $-$, $\wedge$, and $/$ are used both as letters of the alphabet V and as operators in logical and arithmetic expressions. Their meaning depends on the context in which they are found.

The introduction of parameters increases the range of images that can be visualized easily using turtle interpretation. The simple problem of drawing the isosceles right triangle, proposed at the beginning of this chapter, can now be solved. The isosceles triangle on page 43 can be recreated, to the limit of machine precision, by interpretation of the parametric string resulting from one derivation of the following L-system.

$$\omega : \mathtt{T(1)}$$
$$p_1 : \mathtt{T(x)} \rightarrow \mathtt{F(x)} - (135)\mathtt{F}(2 \wedge 0.5 * \mathtt{x}) - (135)\mathtt{F(x)}$$

Chapter Five

APPLICATIONS

This chapter presents a number of examples that illustrate the utility of parametric L-systems for the visualization of processes and structures in botany. They are organized to provide a progression of ideas, revealing the advantages of the parametric extension over the standard L-system formalism. These advantages include:

- the ability to express a range of precise lengths and angles, not restricted to multiples of a discrete base unit,
- the ability to use arithmetic expressions to capture arbitrary growth functions,
- the ability to capture the branching structure of trees, extending the range of L-system applications,
- the specification of parameters facilitating interactive manipulation of the models,
- the presence of numeric parameters capturing module attributes may be more intuitive than encoding them as states represented by letters,
- the ability to control the extent of changes from one derivation step to the next, producing smooth animations that simulate time lapse photography,
- the ability to model continuous phenomena such as diffusion, and
- the applicability of the parametric L-system formalism to standard computational problems, and as a model of parallel computation.

Each example starts from an exposition of the problem, followed by a presentation of the L-system and a discussion of the role of parameters in the model. The L-systems are specified using the notation introduced in Chapter 4, with the addition of three statements: `#define` statements assign values to constants, `#include` statements list identifiers of the bicubic surfaces to be incorporated into the model, and `#ignore` statements list modules to be ignored while context matching. Symbols `/*` and `*/` enclose comments.

The `#define` statements clearly identify key parameters of the model, which can therefore be accessed and modified easily using a text editor or through a graphical interface such as a control panel [103]. The ability to manipulate parameters interactively and receive immediate visual feedback is one of the primary advantages for users of parametric L-systems. For example, consider a situation in which the user wishes to change a branching angle in a model. In order to effect such a change in a standard L-system, it would be necessary to modify one or more productions by adding or removing letters representing rotational commands. In parametric L-systems, on the other hand, the numeric parameter associated with a rotational command module provides a continuum of values from which to choose. The interactive manipulation of L-systems greatly enhances their usefulness as an exploratory tool.

5.1 *Anabaena catenula*

In this section, a model describing the vegetative growth of *Anabaena catenula* is presented. This example illustrates the use of parameters for geometric interpretation, and the model simplification that may be achieved by using parameters to encode module differences. By translating the model presented in Equation 2.1 on page 16

Figure 5.1: A simple visualization of the parametric *Anabaena* model

into a parametric L-system, the following L-system is obtained.

L-system 1: A parametric model of *Anabaena catenula*

```
#define A       1           /* Cell type A */
#define B       2           /* Cell type B */
#define W       .5          /* Width of both cell types */
#define AS      2           /* Size of cell type A */
#define BS      1           /* Size of cell type B */
#define R       1           /* Orientation to the right */
#define L       -1          /* Orientation to the left */
```

ω : !(W)F(BS,B,R)

p_1 : F(s,t,o) : t == A && o == R → F(AS,A,L)F(BS,B,R)

p_2 : F(s,t,o) : t == A && o == L → F(BS,B,L)F(AS,A,R)

p_3 : F(s,t,o) : t == B && o == R → F(AS,A,R)

p_4 : F(s,t,o) : t == B && o == L → F(AS,A,L)

This L-system contains all the geometric information necessary to create a visualization of the development of a filament. The axiom sets the width of the cells using

the !(W) module. A single cell is represented by the module F(s, t, o) where the three parameters represent the length, type, and orientation of the cell, respectively. Note that the length must be the first parameter for correct turtle interpretation of the line-drawing command F. The ordering of the remaining parameters is arbitrary, but must be consistent throughout the production set. The constant values assigned to the type and orientation are defined for convenience, while the length and width are tuned to produce the desired visualization.

The axiom incorporates a single cell of type B oriented to the right, represented by the module F(BS, B, R) where BS = 1, B = 2, and R = 1. The following sequence of parametric words is generated.

!(.5)F(1, 2, 1)
!(.5)F(2, 1, 1)
!(.5)F(2, 1, −1)F(1, 2, 1)
!(.5)F(1, 2, −1)F(2, 1, 1)F(2, 1, 1)
!(.5)F(2, 1, −1)F(2, 1, −1)F(1, 2, 1)F(2, 1, −1)F(1, 2, 1)
!(.5)F(1, 2, −1)F(2, 1, 1)F(1, 2, −1)F(2, 1, 1)F(2, 1, 1)F(1, 2, −1)F(2, 1, 1)F(2, 1, 1)
...

In the first time step, production p_3 increases the module's length from BS = 1 to AS = 2 and changes its type to A = 1, while maintaining the module's orientation. In the second time step, production p_1 divides the cell into a cell F(2,1,-1) of type A and a cell F(1,2,1) of type B. An analogous life cycle can be seen for a cell of type B with the opposite orientation. A visualization of this sequence can be seen in Figure 5.1, where the gaps between cells were modelled by the addition of a module f(.25) between the two F modules in both p_1 and p_2. Dark coloured cells are oriented to the left, light coloured cells to the right.

The distinction between type A and B cells reflects their size difference. With parametric L-systems, we can consider both cells as differing only in age. Cells of type B are younger, and therefore shorter, versions of type A cells. The following

simplified L-system results.

L-system 2: A refined model of *Anabaena catenula*

```
#define W     .5                    /* Width of cells */
#define AS    2                     /* Size of cell type A */
#define BS    1                     /* Size of cell type B */
#define R     1                     /* Orientation to the right */
#define L     −1                    /* Orientation to the left */
ω :  !(W)F(BS,R)
p1 :  F(s,o)  :  s == AS && o == R  →  F(AS,L)F(BS,R)
p2 :  F(s,o)  :  s == AS && o == L  →  F(BS,L)F(AS,R)
p3 :  F(s,o)  :  s == BS            →  F(AS,o)
```

A module representing a cell now only has parameters for length and orientation. The cell type is encoded in the length parameter; a cell of length BS is a type B cell and a cell of length AS is a type A cell. Note that productions p_3 and p_4 from the original L-system have been replaced by a single production p_3 that increases the length parameter from BS to AS. The module's orientation is maintained using the formal parameter o.

Starting from the axiom !(W)F(BS,R), the following sequence of parametric words is generated:

```
!(.5)F(1,1)
!(.5)F(2,1)
!(.5)F(2,−1)F(1,1)
!(.5)F(1,−1)F(2,1)F(2,1)
!(.5)F(2,−1)F(2,−1)F(1,1)F(2,−1)F(1,1)
!(.5)F(1,−1)F(2,1)F(1,−1)F(2,1)F(2,1)F(1,−1)F(2,1)F(2,1)
...
```

The arrangement of cell lengths and orientations corresponds to the previous developmental sequence. This example has shown how geometric information and cell states can be captured using parameters and how careful encoding of information can serve to reduce model complexity.

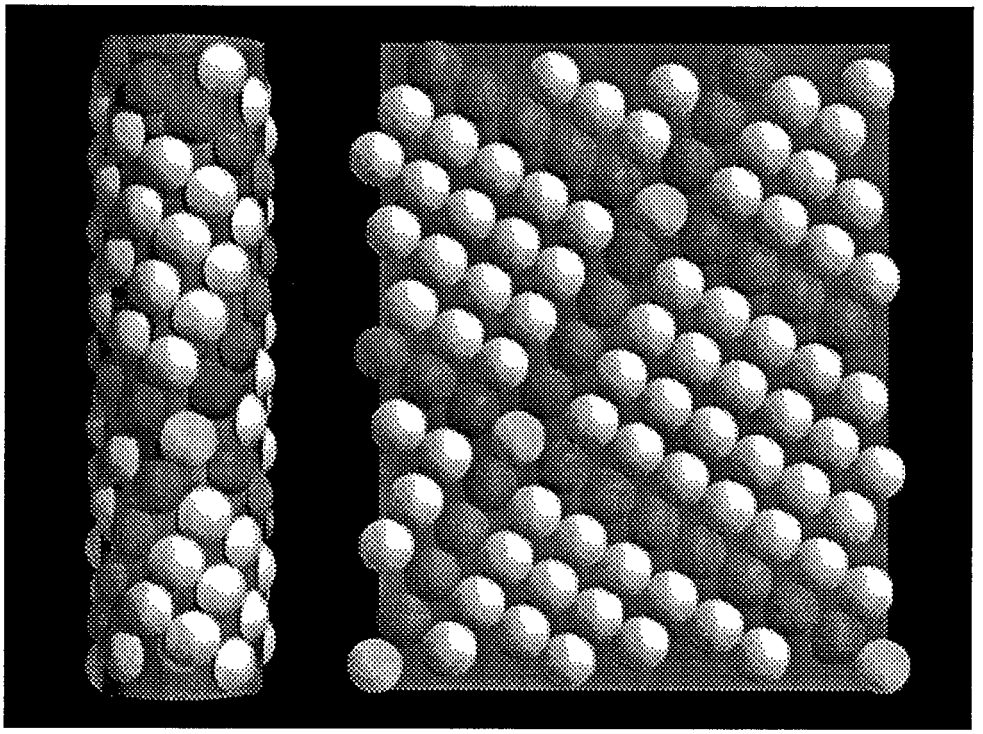


Figure 5.2: Model of phyllotaxis on the surface of a cylinder

5.2 Spiral phyllotaxis

The term *phyllotaxis* literally means leaf arrangement, but it is used in a broader sense to denote the regular arrangement of any lateral organs in plants [37]. Parametric L-systems make it possible to generate these patterns without resorting to special-purpose programs, such as those previously used by Fowler *et al* [48]. Parameters are used to capture the precise values of constants required in both the cylindrical and planar models presented in this section. In addition, the planar model illustrates the use of mathematical expressions modifying the values of parameters over time.

5.2.1 The cylindrical model

Spiral phyllotactic patterns evident in elongated organs can be described by models which position components along a helix on the surface of a cylinder. These

patterns are generally characterized by the following formulae:

$$\phi = n * \alpha \qquad r = const \qquad H = n * h\,, \tag{5.1}$$

where:

- n is the ordering number of a component, counting from the bottom of the cylinder,
- ϕ, r and H are the cylindrical coordinates of component n,
- α is a constant *divergence* angle between the position vectors of two consecutive components, and
- h is the vertical displacement between two consecutive components, measured along the main axis of the cylinder.

An extensive mathematical theory developed by van Iterson [81] and Erickson [37] describes the relationships between the values of r, α, and h corresponding to various classes of phyllotactic patterns. For example, the values incorporated in L-system 3 result in the pattern of disk-shaped components shown in Figure 5.2.

L-system 3: `A cylindrical model of phyllotaxis`

```
#define a    137.5  /* Divergence angle */
#define h    35.3   /* Vertical displacement */
#define r    500    /* Component offset from the main axis */
#define s    50     /* Scaling factor */
#include D          /* Disk shape specification */
ω : A
p1 : A → [&(90)f(r)~D(s)]f(h)/(a)A
```

The operation of this L-system simulates the natural process of *subapical development* characterized by sequential production of consecutive plant modules by the growing tip or *apex* of the plant or organ. The apex `A` produces internodes `f(h)` along the main axis of the modeled structure. Associated with each internode is a disk `~D(s)` placed at a distance `r` from the axis. This offset is achieved by moving the disk

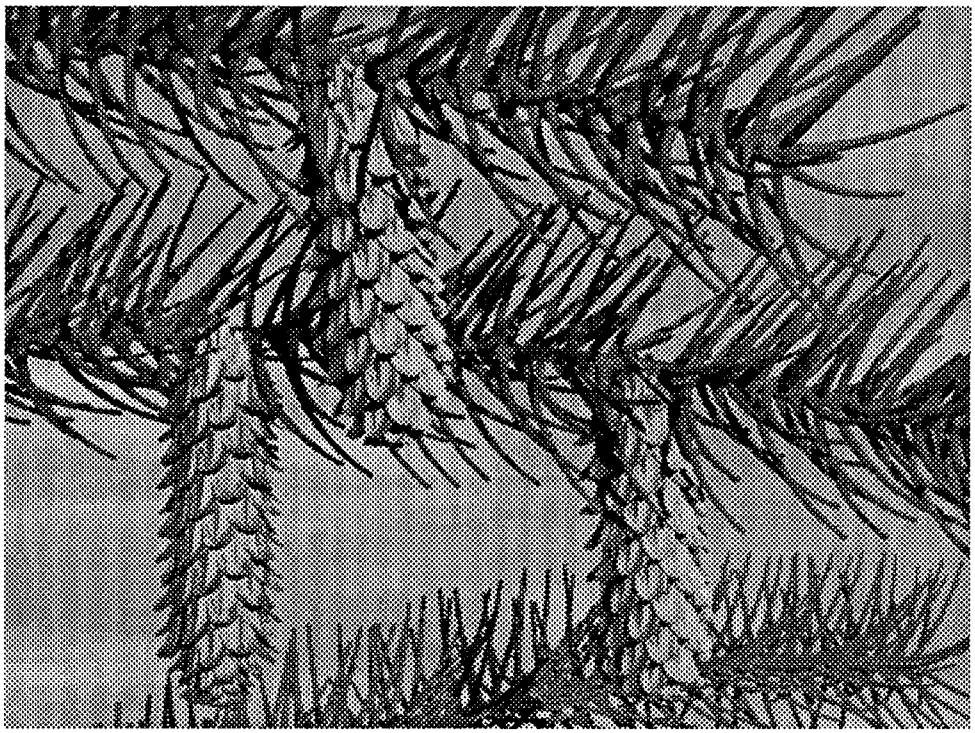


Figure 5.3: Spruce cones ©1990 D. R. Fowler and J. Hanan

away from the axis using the module f(r), positioned at a right angle with respect to the axis by module &(90). The spiral distribution of disks is due to the module /(a), which rotates the apex around its own axis by the divergence angle a in each derivation step.

The model of a spruce branch shown in Figure 5.3 and Plate C.1 displays spiral patterns in both the cones and the arrangement of the needles.

5.2.2 The planar model

The cylindrical model for phyllotaxis does not apply to flat structures, for example those found in many composite inflorescences. In order to describe them, Vogel [149] proposed a planar phyllotaxis model expressed by the formulae:

$$\phi = n * \alpha \qquad\qquad r = c\sqrt{n},$$

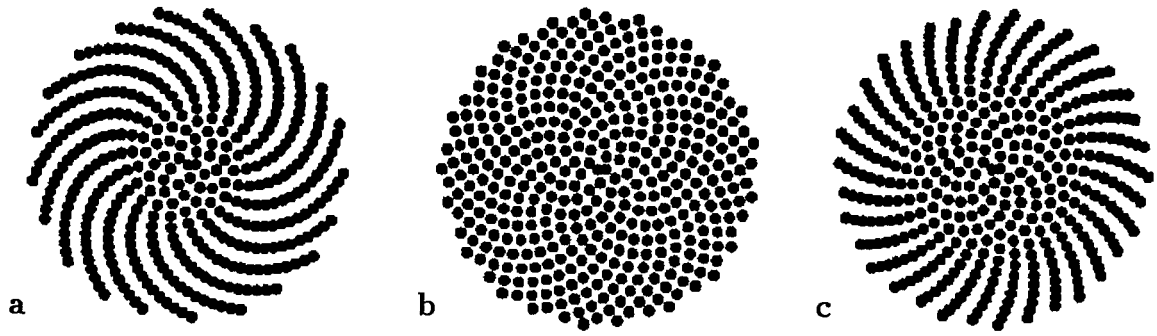


Figure 5.4: Generating phyllotactic patterns on a disk. These three patterns differ only by the value of the divergence angle a, equal to (a) 137.3°, (b) 137.5° (the correct value), and (c) 137.6°.

---

where n is the ordering number of a component, counting outward from the structure's center (the reverse order of floret age in a real plant), ϕ and r are the polar coordinates of component n, α is a constant divergence angle between the position vectors of two consecutive components (usually equal to 137.5°), and c is a scaling factor. This layout of components can be obtained in a straightforward way using the following L-system:

L-system 4: A planar model of phyllotaxis

```
#define a    137.5  /* divergence angle */
#define s    50     /* scaling factor */
#include D          /* disk shape specification */
ω : A(0)
p1 : A(n) → +(a)[f(n ∧ 0.5)~D(s)]A(n + 1)
```

Vogel's model is very sensitive to the value of the divergence angle a, as shown in Figure 5.4.

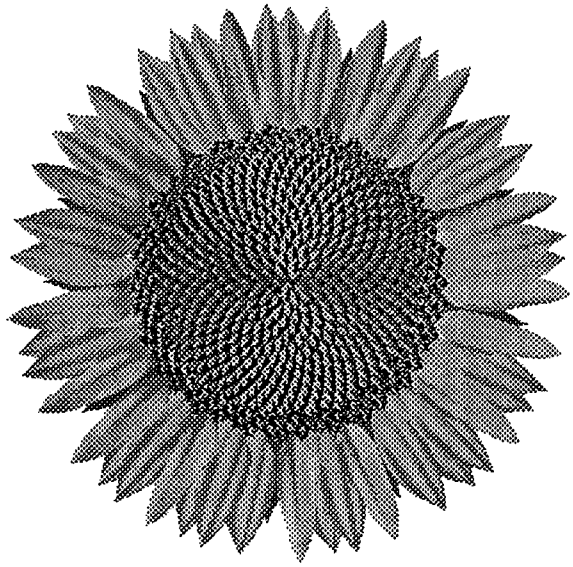


Figure 5.5: Model of a sunflower head

An example of the application of Vogel's model to image synthesis is given in Figure 5.5 and Plate C.2. The following L-system models the sunflower head.

L-system 5: `Sunflower head`

```
#include S                    /* Seed shape */
#include R                    /* Ray floret shape */
#include M N O                /* Petal shapes */
ω  :  A(0)
p1 :  A(n)                                    →  +(137.5)[f(n ∧ 0.5)C(n)]A(n + 1)
p2 :  C(n)  :  n <= 440                       →  [;(1)~S]
p3 :  C(n)  :  440 < n && n <= 565            →  [;(2)~R]
p4 :  C(n)  :  565 < n && n <= 580            →  [;(3)~M]
p5 :  C(n)  :  580 < n && n <= 595            →  [;(4)~N]
p6 :  C(n)  :  595 < n                        →  [;(6)~O]
```

The layout of components is specified by production p_1, similar to that of L-system 4. Productions p_2 to p_6 determine colours and shapes of the components as a function of the parameter of the module `C` which counts the derivation step number. The entire structure shown in Figure 5.5 was generated in 630 steps.

Figure 5.6: Sunflower field ©1990 D. R. Fowler, N. Fuller, J. Hanan, and A. Snider

Other extensions to the basic model consist of varying organ orientation in space and changing their altitude from the plane of the head as a function of n. For example, the sunflower plants in Figure 5.6 and Plate C.3 have flowers in four developmental stages: buds, young flowers starting to open, open flowers and older flowers with drooping petals. The shape and orientation of the surfaces representing petals vary from one stage to another. This ray-traced image contains approximately 400 plants, each with 15 flowers. A flower has 21 petals, each modeled using 600 triangles, and 300 seeds, each modeled using 400 triangles. Counting leaves and buds, the entire scene contains about 800,000,000 triangles.

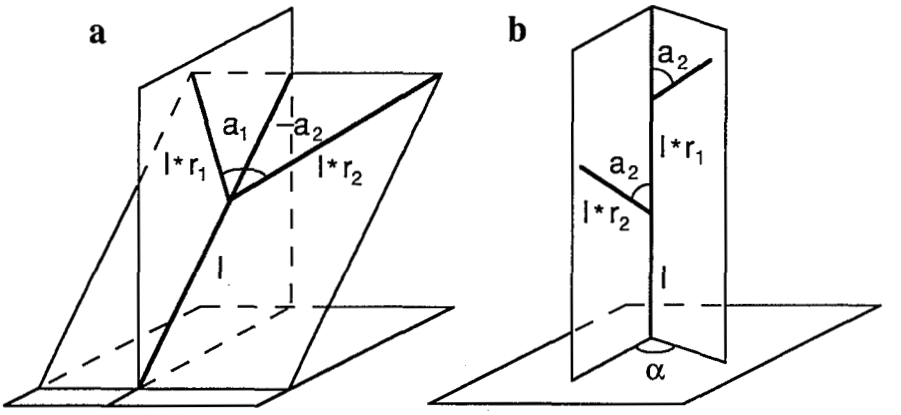


Figure 5.7: Specification of tree geometry according to Honda [74]

5.3 Trees

L-systems have not been widely used for graphical modelling of trees. One reason is that the structure of trees, particularly in moderate climates, is strongly dependent on environmental factors, which are not directly captured by L-systems. In addition, even approximate models of tree architecture require precisely chosen branch growth rates and branching angles, which are difficult to express using standard L-systems. With the addition of parameters, however, L-systems become a convenient tool for specifying non-developmental architectural models of trees similar to those described in Section 1.2.2 on page 6. An example is given below.

One of the earliest presentations of computer simulation for the modelling of trees was by Honda [74], who studied the contribution of tree branching angle and relative branch length to the overall form of trees. His model made the following assumptions (Figure 5.7).

- Tree segments are straight and their girth is not considered.
- A mother segment produces two daughter segments through one branching process.

- The lengths of the two daughter segments are shortened by constant ratios, r_1 and r_2, with respect to the mother segment.
- The mother segment and its two daughter segments are contained in the same *branch plane*. The daughter segments form constant *branching angles*, a_1 and a_2, with respect to the mother branch.
- The branch plane is oriented so as to be closest to a horizontal plane. More formally, the line perpendicular to the mother segment and lying in the branch plane is horizontal. An exception is made for branches attached to the main trunk. In this case, a constant *divergence angle* α between consecutively issued lateral segments is maintained (cf. Section 5.2.2).

Honda's models can be captured using parametric L-systems with turtle interpretation. For example, the following L-system implements those tree models of Honda [74] in which one of the branching angles is equal to 0, yielding structures with clearly delineated main and lateral axes. The model has been extended to consider relative branch widths.

L-system 6: Honda's model for trees

```
#define r1      0.9         /* Contraction ratio for the trunk */
#define r2      0.6         /* Contraction ratio for branches */
#define a0      45          /* Branching angle from the trunk */
#define a2      45          /* Branching angle for lateral axes */
#define d       137.5       /* Divergence angle */
#define wr      0.707       /* Width contraction ratio */
ω : A(1,10)
p1 : A(s,w)  →  !(w)F(s)[&(a0)B(s*r2,w*wr)]/(d)A(s*r1,w*wr)
p2 : B(s,w)  →  !(w)F(s)[−(a2)@VC(s*r2,w*wr)]C(s*r1,w*wr)
p3 : C(s,w)  →  !(w)F(s)[+(a2)@VB(s*r2,w*wr)]B(s*r1,w*wr)
```

According to production p_1, the apex of the main axis A produces an internode F and a lateral apex B in each derivation step, and modifies its own parameters in preparation for the next step. Constants r1 and r2 specify contraction ratios for

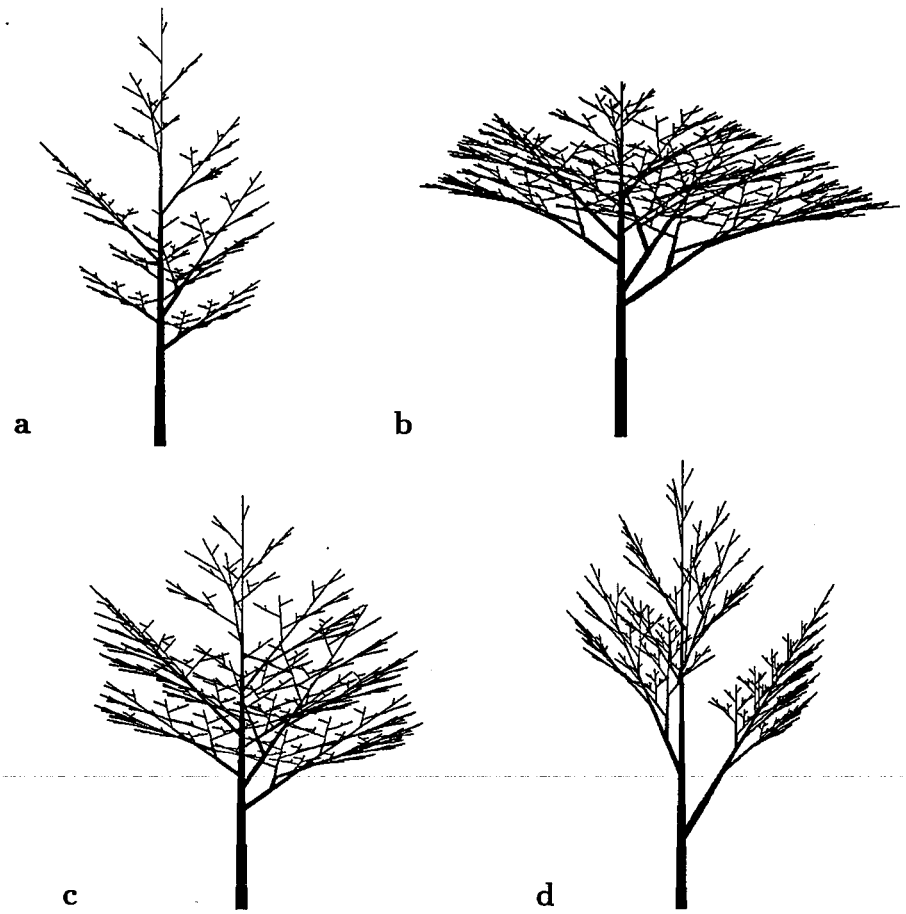


Figure 5.8: Examples of the tree-like structures by Honda [74], generated using L-systems

the straight and lateral segments, a0 and a2 are the branching angles and d is the divergence angle. The length of the internode F is determined by the value of the first parameter of A, and decreases by the values r1 and r2 between mother and daughter internodes. Module !(w) sets the line width to w, the value of the second parameter of A. The width contraction ratio, wr = 0.707, is consistent with a postulate by Leonardo da Vinci (as reported by Mandelbrot [98, page 156]), stating that "all the branches of a tree at every stage of its height when put together are equal in thickness to the trunk below them." In the case where a mother branch of diameter w_1 gives rise to two daughter branches of equal diameter w_2, this postulate yields the equation $w_1^2 = 2w_2^2$, with a solution $w_2/w_1 = 1/\sqrt{2} \approx 0.707$. A general discussion of the relationships between the diameters of the mother and daughter branches is included

in a book by Macdonald [97, pages 131–135].

Productions p_2 and p_3 describe subsequent development of the lateral branches. In each derivation step, the straight apex (either B or C) issues a next-order lateral apex at angle a2 or $-$a2 with respect to the mother axis. Two productions are employed to create lateral apices alternately to the left and right. The modules @V roll the turtle around its own axis so that the turtle's left vector is brought to a horizontal position. Consequently, the branch plane is "closest to a horizontal plane", as required by Honda's model. From a technical point of view, @V is a black-box procedure that modifies the turtle's orientation in space according to the formulae

$$\vec{L} = \frac{\vec{V} \times \vec{H}}{|\vec{V} \times \vec{H}|} \quad \text{and} \quad \vec{U} = \vec{H} \times \vec{L},$$

where vectors $\vec{H}$, $\vec{L}$, and $\vec{U}$ are the heading, left, and up vectors associated with the turtle, $\vec{V}$ is the direction opposite to gravity, and $|\vec{A}|$ denotes the length of vector $\vec{A}$. The tree-like structures shown in Figure 5.8 were generated using the values of constants listed in the following table and coincide with the structures presented by Honda.

Figure	r1	r2	a0	a2
a	0.9	0.6	45	45
b	0.9	0.9	45	45
c	0.9	0.8	45	45
d	0.9	0.7	30	-30

A related L-system incorporating productions to specify leaves was used to generate the trees and bushes in Figure 5.9 and Plate C.7. While no detailed models of particular species have been developed, these examples demonstrate that trees are within the range of models expressible using parametric L-systems.

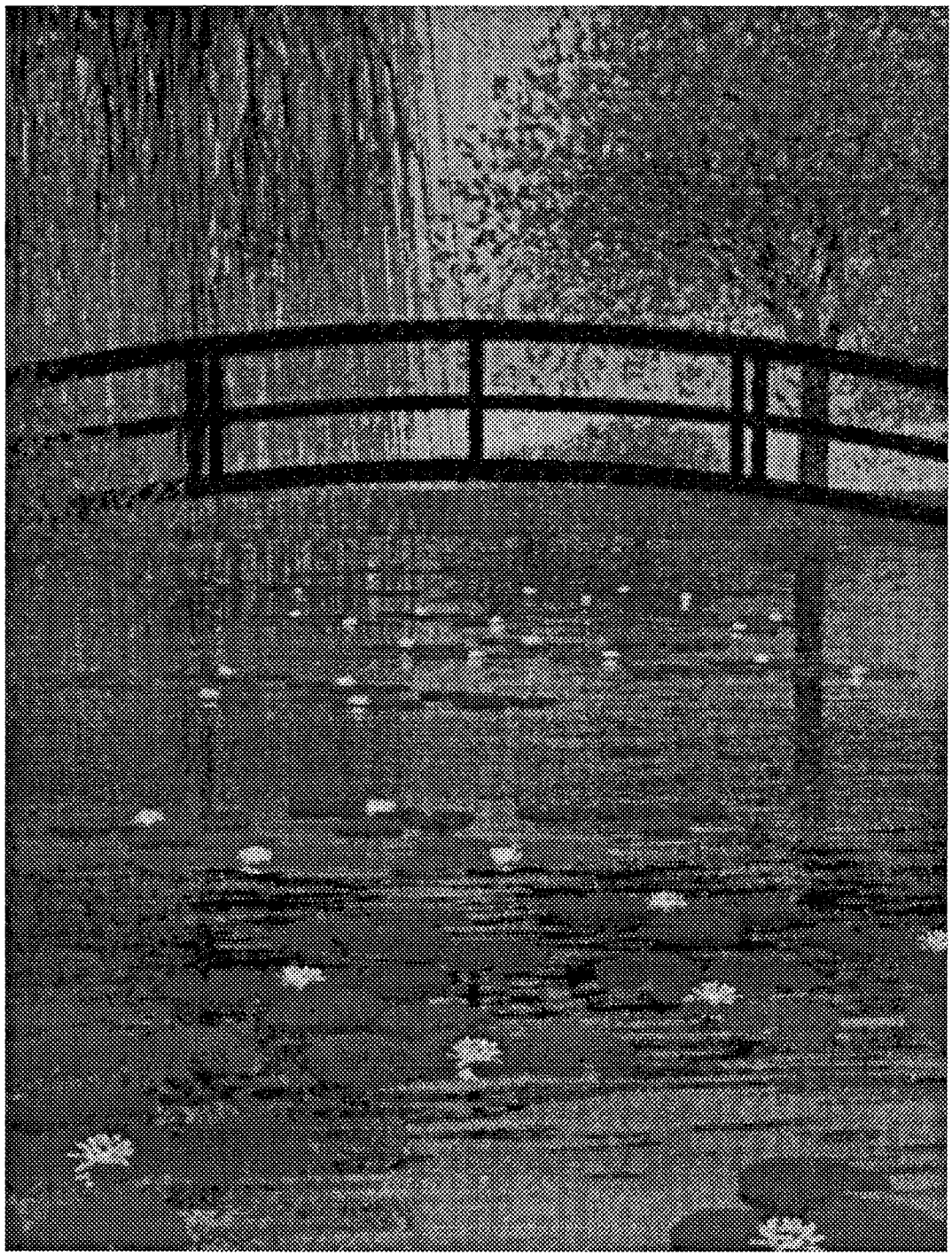


Figure 5.9: Water-lilies ©1990 D. R. Fowler, J. Hanan, P. Prusinkiewicz, and N. Fuller

5.4 Compound leaves

Herman, Lindenmayer, and Rozenberg [68] pointed out that "in the case of a compound leaf [...] some of the lobes (or leaflets), which are parts of a leaf at an advanced stage, have the same shape as the whole leaf at an earlier stage." This observation emphasizes the self-similar nature of compound leaves. Development of these structures is heavily dependent on delays occurring between the initiation and expansion of lateral apices. In a non-parametric L-system, delays are specified using a sequence of states, as in the following model of compound leaves.

L-system 7: `A non-parametric model of compound leaves`

ω : A

p_1 : A $\rightarrow$ F[+D][−D]FA

p_2 : D $\rightarrow$ C

p_3 : C $\rightarrow$ B

p_4 : B $\rightarrow$ A

p_5 : F $\rightarrow$ FF

In this model, the main apex produces an internode and two lateral apices D in each step, according to production p_1. The development of the letter representing a lateral apex is delayed by 3 derivation steps, as the letter progresses through the sequence D $\mapsto$ C $\mapsto$ B $\mapsto$ A. In the meantime, production p_5 models the doubling of internode length in each step. Any increase in the delay would require the addition of a new production for every extra step of delay.

Wood [155, 156] considered delay mechanisms important enough that he introduced a special formalism called *time-delayed L-systems* to implement them. For the same purpose, Hanan [61, Section 2.5] presented a technique for automatic incrementing and decrementing of numeric subscripts within standard L-systems. Using parametric L-systems, the delay mechanism and growth rates can be controlled by parameter values, as in the following parametric model of compound leaf development. Rather than having to add new state symbols and associated productions in order to create a longer delay, the user simply has to increase the value of the delay constant.

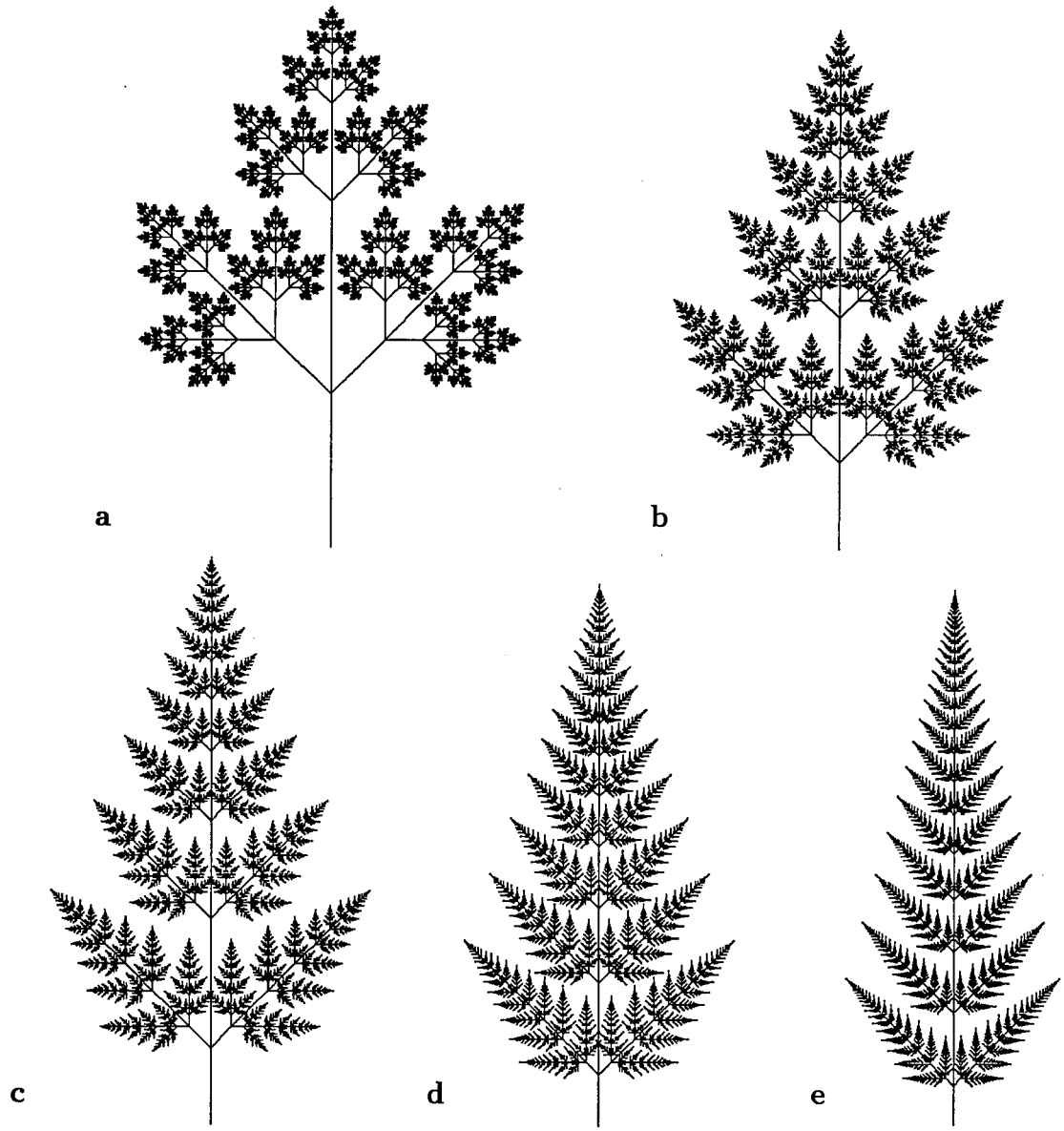


Figure 5.10: Examples of compound leaves

L-system 8: A model of compound leaves

```
#define Delay      3    /* Delay for lateral apices */
#define Rate       1.5  /* Internode elongation rate */
ω : A(0)
p1 : A(d) : d > 0   →  A(d − 1)
p2 : A(d) : d == 0  →  F(1)[+A(Delay)][−A(Delay)]F(1)A(0)
p3 : F(a)           →  F(a ∗ Rate)
```

According to production p_2, in each derivation step the apex A(d) produces two segments F(1) and a pair of daughter branches. Production p_1 delays the development of the daughter branches by Delay steps with respect to the mother branch. This branching pattern is repeated in the daughter branches. Production p_3 gradually elongates the internodes, and in this way establishes proportions between leaf parts. Sample leaves generated by the above L-system are shown in Figure 5.10. The corresponding parameter values are collected in the following table.

Figure	Delay	Rate	Derivation length
a	0	2.00	10
b	1	1.50	16
c	2	1.36	21
d	4	1.23	30
e	7	1.17	45

The model is very sensitive to the growth rate value — a change of 0.01 visibly alters the spacing between leaf components.

5.5 Simple leaves

Although bracketed L-systems serve primarily as a method for generating branching structures, they can also be applied to model surfaces such as leaf blades. One technique, previously described in Section 3.3.2, uses polygons to build leaf surfaces. The boundary of a polygon is composed of segments of an underlying branching structure and extra edges which connect terminal nodes. The leaf shapes depend strongly

on relative growth rates and may change as the segments elongate over time. By encoding the growth rates using parameters, the user can control them easily and accurately. An L-system modelling a class of simple leaves using this approach is given below.

L-system 9: A model of simple leaves

```
#define LA   5          /* initial length - main segment */
#define RA   1          /* growth rate - main segment */
#define LB   1          /* initial length - lateral segment */
#define RB   1          /* growth rate - lateral segment */
#define PD   0          /* growth potential decrement */
```

ω : {.A(0)}

p_1 : A(t) $\rightarrow$ G(LA, RA)[−B(t).][A(t + 1)][+B(t).]

p_2 : B(t) : t > 0 $\rightarrow$ G(LB, RB)B(t − PD)

p_3 : G(s, r) $\rightarrow$ G(s ∗ r, r)

According to production p_1, in each derivation step the apex A(t) extends the main leaf axis by segment F(LA, RA), and creates a pair of lateral apices B(t). New lateral segments are added by production p_2. The parameter t, assigned to the apices B(t) by production p_1, is decremented by the value of the constant PD. The production of new lateral segments stops when t reaches 0. Intuitively, the initial value of the parameter t plays the role of *growth potential* of the branches; the larger the value of t, the longer the branch will grow. Segment elongation is captured by production p_3.

It is convenient to consider a leaf modelled by L-system 9 in two parts: a basal area and an apical area. In the basal area, the number of lateral segments is determined by the initial value of growth potential t and the parameter PD. Since the growth potential increases towards the leaf apex, the consecutive branches contain more and more segments. In contrast, branches in the apical area exist for too short a time to reach their full length. The actual number of segments in this area is proportional to the time a branch has to grow, and gradually decreases towards the apex. As a result of these opposite tendencies, a leaf reaches its maximum width near the middle of the blade. For a given derivation length, the exact position of the branch with the largest number of segments is determined by the value of the parameter PD. The

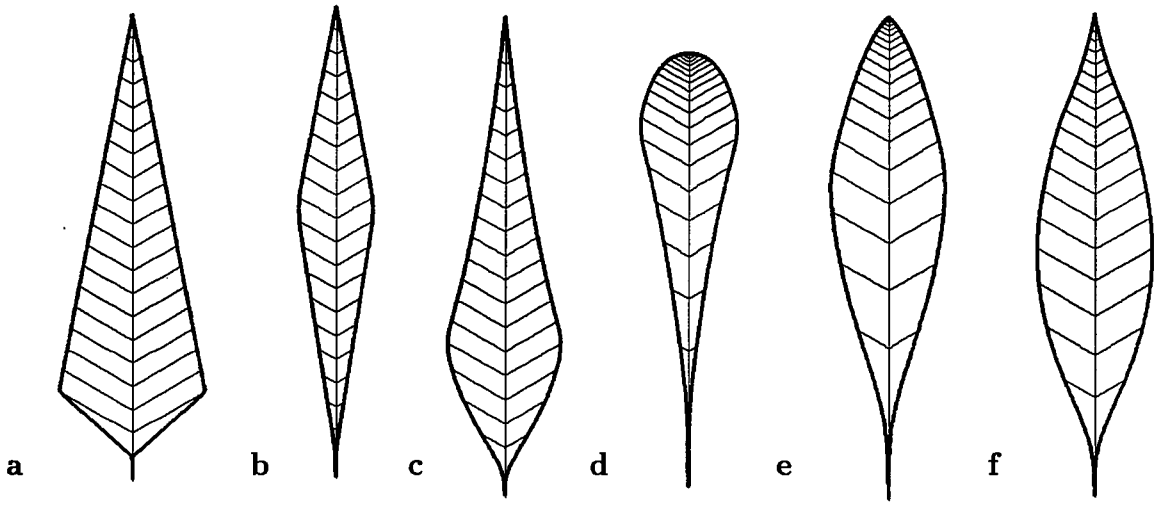


Figure 5.11: Examples of simple leaves

following table collects parameter values corresponding to the leaf models shown in Figure 5.11.

Figure	LA	RA	LB	RB	PD
a	5	1.0	1.0	1.00	0.00
b	5	1.0	1.0	1.00	1.00
c	5	1.0	0.6	1.06	.25
d	5	1.2	10.0	1.00	0.50
e	5	1.2	4.0	1.10	0.25
f	5	1.1	1.0	1.20	1.00

If the parameter PD is equal to 0, all lateral branches have an unlimited growth potential, and the basal part of the leaf does not exist (a). If PD is equal to 1, the basal and apical parts contain equal numbers of lateral branches (b and f). Finer details of the leaf shape result from the growth rates. If the main axis segments and the lateral segments have the same growth rates (RA = RB), the edges of the apical part of the leaf are straight (a and b). If RA < RB, the segments along the main axis elongate at a slower rate than the lateral segments, resulting in a negative edge curvature of the apical part of the leaf (c). In the opposite case, with RA > RB, these edges have positive curvature (d, e and f). The curvature of the basal edges can be analyzed in a similar way.

5.6 Developmental bicubic surfaces

As described in Section 3.3.2, L-systems can be used to model the development of plant organs, such as leaves and petals, using polygons which are modified over time. However, bicubic surfaces provide a more convenient method for modelling smooth curved surfaces; a very complex L-system would be required to produce a polygonal surface as smooth as a bicubic patch. Developmental bicubic surfaces can be incorporated into a model using the following set of black-box routines, which allow the specification of a Bezier-form bicubic surface [10, 42, 61].

- @PS(i) initializes the four rows and columns of control points for surface i to $(0, 0, 0)$.
- @PC(i, r, c) assigns the current position of the turtle to the control point of surface i in row r and column c.
- @PD(i, s, t) draws the surface defined by the control points of surface i using s lines along the rows and t along the columns.

The first step in creating a developmental model of a plant organ is to define the initial and final shapes in the sequence. When using an interactive surface editor, the user works with 16 control points for each surface patch. The manipulation of a three-dimensional control point using a two-dimensional input device, such as a mouse, is not necessarily straightforward. In addition, the creation of the symmetric shapes common in plant components often requires the concerted readjustment of several control points, which can be a tedious task using a standard interactive editor. Parametric L-systems can be used to implement a more intuitive set of parameters defining a particular class of surface shapes. The following L-system allows the user to manipulate parameters for petal width, length, and bending angles in order to model members of a family of petals. It is a simple hierarchical model of one possible

control point layout.

L-system 10: Bicubic surface petals

```
#define CL  100          /* Central length */
#define BW  35           /* Base width */
#define TW  35           /* Tip width */
#define BA  0            /* Base angle */
#define TA  0            /* Tip angle */
```

ω : P

p_1 : P $\rightarrow$ [S[l][r]B[L][R]D]

p_2 : S $\rightarrow$ @PS(0)f(30)

p_3 : B $\rightarrow$ $\wedge$(BA)f(CL) $\wedge$ (TA)

p_4 : D $\rightarrow$;(100)@PD(0,4,4)

p_5 : l $\rightarrow$ +(90)f(BW)@PC(0,0,0) + (90 + atan(CL/BW))
[|f(CL/3)@PC(0,1,0) − (90) $\wedge$ (BA)f(BW * 2/3)@PC(0,1,1)]
[f(50)@PC(0,0,1)]

p_6 : r $\rightarrow$ −(90)f(BW)@PC(0,0,3) − (90 + atan(CL/BW))
[|f(CL/3)@PC(0,1,3) + (90) $\wedge$ (BA)f(BW * 2/3)@PC(0,1,2)]
[f(50)@PC(0,0,2)]

p_7 : L $\rightarrow$ +(90)f(TW)@PC(0,3,0) + (90 − atan(50/TW))
[f(CL/3))@PC(0,2,0) + (90) $\wedge$ (TA)f(TW * 2/3)@PC(0,2,1)]
[|f(30)@PC(0,3,1)]

p_8 : R $\rightarrow$ −(90)f(TW)@PC(0,3,3) − (90 − atan(50/TW))
[f(CL/3)@PC(0,2,3) − (90) $\wedge$ (TA)f(TW * 2/3)@PC(0,2,2)]
[|f(30)@PC(0,3,2)]

According to production p_1 a petal is composed of the start segment S, left and right halves of the leaf base l and r, the body B, left and right halves of the leaf tip L and R, and the drawing segment D. Production p_2 issues the patch initialization command @PS(0). The f(30) module moves the turtle so that the edge of the surface will go through the turtle's initial position. The petal is modelled as two laterally symmetric halves, each consisting of a base and tip portion. Productions p_5 and p_6 define the leaf base by producing mirror-image responses in the turtle with respect

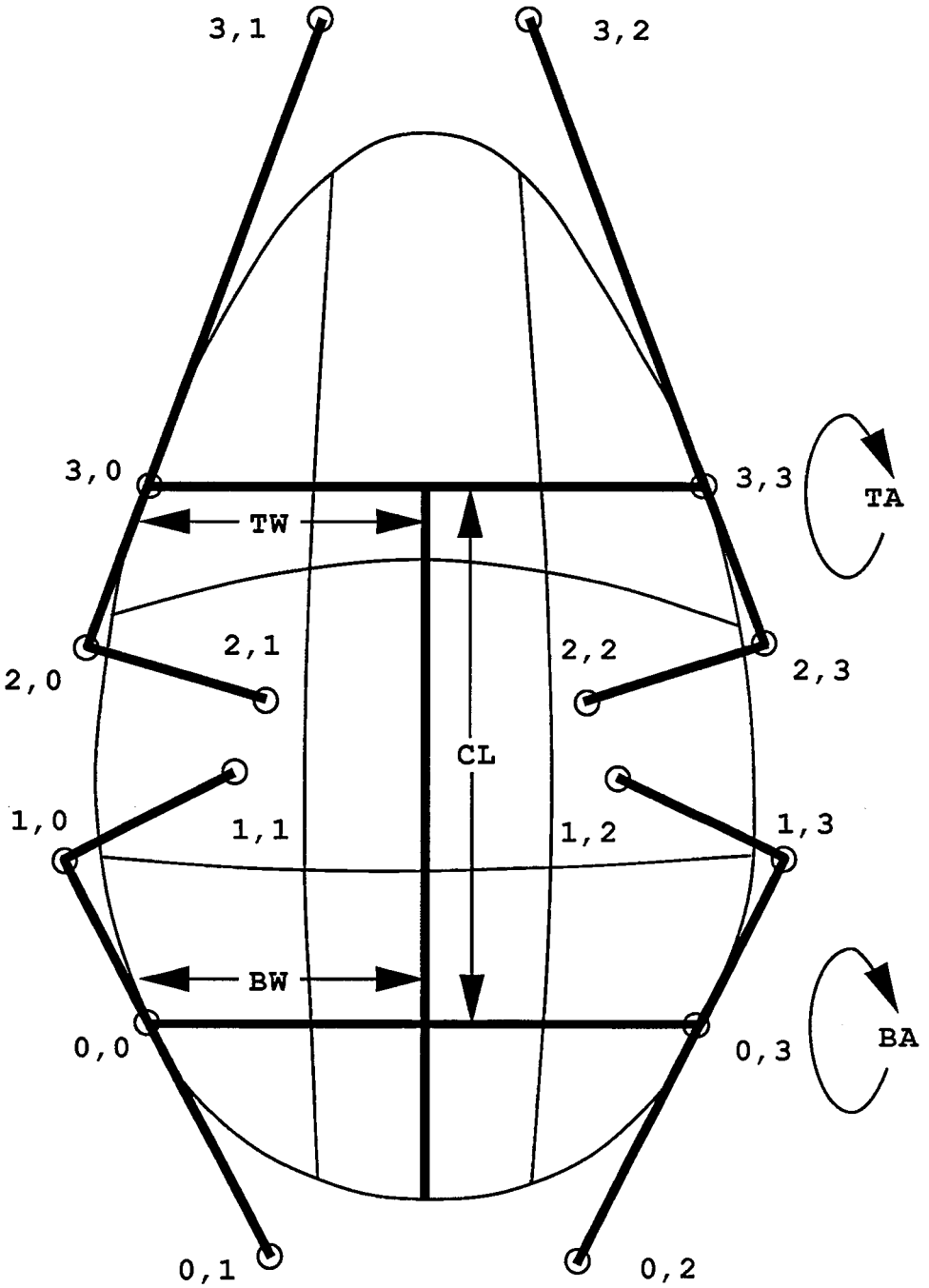


Figure 5.12: Petal control structure. Control points are labelled by row and column.

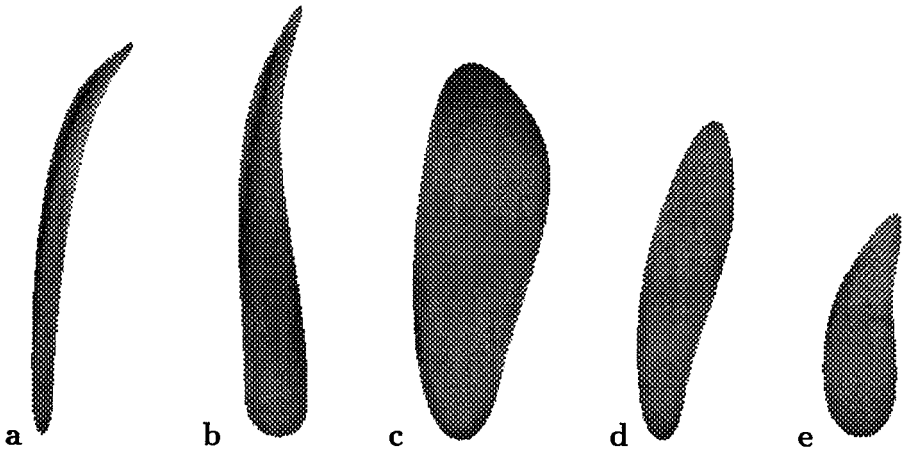


Figure 5.13: Petal shapes

to the central axis. Productions p_7 and p_8 do the same for the leaf tip. Production p_3 defines the central length and relative angles of the base and tip. Production p_4 specifies the colour command ;(100) and the patch drawing command @PD(0,4,4). As illustrated in Figure 5.12, the base of the leaf is defined by the first two rows of control points in the bicubic patch, while the tip is defined by the last two rows. This L-system allows the user to control a petal's shape in terms of its central length CL, its tip and base width, TW and BW, and the angles between base and center line, BA, and between center line and tip, TA. The remainder of the angles and lengths are defined by the family of surfaces to be modelled and the geometry of a Bezier patch. For instance, in order to maintain first order continuity of the edge passing through a control point at a corner of the patch, the control point and its neighbours in the outside row and column must be collinear.

Interactive manipulation of the parameters in the #define statements produced the petal shapes in Figure 5.13, which correspond to the values in the following table.

Figure	CL	BW	TW	BA	TA
a	150	5	5	25	50
b	150	15	5	0	50
c	120	20	25	12	-40
d	100	10	15	25	0
e	50	15	10	12	40

Once the initial and final shapes have been chosen, an L-system must be designed to interpolate between the two shapes. For example, the following L-system interpolates between shapes e and c in Figure 5.13.

L-system 11: Developmental bicubic surface petal

```
#define N    10   /* Number of steps */
#define ICL  50   /* Initial central length */
#define FCL  150  /* Final central length */
#define IBW  15   /* Initial base width */
#define FBW  15   /* Final base width */
#define ITW  10   /* Initial tip width */
#define FTW  5    /* Final tip width */
#define IBA  15   /* Initial base angle */
#define FBA  0    /* Final base angle */
#define ITA  35   /* Initial tip angle */
#define FTA  50   /* Final tip angle */
```

ω : P

p_1 : P $\rightarrow$ [S[l][r]B[L][R]D]

p_2 : S $\rightarrow$ @PS(0)f(30)

p_3 : B $\rightarrow$ $\wedge$(IBA, FBA, (FBA − IBA)/N)f(ICL, FCL, (FCL − ICL)/N)
$\wedge$(ITA, FTA, (FTA − ITA)/N)

p_4 : D $\rightarrow$;(100)@PD(0, 4, 4)

p_5 : l $\rightarrow$ +(90)f(IBW, FBW, (FBW − IBW)/N)@PC(0, 0, 0) + (90 + atan(ICL/IBW),
90 + atan(FCL/FBW), (atan(FCL/FBW) − atan(ICL/IBW))/N)
[|f(ICL/3, FCL/3, (FCL − ICL)/3/N)@PC(0, 1, 0) − (90)
$\wedge$(IBA, FBA, (FBA − IBA)/N)f(IBW ∗ 2/3, FBW ∗ 2/3, 2/3 ∗ (FBW − IBW)/N)
@PC(0, 1, 1)][f(50)@PC(0, 0, 1)]

p_6 : r $\rightarrow$ −(90)f(IBW, FBW, (FBW − IBW)/N)@PC(0, 0, 3) − (90 + atan(ICL/IBW),
90 + atan(FCL/FBW), (atan(FCL/FBW) − atan(ICL/IBW))/N)
[|f(ICL/3, FCL/3, (FCL − ICL)/3/N)@PC(0, 1, 3) + (90)
$\wedge$(IBA, FBA, (FBA − IBA)/N)f(IBW ∗ 2/3, FBW ∗ 2/3, 2/3 ∗ (FBW − IBW)/N)
@PC(0, 1, 2)][f(50)@PC(0, 0, 2)]

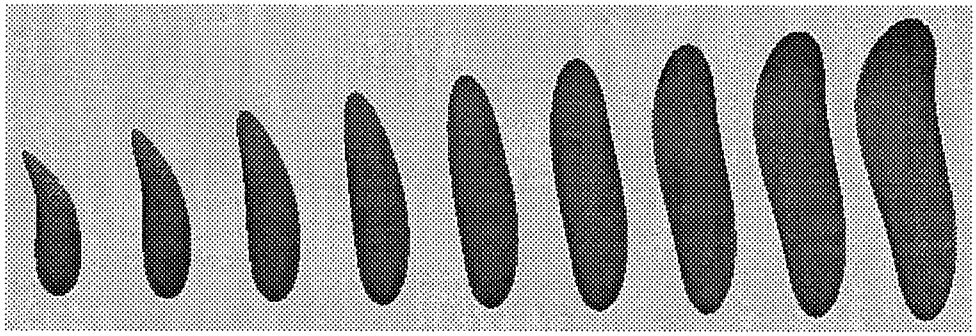


Figure 5.14: Development of a petal

L-system 11: Developmental bicubic surface petal - continued

p_7 : L $\rightarrow$ +(90)f(TW)@PC(0,3,0)+(90 − atan(50/TW))
[f(ICL/3),FCL/3,(FCL − ICL)/3/N)@PC(0,2,0)+(90)
∧(ITA,FTA,(FTA − ITA)/N)f(ITW ∗ 2/3,FTW ∗ 2/3,
2/3 ∗ (FTW − ITW)/N)@PC(0,2,1)][|f(30)@PC(0,3,1)]

p_8 : R $\rightarrow$ −(90)f(TW)@PC(0,3,3) − (90 − atan(50/TW))
[f(ICL/3,FCL/3,(FCL − ICL)/3/N)@PC(0,2,3) − (90)
∧(ITA,FTA,(FTA − ITA)/N)f(ITW ∗ 2/3,FTW ∗ 2/3,
2/3 ∗ (FTW − ITW)/N)@PC(0,2,2)][|f(30)@PC(0,3,2)]

p_9 : f(v,V,i) : v < V $\rightarrow$ f(v + i,V,i)

p_{10} : +(v,V,i) : v < V $\rightarrow$ +(v + i,V,i)

p_{11} : −(v,V,i) : v < V $\rightarrow$ −(v + i,V,i)

p_{12} : ∧(v,V,i) : v < V $\rightarrow$ ∧(v + i,V,i)

The turtle interpretation commands with values to be interpolated have three parameters: v representing the current value, V representing the limit or final value, and i representing the increment to be applied in each step. Productions p_1 to p_8 are the same as before, except that modules representing commands with parameters to be interpolated have the appropriate initial values included. Productions p_9 to p_{12} control the linear interpolation of lengths and angles. This L-system produces the sequence of images presented in Figure 5.14. The sequence of flower heads shown

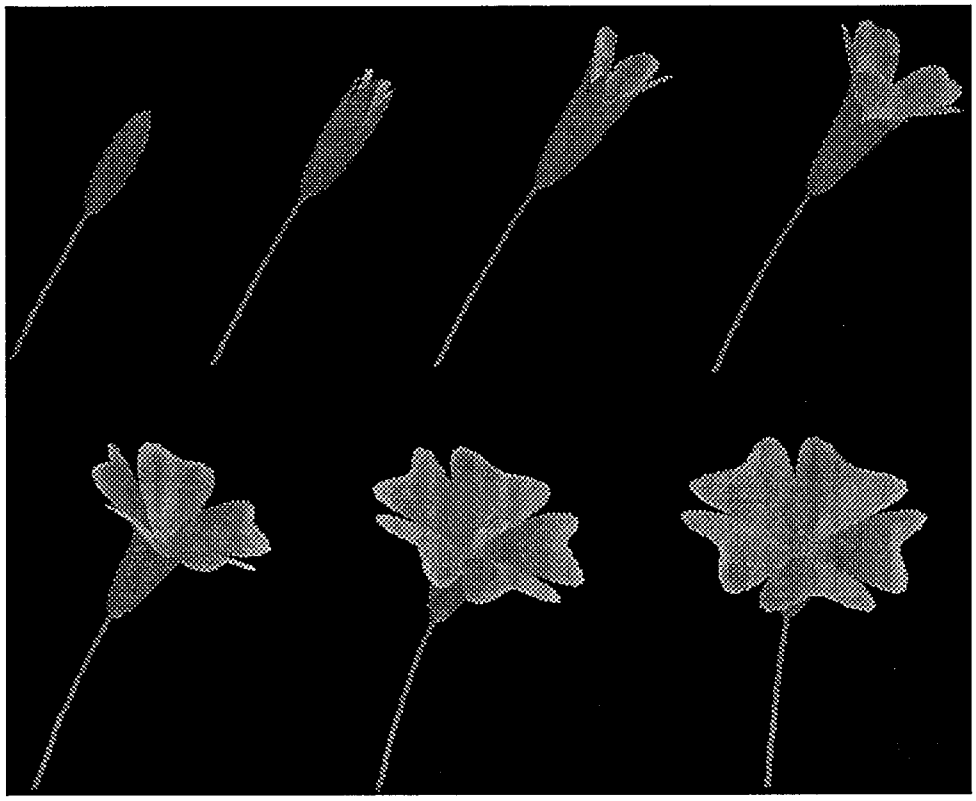


Figure 5.15: Development of a rose campion flower ©1991 P. Prusinkiewicz and M. Hammel

in Figure 5.15 and Plate C.4 come from an animation of rose campion development produced by Prusinkiewicz and Hammel [118] using a similar technique.

The presence of parameters allows the specification of control points by row and column number in the black-box routines. A less intuitive symbolic identification of the black-box routines would have been required for standard L-systems.

5.7 Animation of development

A developmental plant model expressed using an L-system presents consecutive stages of the plant's growth as time progresses in discrete steps. The resulting images can be displayed in a sequence, simulating time-lapse photography of the modelled organism. In order to create a smooth animation of development, the modelling system must provide two capabilities. First, the user must be able to express arbitrary

sizes in the model, which was one of the main motivations for the creation of parametric L-systems. Second, the user must be able to control the extent of changes in the model from one derivation step to the next. By making these changes sufficiently small, the illusion of a continuous process is created.

A conceptually elegant approach to the latter problem is to decouple model development, defined in continuous time, from model observation, taking place at discrete time intervals. For context-free models this has been achieved in the formalism of *timed D0L-systems* developed by Prusinkiewicz [122, Chapter 6]. Following a similar line of reasoning, this section presents a method for creating animations using the formalism of parametric L-systems.

A user-defined parameter dt, called the *time step*, is introduced to control the progress of time. An age parameter is associated with each module in the string, and is advanced by dt in each derivation step. Production application is determined by comparison of the age, or some function of age, to a threshold value. Geometric parameters are expressed using growth functions associated with the age of the module. These growth functions must be selected so that no sudden changes occur when modules divide, using continuity criteria to be discussed later. Note that this usage of the term growth function is different from the non-parametric case, where it referred to the number of characters in the string.

As an example, consider the parametric model for the vegetative growth of *Anabaena catenula* presented in L-system 2 on page 61. The growth of a module is tightly coupled with the derivation step; in a single step the module grows from length BS to AS, in the next it divides into two modules with total length AS + BS. In order to control development using a finer time step dt, age is introduced as a parameter of the modules. When a new cell is created it has an initial age α, incremented by dt in each derivation step. The cell exists until it reaches a terminal age β, at which time it divides. Note that a given increment dt may consist of two parts at the time of cell division: $\mathtt{dt}_1$ needed to reach the terminal age, and $\mathtt{dt}_2$ which advances the age of the newly created cells. In the following L-system, the terminal age is represented by the age threshold T and the growth function is not yet specified.

L-system 12: An age-controlled model of *Anabaena catenula*

```
#define dt      1          /* Time step */
#define W       .25        /* Width of cells */
#define T       1.2        /* Age threshold for division */
#define AA      1.0        /* Age decrement for cell type A */
#define BA      1.2        /* Age decrement for cell type B */
#define R       1          /* Orientation to the right */
#define L       −1         /* Orientation to the left */
#define g(a)    ????       /* Unspecified growth function */
```

ω : !(W)F(g(T − BA), R, T − BA)

p_1 : F(l, o, a) : a + dt >= T && o == R →
F(g(a + dt − AA), L, a + dt − AA)F(g(a + dt − BA), R, a + dt − BA)

p_2 : F(l, o, a) : a + dt >= T && o == L →
F(g(a + dt − BA), L, a + dt − BA)F(g(a + dt − AA), R, a + dt − AA)

p_3 : F(l, o, a) : a + dt < T → F(g(a + dt), o, a + dt)

A single cell is represented by a module F(l, o, a), where the parameter l captures the cell's length, o captures its orientation, and a captures its age. The axiom sets the width of the filament using the !(W) module, and incorporates a single cell of type B oriented to the right, represented by the module F(g(T − BA), R, T − BA). In order to ensure consistency in the model, the starting age is specified by subtracting the age decrement BA from the cell division threshold T, and the cell's length is specified using the value of the growth function g(a) at that age. Production p_1 simulates the division of a cell oriented to the right when its age reaches or exceeds the threshold T, producing an older cell represented by F(g(a + dt − AA), L, a + dt − AA) and a younger cell represented by F(g(a + dt − BA), R, a + dt − BA). The mechanism for specifying the new ages of the modules based on a decrement from the age plus the time step dt ensures that the dt_2 portion of the time step is incorporated into the age of the new modules. Application of this production to the module F(2, 1, 1.4) with dt = 0.7 would result in the string F(2, −1, 1.1)F(1, 1, .1), where the "excess" age has been distributed into the new modules. Production p_2 performs an analogous

function for a cell oriented to the left. In each derivation step that p_1 or p_2 does not apply, production p_3 increments the age of a module and updates its length.

Note that this L-system only works properly for dt less than or equal to the smallest amount of time between any two state transitions, which in this case is equal to $\mathtt{AA} = 1$. For instance, application of production p_1 to the module $\mathtt{F}(2, 1, 1.6)$ with $\mathtt{dt} = 1.6$ would result in the string $\mathtt{F}(2, -1, 2.2)\mathtt{F}(1, 1, 1.2)$, in which the first module should have been replaced by production p_2 already. If the L-system is being used for animation purposes, keeping dt below this threshold is not a problem; a small dt ensures small changes from frame to frame. In the timed L-system formalism, the notion of derivation is modified to allow the use of an arbitrarily large dt without increasing the number of productions; however, this approach is fundamentally limited to context-free models. In parametric L-systems on the other hand, larger time steps can be accommodated by providing productions that carry out the equivalent of multiple derivations in a single step. In conjunction with p_1, the following production would handle any value of dt up to 2.

$$
\begin{aligned}
p_0: \quad \mathtt{F}(\mathtt{l}, \mathtt{o}, \mathtt{a}) \quad : \quad & \mathtt{a} + \mathtt{dt} >= \mathtt{T} + \mathtt{AA} \;\&\&\; \mathtt{o} == \mathtt{R} \quad \rightarrow \\
& \mathtt{F}(\mathtt{g}(\mathtt{a} + \mathtt{dt} - \mathtt{AA} - \mathtt{BA}), \mathtt{L}, \mathtt{a} + \mathtt{dt} - \mathtt{AA} - \mathtt{BA}) \\
& \mathtt{F}(\mathtt{g}(\mathtt{a} + \mathtt{dt} - \mathtt{AA} - \mathtt{AA}), \mathtt{R}, \mathtt{a} + \mathtt{dt} - \mathtt{AA} - \mathtt{AA}) \\
& \mathtt{F}(\mathtt{g}(\mathtt{a} + \mathtt{dt} - \mathtt{AA} + \mathtt{AA}), \mathtt{R}, \mathtt{a} + \mathtt{dt} - \mathtt{AA} + \mathtt{AA})
\end{aligned}
$$

This is equivalent to applying p_2 to the first module in the successor of p_1, and p_3 to the second. Similar productions would have to be added for p_2 and p_3. If larger time steps were required, more productions would have to be added.

In order to produce a smooth animation with this L-system, the growth function g(a), which determines the length of the modules based on their age, must be specified. Potentially, observations of real organisms can be used to estimate growth functions [79, 80]. If detailed information is unavailable, growth functions can be determined based on the desired degree of smoothness in the animation. According to Mitchison and Wilcox [107], both the cells resulting from a division in *Anabaena* reach the same size before dividing again, with the smaller cell taking an average of twenty percent more time to do so. Maintaining the assumption that the smaller cell is just a younger version of the larger, their observation can be incorporated into the

model by assuming a terminal age of $\beta = 1.2$, and having the smaller cell's age start at $\alpha = 0$ while the larger starts at $\alpha = 0.2$. In determining the growth function for this model, the basic continuity requirements are defined by general physical properties of biological systems: the length of each cell is a continuous function of time, and the length of a cell before subdivision is equal to the sum of the lengths of the daughter cells. In the context of an age-controlled parametric L-system model of a non-branching organism, these requirements can be considered as follows:

R1. The *growth function* $g_l(a, \tau)$, which specifies the value of parameter l for module a as a function of the age parameter τ, must be a continuous function of τ in the domain $[\alpha_{min}, \beta]$, where age α_{min} is the minimum of the initial age values assigned to a by the axiom ω or by some production in P.

R2. For each production in P, where module a produces modules $b_1 b_2 \ldots b_n$ at time threshold β specified in the condition, the following equality holds:

$$g_l(a, \beta) = \sum_{i=1}^{n} g_l(b_i, t(b_i, \beta)) \tag{5.2}$$

where $t(b_i, \beta)$ represents the age assigned to module b_i if the production is applied at time β.

In practice, requirement R1 is used to select the class of growth functions under consideration, and the equations resulting from requirement R2 are used to determine the parameters in growth function definitions. For example, in the case of the age-controlled L-system specified in L-system 12 on page 86, requirement R1 can be satisfied using a linear growth function

$$g_l(F, \tau) = A\tau + B. \tag{5.3}$$

Requirement R2 can be reduced to the form

$$g_l(F, 1.2) \;=\; g_l(F, 0.2) + g_l(F, 0) \tag{5.4}$$

By substituting equation (5.3) into (5.4), we obtain

$$1.2A + B \;=\; (0.2A + B) + (0A + B) \tag{5.5}$$

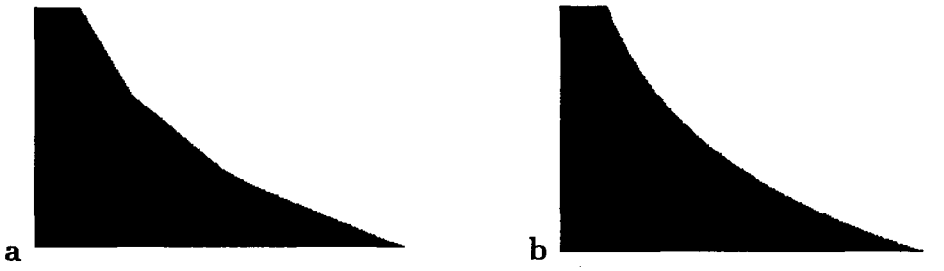


Figure 5.16: Diagrammatic representation of the development of *Anabaena catenula*, with (a) linear and (b) exponential growth of cells

Therefore, the desired continuity of development is provided by all linear growth functions where $A = B$. This can be specified by replacing the definition of g(a) in L-system 12 as follows, where $A = B = 30$.

```
#define g(a)  30*(a)+30          /* Linear growth function */
```

In this case the growth function is simple enough that its recalculation whenever a new age is assigned to a module will not affect the time taken to derive a new string to a great degree. If very complex growth functions are required, forward differencing can be applied to make the calculations more efficient.

Figure 5.16a illustrates the developmental process for $A = B = 30$. The diagram was obtained by plotting the cells in the filament as horizontal line segments on the screen. The image was created in 325 derivation steps with time step `dt` $= .01$.

The slopes of the right hand edges of the diagram represent growth rates of the entire structure. Notice that they remain constant in the periods between cell divisions, then change. This effect is disconcerting in animation, since the rate of organism growth suddenly increases with each cell division. In order to prevent this, it is necessary to extend requirements R1 and R2 to a higher order of continuity N. Specifically, equation (5.2) takes the form

$$g^{(k)}(a,\beta) = \sum_{i=1}^{n} g^{(k)}(b_i,\alpha_i) \text{ for } k = 0,1,\ldots,N, \tag{5.6}$$

where $g^{(k)}(a,\tau)$ is the k^{th} derivative of the growth function $g(a,\tau)$ with respect to age τ.

In the case of *Anabaena*, an attempt to achieve first-order continuity assuming linear growth functions yields an uninteresting solution, $g(F, \tau) = 0$. Thus, more complex growth functions must be considered. Sigmoidal growth, represented by functions with a plot in the shape of a letter S, is commonly found in biological processes [141]. The initial part of a sigmoidal curve describes the growth of a young organism, and can be approximated by an exponential function:

$$g(F, \tau) = Ae^{B\tau}. \tag{5.7}$$

The objective is to find values of parameters A and B that satisfy equation (5.6) for $k = 0, 1$. By substituting equation (5.7) into (5.6), we obtain

$$AB^k e^{1.2B} = AB^k e^{0.2B} + AB^k, \tag{5.8}$$

which implies that any solution of this equation yields continuity of infinite order. Solution of equation (5.8) for any value of k yields $B \approx 0.632$. Parameter A is a scaling factor and can be chosen arbitrarily. L-system 12 can be modified to use this growth function with $A = 30$ by changing the definition of g(a) as follows:

```
#define g(a) 30*exp(0.632*a)          /* Exponential growth function */
```

The corresponding diagrammatic representation of development is shown in Figure 5.16b. The right hand edge of the diagram, representing the growth rate of the whole structure, is a smooth exponential curve.

This method for creating smooth animations using parametric L-systems can be extended to context-free branching models by applying continuity criteria along each branching axis. Further research is required to determine its applicability in the context-sensitive case. For instance, in a diffusion-based model, the calculation of the exact moment within a given time step that the concentration exceeds a threshold, triggering the application of a production, can be difficult, but is necessary to ensure that newly created modules are assigned the appropriate age.

5.8 Diffusion in *Anabaena catenula*

In the simple model of the vegetative growth of *Anabaena* presented in Equation 2.1 on page 16, the vegetative cells divide and produce two daughter vegetative cells. In reality, however, some of these cells differentiate into *heterocysts*, non-reproductive cells hypothesized to produce growth regulating substances. The distribution of heterocysts in the filament forms a pattern characterized by a relatively constant number of vegetative cells separating consecutive heterocysts. How the organism maintains the constant spacing of heterocysts while growing is a question that has been addressed by a number of researchers [6, 20, 28, 107]. In this section, the formalism of parametric L-systems is used to express the mathematical model of this phenomena proposed by De Koster and Lindenmayer [28].

The model is based on a biologically well-motivated hypothesis that the heterocyst distribution is regulated by nitrogen compounds produced by the heterocysts, transported from cell to cell across the filament, and decaying in the vegetative cells. The concentration c of the compounds in a cell is assumed to be uniform within the cell and to vary according to the formula

$$\frac{\mathrm{d}c}{\mathrm{d}t} = D(l - c) + D(r - c) - kc$$

where D is the diffusion constant, l is the concentration in the cell to the left, r is the concentration in the cell to the right, and k is the decay constant. By discretizing this equation, and assuming that $D = k = K$, the change in c in a unit of time can be expressed as

$$\Delta c = K(l + r - 3c).$$

If the compound concentration in a young vegetative cell falls below a specific threshold, the cell differentiates into a heterocyst. This endogenous control mechanism is captured by the parametric L-system given below. By tuning the model parameters until a realistic growth sequence is created, a researcher can obtain estimates of the corresponding physical constants, even though they are not directly observable.

L-system 13: A model of heterocyst development in *Anabaena*

```
#define CH      900             /* High concentration */
#define CT      0.4             /* Concentration threshold */
#define ST      3.9             /* Segment size threshold */
#define K       0.25            /* Diffusion/decay constant */
#include H                      /* Heterocyst shape specification */
#ignore f ~ H
```

ω : $-(90)F(0,0,CH)F(4,1,CH)F(0,0,CH)$

p_1 : $F(s,t,c)$: $t = 1 \,\&\&\, s >= 6$ $\rightarrow$ $F(s/3*2,2,c)f(1)F(s/3,1,c)$

p_2 : $F(s,t,c)$: $t = 2 \,\&\&\, s >= 6$ $\rightarrow$ $F(s/3,2,c)f(1)F(s/3*2,1,c)$

p_3 : $F(h,i,l) < F(s,t,c) > F(o,p,r)$: $c > CT \,||\, s > ST \rightarrow$
$F(s+.1,t,c+K*(l+r-3*c))$

p_4 : $F(h,i,l) < F(s,t,c) > F(o,p,r)$: $!(c > CT \,||\, s > ST) \rightarrow F(0,0,CH) \sim H(1)$

p_5 : $H(s)$: $s < 3$ $\rightarrow$ $H(s*1.1)$

Cells are represented by modules $F(s,t,c)$, where s stands for cell length, t is cell type (0 — heterocyst, 1 and 2 — vegetative types oriented to the left and right, respectively), and c represents the concentration of nitrogen compounds. Productions p_1 and p_2 describe divisions of the vegetative cells into two daughter cells of unequal length, with the ordering of the longer and shorter daughter cells depending on the parent's orientation. The $f(1)$ module provides spacing between the cells for visualization purposes. Production p_3 captures the processes of transportation and decay of the nitrogen compounds. Their concentration c is related to the concentration in the neighbouring cells, l and r, and changes according to the formula $c' = K*(l+r-3*c)+c$ where $K = 0.25$. Production p_4 describes differentiation of a vegetative cell into a heterocyst. The condition specifies that this process occurs when the concentration of nitrogen compounds falls below the threshold value $CT = .4$ in a cell that is young enough, as indicated by its length being below the threshold value $ST = 3.9$. Production p_5 describes the subsequent growth of the heterocyst in size.

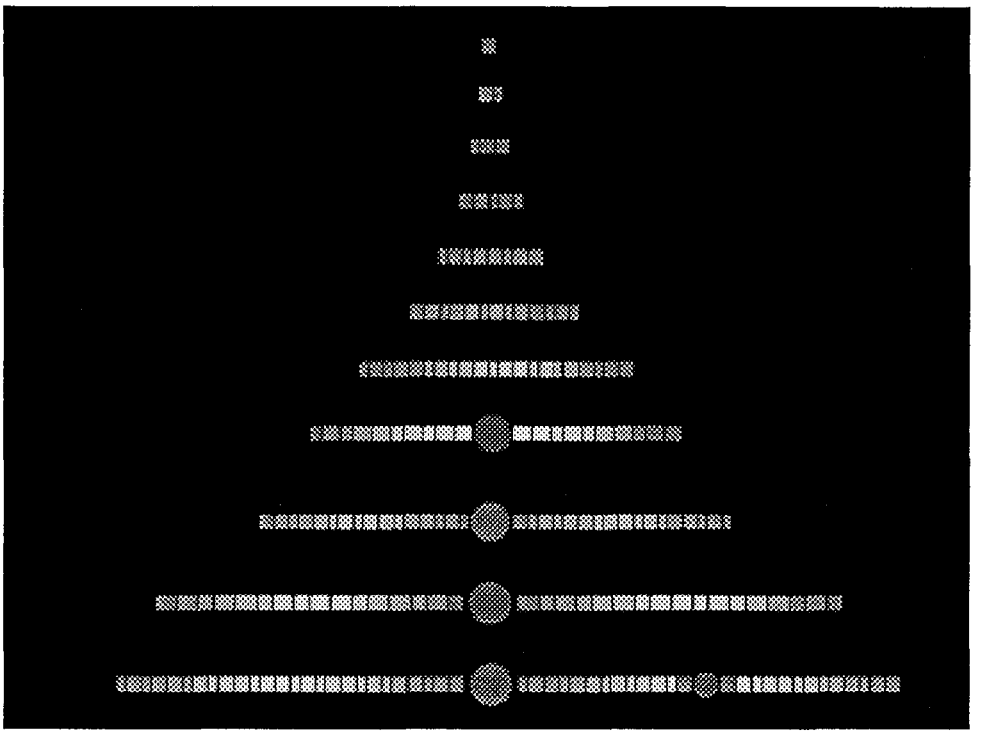


Figure 5.17: Developmental sequence of *Anabaena catenula* with heterocysts

Snapshots of the developmental sequence of *Anabaena* are given in Figure 5.17. The vegetative cells are shown as rectangles, coloured according to their concentration of nitrogen compounds; the higher the concentration, the darker the cell. The heterocysts are represented as disks. The values of parameters K, CH, CT and ST were selected to provide correct distribution of the heterocysts, and correspond closely to the values reported in [28]. The diffusion and decay process has been conveniently captured by the parametric L-system formalism.

5.9 Development of an inflorescence — *Mycelis muralis*

The study of compound flowering structures or *inflorescences* has played a particularly visible role among the applications of L-systems [50, 51, 52, 53, 83, 92]. L-systems with turtle interpretation are a proven tool for realistic image synthesis of inflorescences [61, 119, 123], due to their ability to simulate a variety of lineage and

endogenous control mechanisms. Nevertheless, models expressed using standard L-systems are sometimes convoluted and may not lend themselves to clear presentation. The addition of parameters to the formalism makes it much easier to express complex models, such as the developmental model of the inflorescence of *Mycelis muralis* discussed below.

The development of *Mycelis* is difficult to model for two reasons. First, the plant exhibits a basipetal flowering sequence, which means that flowering starts at the top of the plant and proceeds downwards. Secondly, at some developmental stages the plant has an *acrotonic* structure, where the upper branches are more developed than the lower ones. Both phenomena are in a sense counter-intuitive, since it would seem that the older branches situated near the plant base should start growing and producing flowers before the younger ones at the plant top. To explain these effects, several models were proposed and formally analyzed by Janssen and Lindenmayer [83]. Their *Model II* is restated in the following parametric L-system.

L-system 14: `Mycelis muralis - Model II`

```
#include 0            /* flower shape specification */
#ignore / + ~ 0
ω  : I(20)FA(0)
p1 :    S <  A(t)                  →  T(0)~0
p2 :         A(t)      : t > 0     →  A(t − 1)
p3 :         A(t)      : t == 0    →  [+(30)G]F/(180)A(2)
p4 :    S <  F                     →  FS
p5 :         F > T(c)               →  T(c + 1)FU(c − 1)
p6 : U(c) <  G                     →  I(c)FA(2)
p7 :         I(c)      : c > 0     →  I(c − 1)
p8 :         I(c)      : c == 0    →  S
p9 :         S                      →  ϵ
p10 :        T(c)                   →  ϵ
```

The axiom consists of three components. Modules F and A(0) represent the initial segment and the apex of the main axis. Module I(20) is the source of a signal representing florigen, a hypothesized hormone controlling the development of flowers.

In this case, florigen is sent towards the apex by leaves located at the plant base, which is not included in the model.

The developmental process along the main axis consists of two phases, which are repeated in branches of higher orders. First, the main axis is formed in a process of subapical growth specified by production p_3. The apex produces consecutive segments F at the rate of one segment every three derivation steps (the delay is controlled by production p_2), and initiates branches G positioned at an angle of 30° with respect to the main axis. At this stage, the branches do not develop further, simulating the effect of *apical dominance*, which is the inhibition of branch development during the active production of new branches by the apex.

After a delay of 20 derivation steps, counted by production p_7, an acropetal flower-inducing signal S is sent by production p_8. Production p_4 transports S across the segments at the rate of one internode per step. Since new internodes are produced by the apex at a three times slower rate, the signal eventually reaches the apex. At this point, the second developmental phase begins. Production p_1 transforms apex A(t) into a bud ~O. Further branch production is stopped and a signal T(c) is sent towards the base in order to enable the development of lateral branches. Parameter c is incremented each time signal T(c) traverses an internode, according to production p_5. Subsequently, production p_6 introduces the value of parameter c into the corresponding branches, using module U(c) as a carrier. The successor of production p_6 has the same format as the axiom, thus module I(c) determines the delay between the initiation of branch development and the time that signal S is sent to terminate further internode creation. This delay c is smallest for the top branches and increases towards the plant base. Consequently, parameter c can be interpreted as the *growth potential* of the branches, allowing lower branches to grow longer than the higher ones. On the other hand, the development of the upper branches starts sooner, thus in some stages they will be more developed than the lower ones, and the flowering sequence will progress downwards, corresponding to observations of the real plant [83].

A diagrammatic developmental sequence of *Mycelis muralis* simulated using this L-system is shown in Figure 5.18. Initially, the segments are shown as thin black

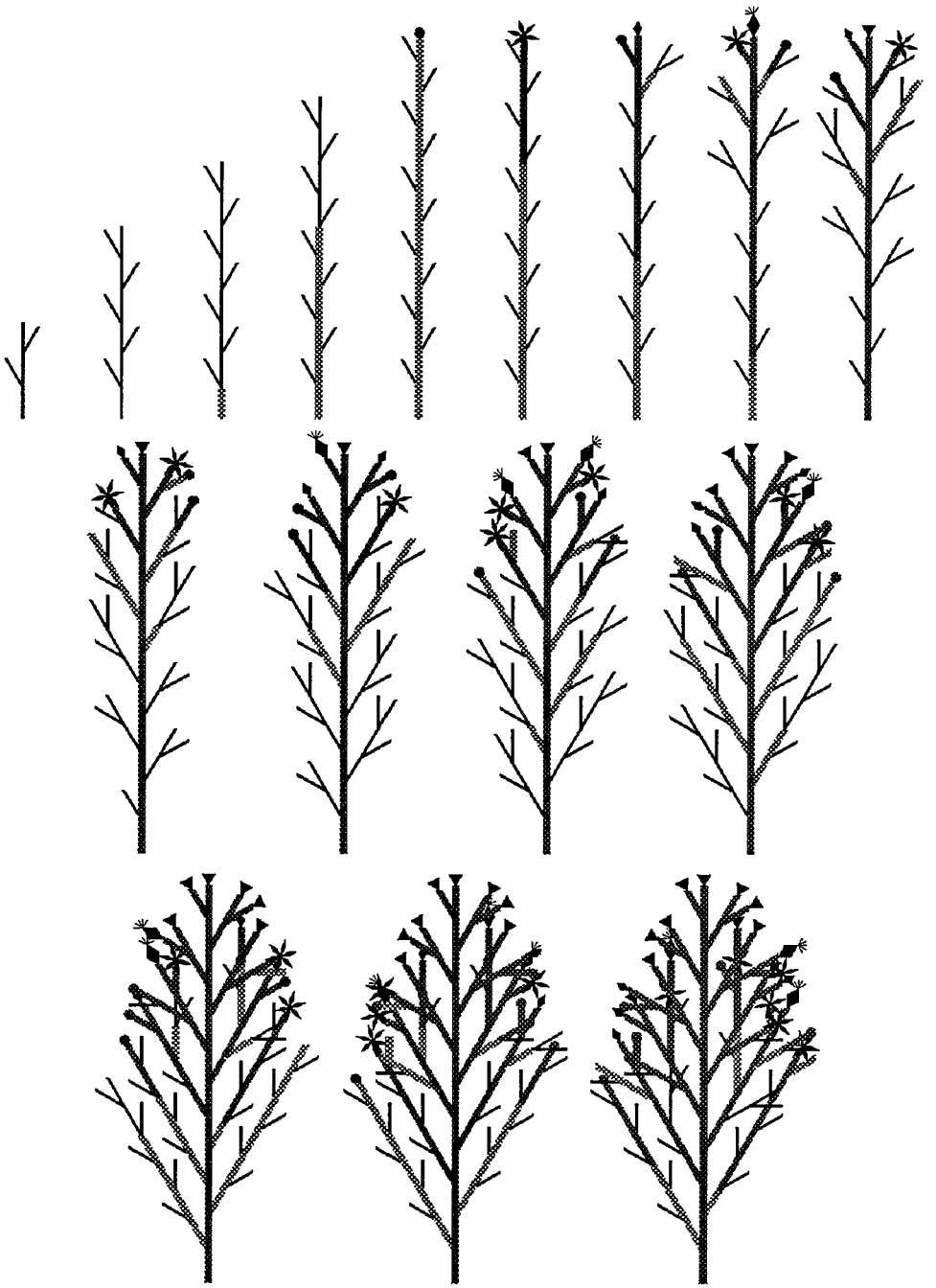


Figure 5.18: Development of *Mycelis muralis*

Figure 5.19: A three-dimensional rendering of the *Mycelis* model ©1987 P. Prusinkiewicz and J. Hanan

lines. The passage of florigen S thickens them and turns them grey, and the lifting of apical dominance darkens the thick lines.

The L-system describing the three-dimensional structure shown in Figure 5.19 and Plate C.5 differs from the two-dimensional model only in details. The angle value associated with the module "/" in production p_3 has been changed to 137.5°, resulting in a spiral arrangement of lateral branches around the mother axis. The leaves subtending branches have been included in the model, and the flowers have been assumed to undergo a series of changes from bud to open flower to fruit.

Another developmental model of *Mycelis*, referred to as *Model III* by Prusinkiewicz and Lindenmayer [122], is captured by the following parametric L-system.

L-system 15: Mycelis muralis - Model III

```
#include O      /* Flower shape specification */
#ignore / + ~ O F
ω  : I(20)FA(0)
p1 : S <  A(t)                 →  TV~O
p2 : V <  A(t)                 →  TV~O
p3 :      A(t)    : t > 0      →  A(t − 1)
p4 :      A(t)    : t == 0     →  M[+(30)G]F/(180)A(2)
p5 : S <  M                    →  S
p6 :      S > T                →  T
p7 : T <  G                    →  FA(2)
p8 : V <  M                    →  S
p9 :      T > V                →  W
p10 :     W                    →  V
p11 :     I(t)    : t > 0      →  I(t − 1)
p12 :     I(t)    : t == 0     →  S
```

The initial phases of development are the same as in model II. First, apex A creates the main axis and initiates lateral branches (productions p_3 and p_4). Symbol M in the successor of production p_4 marks consecutive branching points. After a delay of 20 steps, established in the axiom ω and counted by production p_{11}, flowering signal S is generated at the inflorescence base (p_{12}) and sent up along the main axis (p_5). Upon reaching the apex, S induces its transformation into a terminal flower ~O, and initiates two basipetal signals T and V (p_1). The basipetal signals also can be initiated by production p_2, which is necessary for proper timing in the development of the topmost lateral branch. Signal T propagates basipetally at the rate of one internode per derivation step (p_6) and lifts apical dominance, thus allowing the lateral branches to grow (p_7). The presence of the second basipetal signal V is the distinctive feature of model III. Its role is to enable the formation of flowers on the lateral branches by generating the flowering signal S at their bases (p_8). Since signal V propagates

down the main axis at the rate of one internode per two derivation steps (p_9, p_{10}), the interval between the lifting of apical dominance by signal T and the induction of the flowering signal S by signal V increases linearly towards the inflorescence base. This process allows the lower branches to grow longer than the upper ones, resulting in a structure that is more developed near the base than near the apex in later developmental stages.

The entire control process is repeated for each axis: its apex is transformed into a flower by signal S, the growth of lateral axes is successively enabled by signal T, and the second basipetal signal V is sent to induce the flowering signal S in the next-order axes. Consequently, a basipetal flowering sequence is observed along all axes of the inflorescence.

A comparison of these models reveals that Model II controls the flowering on lateral branches using growth potential c accumulated by signal T on its way down, while model III employs the time interval between signals T and V for the same purpose. Since both models produce identical developmental sequences, it is not possible to decide which one is more faithful to nature without gathering additional data related to plant physiology. Nevertheless, the models clearly indicate that the flowering sequence of *Mycelis* cannot be explained only in terms of a flowering signal and the lifting of apical dominance. Another factor, whether it is an accumulated delay or a third signal, is needed. The mathematical models bring forward evidence and assist in formulating plausible hypotheses related to the control mechanisms that may be employed by nature. Determination of the final answer will require further study of the real plant.

In these two models, parametric L-systems provided a simple mechanism for specifying the series of state changes capturing time delays, both before the release of florigen at the plant base and in the production of internodes by the apex. To effect a change in either delay, the appropriate parameter is incremented or decremented. Additionally, in L-system 14 the parameter c in the downward signal T(c) lifting apical dominance represents the accumulation of growth potential for the lateral branches in a straightforward manner.

5.10 L-systems as a model of parallel computation

Parametric L-systems were introduced to model plants [120, 122], but their domain of application includes standard computational problems as well [121]. This section explores L-systems as a general model of computation, and the relationship of that model to parallel machine architectures. The following examples illustrate the computational possibilities of parametric L-systems.

Example 5.10.1. The L-system

$$\begin{aligned} \omega &: \ \mathtt{A(1,1)} \\ p_1 &: \ \mathtt{A(x,y)} \ \rightarrow \ \mathtt{A(x*y, y+1)} \end{aligned}$$

computes the value of n factorial in n derivation steps:

$$\mathtt{A(1,1)} \Rightarrow \mathtt{A(1,2)} \Rightarrow \mathtt{A(2,3)} \Rightarrow \mathtt{A(6,4)} \Rightarrow \ldots \Rightarrow \mathtt{A(n!, n+1)}.$$

Example 5.10.2. The L-system

$$\begin{aligned} \omega &: \ \mathtt{A(1,1)} \\ p_1 &: \ \mathtt{A(x,y)} \ \rightarrow \ \mathtt{A(y, x+y)} \end{aligned}$$

computes consecutive terms of the Fibonacci series:

$$\mathtt{A(1,1)} \Rightarrow \mathtt{A(1,2)} \Rightarrow \mathtt{A(2,3)} \Rightarrow \mathtt{A(3,5)} \Rightarrow \ldots$$

Example 5.10.3. The following L-system implements the *odd-even-transposition* parallel algorithm for sorting numbers $a_1 a_2 \ldots a_n$ [2, pages 89-91].

$$\begin{aligned} \omega &: \ \mathtt{A(a_1)B(a_2)A(a_3)B(a_4)}\ldots \\ p_1 &: \ \mathtt{A(x)} > \mathtt{B(y)} \ : \mathtt{x} > \mathtt{y} \ \rightarrow \ \mathtt{B(y)} \\ p_2 &: \ \mathtt{A(x)} \ \rightarrow \ \mathtt{B(x)} \\ p_3 &: \ \mathtt{A(x)} < \mathtt{B(y)} \ : \mathtt{x} > \mathtt{y} \ \rightarrow \ \mathtt{A(x)} \\ p_4 &: \ \mathtt{B(y)} \ \rightarrow \ \mathtt{A(y)} \end{aligned}$$

This L-system operates by exchanging parameter values a_i and a_{i+1} if $a_i > a_{i+1}$. Pairs whose first element has an odd-numbered index i are considered in odd-numbered

derivation steps; pairs whose first element has an even-numbered index i are considered in even-numbered steps. This principle is illustrated by the following sample derivation.

$$\begin{aligned}
\omega &: \texttt{A(5)B(4)A(3)B(2)A(1)} \\
\mu_1 &: \texttt{B(4)A(5)B(2)A(3)B(1)} \\
\mu_2 &: \texttt{A(4)B(2)A(5)B(1)A(3)} \\
\mu_3 &: \texttt{B(2)A(4)B(1)A(5)B(3)} \\
\mu_4 &: \texttt{A(2)B(1)A(4)B(3)A(5)} \\
\mu_5 &: \texttt{B(1)A(2)B(3)A(4)B(5)}
\end{aligned}$$

In spite of its simplicity, the parallel odd-even-transposition algorithm sorts n numbers in $O(n)$ time, which is better than the $O(n \log n)$ lower bound for sequential sorting.

Example 5.10.4. The L-system given below generates prime numbers using a parallel version of the *sieve of Eratosthenes.*

$$\begin{aligned}
\omega &: \texttt{N(1)E} \\
p_1 &: \texttt{N(k)} \rightarrow \texttt{N(k+1)A(k+1)} \\
p_2 &: \texttt{A(k)} \rightarrow \epsilon \\
p_3 &: \texttt{A(k)<E} \rightarrow \texttt{P(k)E} \\
p_4 &: \texttt{A(k)<P(n)} \ : \ \texttt{n\%k!=0} \rightarrow \texttt{P(n)A(k)}
\end{aligned}$$

As specified by production p_1, module N(k) generates a series of modules A(k) representing consecutive integers. Each module A(k) propagates to the right through a sequence of modules P(n) representing prime numbers found so far. If A(k) immediately precedes P(n) in a string, and n does not divide k, A(k) moves to the right of P(n). Specifically, production p_4 copies A(k) to the right side of P(n), while p_2 removes its previous occurrence. If n divides k evenly, production p_2 acts alone and removes A(k) from the string. Thus, A(k) reaches the end of the string marked by module E if and only if k is prime. A(k) is then transformed to P(k) by production p_3. These operations are illustrated by the following derivation steps.

$$\begin{aligned}
\mu_8 &: \texttt{N(9)A(9)P(2)P(3)A(7)P(5)E} \\
\mu_9 &: \texttt{N(10)A(10)P(2)A(9)P(3)P(5)A(7)E} \\
\mu_{10} &: \texttt{N(11)A(11)P(2)P(3)P(5)P(7)E}
\end{aligned}$$

As shown by the above examples, the most obvious characteristic of computation using parametric L-systems is the *monadic* nature [153] of production application. Information is passed *unilaterally* from the modules specified by the production predecessor to the modules in the successor. This contrasts L-systems with applicative languages, such as C or Lisp, in which computation occurs in *dyads*: a function is called, then returns a value to the calling environment.

The relationship between L-systems and the *SIMD* model of parallel computation [2, Chapter 1] is illustrated by Example 5.10.3. The string of modules corresponds to a collection of processors which act according to the same production set (*S*ingle *I*nstruction), but hold different parameters (*M*ultiple *D*ata). In this case, the processors are arranged in a linear array in which the neighbours can communicate. Since each production creates one module, the array contains a constant number of processors connected in a fixed way.

An extension to the SIMD model is suggested by Example 5.10.4, which allows productions to create new modules and destroy existing ones. The corresponding parallel architecture can be envisioned as a linear array of processors which may *self-replicate* and *die* in the course of computation. Such a model can be simulated using existing computers; an implementation of L-systems on a Connection Machine, proposed by R. and S. Pinter [115], is the closest approximation of this model to date. A separate processor is allocated to each module of the generated string. After a derivation step, the processors are reallocated to accommodate productions that create or delete modules. As reallocation may dominate the computation time, conclusions drawn from this implementation cannot be automatically extrapolated to the original model.

The addition of parameters and mathematical expressions to the L-system formalism extends the range of applications from biological simulation and visualization to more traditional computational problems. Parametric L-systems provide a model for parallel computation with self-replicating processors, and are worthy of further research as a potential foundation for a class of programming languages based on monadic computation and concurrency.

5.11 Summary

The examples presented in this chapter illustrate the suitability of parametric L-systems for a broad range of applications. The addition of parameters to the L-system formalism has extended the range of models that can be expressed and provides a tool that is more flexible than standard L-systems.

The models of *Anabaena catenula* presented in Section 5.1 demonstrate the use of parameters to capture geometric and cell-state information.

The phyllotaxis models of L-systems 3 to 5 rely on parameters for the exact specification of angles and distances necessary for the geometric description of their spiral patterns. Additionally, in L-systems 4 and 5, mathematical expressions are used to change the values of parameters gradually over time.

Although no detailed models of particular tree species have been developed to date, L-system 6 demonstrates that the branching structure of trees can be captured using the parametric L-system formalism.

The developmental models of plant organs in L-systems 8 through 11 are controlled by lineage mechanisms captured in the parameters expressing growth potential, developmental delay, and growth rates. The interactive manipulation of these parameters allows the creation of a family of shapes of each type. A sequence of images illustrating the development of an organ can be created by interpreting the L-system string after each derivation step.

In order to create smooth animations of plant growth using parametric L-systems, the concept of age-controlled L-systems was introduced in Section 5.7. L-system 12 provides an example of the use of parameters to control the progression of time and for the expression of continuous growth functions.

The description of endogenous models of plant development is illustrated in Sections 5.8 and 5.9. L-system 13 models the diffusion and decay of nitrogenous compounds in a *Anabaena catenula* filament. The models presented in L-systems 14 and 15 combine growth potentials, delays, and signals moving through a growing structure to describe the flowering sequence of *Mycelis muralis*.

These examples illustrate the wide range of biological problems that can be modelled, analyzed, and visualized using parametric L-systems. As revealed in Section 5.10, parametric L-systems can also be applied to standard computational problems, and can be considered as a computational model for parallel machines.

Chapter Six

FURTHER EXTENSIONS

As illustrated in Chapter 5, parametric L-systems can be used to model a wide variety of plant architectures. The practical experience stemming from the development of these and other models [62, 120, 121, 122] led to further extensions of the formalism. Their focus is on increased flexibility and efficiency in model specification and operation. Extensions have been made in three areas:

- specification of hierarchical models, making it simple to reuse previously developed L-system components,
- a straightforward method for removing groups of modules that represent structures no longer required in a model, and
- incorporation of features of common computer languages, which enhance parametric L-systems as a plant modelling mini-language.

6.1 Hierarchical modelling

During the modelling process, it often becomes apparent that a part of the new plant, such as a leaf or petal, has similar structure to a previously modelled part of a different plant. A great deal of time and effort can be saved by incorporating the corresponding production rules into the current model. However, the integration of predefined components into a single L-system is not without difficulties. There may be collisions between symbols in the L-systems to be merged; for example, the module

A representing the apex of a branch will not necessarily develop the same way as the A representing the apex of a leaf. The reused components may have incompatible sizes and operate on different time scales. These problems can be addressed using a brute force approach, by renaming all conflicting symbols and changing parameter values in the component L-systems. Rather than rely on this error-prone, ad hoc process, I propose the extension of parametric L-systems with a *sub-L-system* mechanism that insulates the component's modules from those of the remainder of the model, and limits the scope of its production rules to a substring of the current parametric word. Sub-L-systems are invoked directly from the "higher level" L-system in a manner similar to subroutines in a computer program. This yields a hierarchical model with nested sub-L-systems. Cyclic nesting is allowed, as discussed in Section 6.1.3.

6.1.1 Context-free sub-L-systems

In the context-free case, the specification of a parametric L-system with sub-L-systems is similar to the specification of a 0L-system. However, instead of a single set of productions, there is a set of productions for the overall structure, called the *main* L-system, plus a set of productions for each independently modelled plant component. In order to produce a string representation of the component, the main L-system introduces modules representing the axiom of the sub-L-system into the string, delimited by the modules ?(id) and \$. The parameter of the ? module identifies the sub-L-system to be applied to these modules. The substring delimiters must occur in matching pairs in the same way as parentheses are used in an arithmetic expression, and may be nested within the string. The newly introduced sequence of modules will be replaced in subsequent steps by the results of successive derivation steps of the identified sub-L-system. Thus, the parametric string is partitioned into substrings; those modules enclosed in a pair of delimiters ? and \$ are subject to the rules of the identified sub-L-system, while the remainder are governed by the rules of the main L-system.

The following L-system illustrates the operation of the sub-L-system mechanism in the context-free case, using a simple branching structure as an example. Note that the axiom ω of the sub-L-system does not play a role in the model. It is used when

the component L-system is being developed and tested.

```
L-system 16:  A simple branching structure

Lsystem: 1      /* Main L-system */
ω  :  A
p11 :  A  →  I[?(2)A$]A
endLsystem

Lsystem:  2     /* Sub-L-system for the branch */
ω  :  A
p21 :  A  →  IA
endLsystem
```

Starting from the axiom A, the following parametric words are generated in the first five derivation steps.

```
I[?(2)A$]A
I[?(2)IA$]I[?(2)A$]A
I[?(2)IIA$]I[?(2)IA$]I[?(2)A$]A
I[?(2)IIIA$]I[?(2)IIA$]I[?(2)IA$]I[?(2)A$]A
I[?(2)IIIIA$]I[?(2)IIIA$]I[?(2)IIA$]I[?(2)IA$]I[?(2)A$]A
```

In each step, the production that is applied to a module A depends on the sub-L-system identified by the delimiters immediately enclosing it. Thus, production p_{11} is applied to the module A appearing at the right of each word, producing an internode I, a lateral branch incorporating the sub-L-system reference ?(2)A$, and a new apex A. Production p_{21} is applied to the modules A appearing in each branch, producing an internode I and a new apex A. No further branching occurs in the lateral branches, since production p_{11} is not applicable within the scope of any reference to L-system 2.

The sub-L-system mechanism presented above addresses the issue of collision of symbols used to denote modules, but still requires the user to harmonize the geometric scales of component sub-L-systems. Rather than forcing the user to do this

Figure 6.1: Model of the sedge *Carex laevigata* ©1989 J. Hanan and P. Prusinkiewicz

on a module by module basis, the sub-L-system mechanism is further extended by introducing a global scaling factor as the second parameter of the ? module, giving it the format ?(*id,scale*). Interpretation of the resulting strings is straightforward; commands in a delimited substring have the same turtle interpretation as before, but the dimensions of the graphical objects produced are scaled by the product of the scaling factors of nested sub-L-systems. The main L-system is assumed to have a scaling factor of 1.

As an example, consider the model of the sedge *Carex laevigata* shown in Figure 6.1 and in Plate C.6. The underlying L-system incorporates sub-L-systems for the male and female inflorescences, and a developing polygonal leaf model.

L-system 17: Sub-L-system based model of the sedge *Carex laevigata*

```
Lsystem: 1                /* Main L-system for the sedge */
#define IR 1.02             /* Internode growth rate */
#define SW .0075            /* Initial width of the stem */
#define SWR 1.06            /* Stem width growth rate */
ω  : /(30)+(10)#(SW)A(1)
p11: A(a)  : a == 30     → F(1)/(137.5)?(3,1.25)x$
p12: A(a)  : a%10 == 0   → F(1)/(137.5)[L(a)][S(a)]#(SW)A(a+1)
p13: A(a)                → F(1)A(a+1)
p14: #(d)  : d < 200     → #(d*SWR)
p15: S(a)                → ∧(25)'(-.1)!(.3)F((30-a)*.5)'(0)?(2,1.25)x$
p16: L(a)                → ∧(60)!(.1)?(4,1)x(a-10,a,(a-13)/100)$
p17: F(t)  : t < 2       → F(t*IR)
p18: F(t)  : !(t < 2)    → F(t*IR/2)F(t*IR/2)
endLsystem

Lsystem: 2                /* Sub-L-system for the female spike */
#include S G               /* Surface definitions for a seed */
#define FIR 1.01          /* Female internode growth rate */
#define FSR 1.05          /* Female seed growth rate */
ω  : F(5)x
p21: x                   → [&(30)~G(2.25)F(.5)]F(.1)/(180)
                           [&(30~G(2.25)F(.5)]/(137.5)A(0)
p22: A(t)  : t < 75      → F(.2)[B]/(137.5)A(t+1)
p23: B                   → &(35)[~C(1)][~G(1)#(1)F(.5)]
p24: F(t)  : t < 1       → F(t*FIR)
p25: &(a)  : a < 50      → &(a*FSR)
p26: S(t)  : t < 2       → S(t*FSR)
p27: G(t)  : t < 2       → G(t*FSR)
endLsystem
```

(L-system 17 continued)

```
Lsystem:  3        /* Sub-L-system for the male spike */
#include M                  /* Male surface definition */
#define MIR 1.025           /* Internode growth rate */
#define MSR 1.02            /* Seed growth rate */
ω  :  F(5)x
p31 :  x                          →  A(0)
p32 :  A(t)       :  t < 55       →  F(.2)[B]/(137.5)A(t + 1)
p33 :  B                          →  ∧(5)[~M(1)]
p34 :  F(t)       :  t < 1        →  F(t * MIR)
p35 :  ∧(a)       :  a < 15       →  ∧(a * MSR)
p36 :  M(t)       :  t < 3        →  M(t * MSR)
endLsystem
```

```
Lsystem:  4        /* Sub-L-system for the leaf */
#define W .05               /* Starting width */
#define WR 1.04             /* Width growth rate */
#define L .16               /* Starting length */
#define LR 1.05             /* Length growth rate */
ω  :  #(.1)x(1, 10, −.02)
p41 :  x(S, A, T)  :  S > 0       →  x(S − 1, A, T)
p42 :  x(S, A, T)  :  !(S > 0)    →  [A(S, S + A, T)][B(S, S + A, T)]
p43 :  f(n)        :  n < 15      →  f(n * WR)
p44 :  F(n)        :  n < 15      →  F(n * LR)
p45 :  A(a, A, T)  :  a == A      →
                      '(T){[−(90)f(W)].{.F(L)/(45)[−(90)f(W)}.}]; A(a + 1, A, T)
p46 :  B(a, A, T)  :  a == A      →
                      '(T){[+(90)f(W)].{.F(L)/(45)[+(90)f(W)}.}]; B(a + 1, A, T)
p47 :  A(a, A, T) →   {[−(90)f(W)].{.F(L)[−(90)f(W)}.}]; A(a + 1, A, T)
p48 :  B(a, A, T) →   {[+(90)f(W)].{.F(L)[+(90)f(W)}.}]; B(a + 1, A, T)
endLsystem
```

The development of the branching structure is controlled by the main L-system. Productions p_{11} to p_{13} model the growth of the main stem of the plant, with leaves L and branches S initiated every 10 steps by production p_{12}. In step 30, production p_{11} initiates the creation of the male spike using the sub-L-system reference ?(3, 1.25)x$. Production p_{15} models a side branch tipped by a female spike, which is specified by the modules ?(2, 1.25)x$. Production p_{16} inserts the sub-L-system for the leaf, using the reference ?(4, 1)x(a − 10, a, (a − 13)/100)$.

Sub-L-systems 2 and 3 are based on the model of cylindrical phyllotaxis first presented in L-system 5.2.1 on page 62. The tapered shape is created by the continued growth of spikelets after they have been initiated.

Sub-L-system 4, which controls the growth of the leaf, has three parameters in its axiom. The first represents a delay between a leaf's initiation and the start of its growth. The remaining two parameters control the bending of the leaf due to gravity through the application of productions p_{45} and p_{46}. The leaf is modelled using the developmental surface technique described in Section 3.3.2 on page 38.

Note that the module x is specified as the axiom in all three sub-L-systems references, but that very different productions are applied in each case.

6.1.2 Time-scaled sub-L-systems

The sub-L-system mechanism presented to this point does not address the issue of harmonizing time scales among component L-systems. For instance, how do you model the development of a tree, where a derivation step may represent a year's growth, while at the same time modelling leaf development, where one step represents a few hours' growth. The simplest approach to the combination of such L-systems by hand is to introduce delays into the L-system that has the higher rate of development. The same effect is achieved for components modelled by age-controlled L-systems (Section 5.7) by adjusting the time step constant. An alternative to the hand-coding of delays is captured by a further extension of sub-L-systems, as follows.

In a *time-scaled* sub-L-system, a relative time scale is specified as the third parameter of the start module ?, indicating the number of derivation steps to be processed in the sub-L-system for each step in the higher level one. This parameter can be

assigned any non-negative real value, as in the following example.

```
L-system 18:  Time-scaled sub-L-system example

Lsystem: 1   /* Main L-system */
#define SR 1   /* Sub-Lsystem rate */
ω  :  A
p11 :  A  →  I[?(2,1,SR)B]A
endLsystem

Lsystem: 2     /* Sub-L-system */
ω  :  B
p21 :  B  →  IB
endLsystem
```

The defined constant SR determines how many derivation steps will be performed by sub-L-system 2 for each step in the main L-system. Starting from the axiom A, the following strings will be produced in the first few derivation steps. The sequence in the left column is produced for SR = .5, in the middle for SR = 1, and in the right column for SR = 2. The complete sequence for the central column is the same as for L-system 16 on page 107, with the sub-L-system reference ?(2) replaced by ?(2, 1, 1) and the module A replaced by B. The ellipsis, $\cdots$, represents modules that have been left out of the list for clarity.

I[?(2,1,.5)B$]A	I[?(2,1,1)B$]A	I[?(2,1,2)B$]A
		I[?(2,1,2)IB$]A
I[?(2,1,.5)B$]I[$\cdots$	I[?(2,1,1)IB$]I[$\cdots$	I[?(2,1,2)IIB$]I[$\cdots$
		I[?(2,1,2)IIIB$]I[$\cdots$
I[?(2,1,.5)IB$]I[$\cdots$	I[?(2,1,1)IIB$]I[$\cdots$	I[?(2,1,2)IIIIB$]I[$\cdots$
		I[?(2,1,2)IIIIIB$]I[$\cdots$
I[?(2,1,.5)IB$]I[$\cdots$	I[?(2,1,1)IIIB$]I[$\cdots$	I[?(2,1,2)IIIIIIB$]I[$\cdots$
		I[?(2,1,2)IIIIIIIB$]I[$\cdots$
I[?(2,1,.5)IIB$]I[$\cdots$	I[?(2,1,1)IIIIB$]I[$\cdots$	I[?(2,1,2)IIIIIIIIB$]I[$\cdots$

Figure 6.2: Developmental sequence of a spiral

In each derivation step of the main L-system, production p_{11} produces a new internode I and a new branch [?(2, 1, SR)B$]. Only the first two internodes and the first branch are shown. The growth of the branch is controlled by production p_{21} in conjunction with the sub-L-system rate SR. In the central column, with SR = 1, the branch grows by one module I for every step of the main L-system. In the left-hand column, with SR = .5, the branch grows by one module I for every 2 steps of the main L-system. In the right-hand column, with SR = 2, the branch grows by 2 I's in each step of the main L-system. Thus, the growth rate of the branch can be controlled by modification of the sub-L-system time scale.

6.1.3 Cyclic references to sub-L-systems

An aspect of time-scaled sub-Lsystems that can cause difficulties is illustrated by the following L-system, which models the simple self-similar spiral structure shown in Figure 6.2.

```
L-system 19:  A spiral

Main Lsystem: 19
#define SR 1   /* Sub-L-system rate */
ω : A
p1 : A → F?(1,.8,SR)+A$
endLsystem
```

This L-system operates by referencing itself to produce a scaled down line segment at an angle of 75° to the previous segment. Given that the time scale SR = 1, evaluation

proceeds correctly, producing the following strings in the first few derivation steps.

$$\begin{aligned}
&\mathtt{A}\\
&\mathtt{F?(1,.8,1)+A\$}\\
&\mathtt{F?(1,.8,1)+F?(1,.8,1)+A\$\$}\\
&\mathtt{F?(1,.8,1)+F?(1,.8,1)+F?(1,.8,1)+A\$\$\$}\\
&\mathtt{F?(1,.8,1)+F?(1,.8,1)+F?(1,.8,1)+F?(1,.8,1)+A\$\$\$\$}\\
&\cdots
\end{aligned}$$

If the time scale is increased, the cyclic reference will cause an infinite loop. For instance, if $\mathtt{SR} = 2$ the following strings will be produced.

$$\begin{aligned}
&\mathtt{A}\\
&\mathtt{F?(1,.8,2)+A\$}\\
&\mathtt{F?(1,.8,2)+F?(1,.8,2)+F?(1,.8,2)+F?(1,.8,2)+F}\cdots
\end{aligned}$$

Each time a sub-L-system reference $\mathtt{?(1,.8,2)+A\$}$ is processed, a new identical reference is introduced in the first sub-step, which is itself processed in the second sub-step, continuing the cycle. The module $ at the end of the string is never reached, and sub-L-system processing continues to the limit of machine resources. This can be prevented by modifying the production rules, either to reduce the time scale value in each step or to provide a stopping condition, or both, as in the following L-system.

```
L-system 20:  Spiral 2

Lsystem: 1
ω : A(3)
p1 : A(a) : a > 0 → F?(1,.8,a)+A(a-1)$
endLsystem
```

In conjunction with the decreasing value of module A's first parameter, the condition $\mathtt{a} > 0$ prevents the introduction of sub-L-systems once the limit is reached, as can be seen in the following sequence of strings, which are created in 3 derivation steps of

the main L-system.

$$\begin{aligned}
&\texttt{A(3)}\\
&\texttt{F?(1,.8,3)+A(2)\$}\\
&\texttt{F?(1,.8,3)+F?(1,.8,2)+F?(1,.8,1)A(0)\$\$\$}\\
&\texttt{F?(1,.8,3)+F?(1,.8,2)+F?(1,.8,1)A(0)\$\$\$}
\end{aligned}$$

In the second derivation step of the main L-system, the modules +A(2) are processed in 3 sub-steps, as specified by the third parameter of the sub-L-system reference. In the first sub-step, the modules +F?(1, .8, 2)A(1)$ are produced. In the second sub-step, the newly introduced substring A(1) produces the modules +F?(1, .8, 1)A(0)$. In the third substep no further changes take place, as the condition in p_{11} evaluates to false and no more sub-L-system references are introduced. The use of the same decreasing parameter of the module A as the time scale for new sub-L-systems will also end the recursion when the value becomes 1, resulting in the same string.

This "recursive" mechanism is interesting from a theoretical point of view (cf. Wyvill [157]) and is useful for hierarchical modelling of self-similar structures, but its relation to biological mechanisms is not certain at this time.

6.1.4 Context-sensitive sub-L-systems

As discussed to this point, sub-L-system operation has been restricted to the context free case. The introduction of sub-L-systems to context sensitive parametric L-systems is straightforward. The context matching procedures are not changed, and the sub-L-system delimiters are typically included in the list of symbols to be ignored. When information must be transferred across a sub-L-system boundary, either the same letter must be used to represent the signal in all components, or productions must be added to recognize the different signals from various components. This is

illustrated in the following example.

```
L-system 21:  Context sensitive sub-L-system example

Lsystem: 1          /* Main L-system */
#ignore ? $
ω  : SI?(2)X$I
p11 : S < I → S
p12 :     S → I
endLsystem

Lsystem: 2          /* Sub-L-system 2 */
#ignore ? $
ω  : X
p21 : Z < X → Z
p22 :     Z → X
endLsystem
```

The following strings represent the axiom and the results of the first 3 derivation steps of this L-system.

```
SI?(2)X$I
IS?(2)X$I
II?(2)X$I
II?(2)X$I
```

The signal S cannot be transmitted through the sub-L-system string, as production p_{21} only recognizes Z in its left context. In order for the signal to be passed, production p_{22} can be converted to recognize the external signal as in the following sub-L-system.

```
Lsystem: 2'         /* First alternative to sub-L-system 2 */
#ignore ? $
ω  : X
p21 : S < X → S
p22 :     S → X
endLsystem
```

Alternatively, new productions could be added to recognize the signal, as in the following sub-L-system.

```
Lsystem: 2''          /* Second alternative to sub-L-system 2 */
#ignore ? $
ω  : X
p21 : Z < X → Z
p22 :     Z → X
p23 : S < X → S
p24 :     S → X
endLsystem
```

Either option would produce the following sequence of strings.

```
SI?(2)X$I
IS?(2)X$I
II?(2)S$I
II?(2)X$S
```

Further research is required to establish a general mechanism facilitating the harmonization of signals between sub-L-systems.

The passage of information across sub-L-system boundaries is also a problem when time-scaled sub-L-systems are used. If a sub-L-system is running faster than the L-system in effect for a neighbouring module, a message may be created and disappear before it can be considered as context by that module. On the other hand, a module within the sub-L-system may react to a single signal coming across the boundary more than once in a single derivation step of the main L-system. Analogous problems exist for sub-L-system that run slower than the main L-system. These situations can

be illustrated as follows.

```
L-system 22:  Context-sensitive time-scaled sub-L-system example

Lsystem: 1        /* Main L-system */
#define SR 1      /* Sub-Lsystem rate */
#ignore ? $
ω  :  SI?(2,1,SR)X$I
p11 :  S <  I  →  S
p12 :       S  →  I
endLsystem

Lsystem: 2          /* Sub-L-system */
#ignore ? $
ω  :  X
p21 :  S <  X  →  S
p22 :       S  →  Y
p23 :  S <  Y  →  Z
endLsystem
```

As before, the constant SR determines how many derivation steps will be performed in sub-L-system 2 for each step in the main L-system. The following strings will be produced in the first 2 derivation steps of the main L-system.

S?(2,1,.5)X$I	S?(2,1,1)X$I	S?(2,1,2)X$I	S?(2,1,4)X$I
			S?(2,1,4)S$I
		S?(2,1,2)S$I	S?(2,1,4)Y$I
			S?(2,1,4)Z$I
I?(2,1,.5)X$I	I?(2,1,1)S$I	I?(2,1,2)Y$S	I?(2,1,4)Z$I
			I?(2,1,4)Z$I
		I?(2,1,2)Y$I	I?(2,1,4)Z$I
			I?(2,1,4)Z$I
I?(2,1,.5)X$I	I?(2,1,1)Y$S	I?(2,1,2)Y$I	I?(2,1,4)Z$I

The sequences in the columns from left to right are produced for SR = .5, SR = 1, SR = 2, and SR = 4, respectively. In the left hand column, with SR = .5, the signal S is not received by the slower sub-L-system, as it is removed by production p_{12} before it can be recognized. In the second column, with SR = 1, the signal is passed through the string and the module X is converted to a Y by production p_{22}. In the third column, with SR = 2, the signal is passed through in a single step of the main L-system. In the right hand column, with SR = 4, the signal is not transmitted, as it is created and disappears before it can be recognized by the main L-system symbols. In addition, the presence of the signal for longer than 2 steps of the sub-L-system causes it to be received twice, transforming the module Y into a Z according to production p_{23}.

These problems are compounded for neighbouring sub-L-systems operating at different rates, or when using fractional time scales. An ad hoc solution can be achieved by establishing modelling conventions to be applied by the user in these situations. A more general solution is related to asynchronous and locally synchronous development of plant modules [72] and is worthy of further research.

6.1.5 Summary

In summary, the sub-L-system mechanism has been introduced to enable the creation of hierarchical models, allowing straightforward reuse of previously developed L-system components as part of a single model. The mechanism isolates module and production definitions within a sub-L-system, and provides a means for harmonizing geometric scales among components. In the context-free case, different time scales of the component models can be resolved using a time scaling factor. If cyclic references to sub-L-systems are made, care must be taken to avoid infinite processing "loops". Context-sensitive models can be developed, and information can be passed across sub-L-system boundaries if a modelling convention is established and followed, but a more general solution will require investigation of asynchronous communications among modules.

Aside from having value as a tool for hierarchical modelling, the sub-L-system mechanism can be used to implement a form of the table L-systems concept first presented by Rozenberg [132]. Table L-systems have a number of sets of productions

instead of just one. An external mechanism is used to select the set of productions applicable to the entire string in any given step. This allows the simulation of environmental effects which modify the way a plant develops, such as the shortening of daylight hours as the growing season progresses. Table L-systems can be implemented using sub-L-systems by enclosing the entire string in a sub-L-system reference and having the table switching controlled by the main L-system.

6.2 Substring removal

In plant development there is often a progression of very different forms, for instance from bud to flower to fruit. Whether this is modelled by replacing a complete structure by another, or by metamorphosis of existing sub-structures while others, such as petals, drop off, there needs to be a mechanism to remove a large number of symbols from the string in a single step, without requiring productions for each symbol. This function is performed by the cut symbol %. Whenever it is detected in the string during the generation process, it and all following symbols up to the closest unmatched right bracket] are ignored for derivation purposes. If an unmatched right bracket is not found, symbols are ignored until the end of the string. Consequently, the string resulting from a derivation step does not contain successors of the ignored modules. However, the modules are not ignored in the context matching process.

A simple example of the use of the cut symbol is illustrated by the following L-system.

```
L-system 23:  A simple flower
#include B              /* bud surface definition */
#include C              /* flower center surface definition */
#include S              /* Seed surface definition */
ω  :  A(1)~B
p1 :  A(a)  :  a == 1  →  F[P]A(2)%
p2 :  A(a)  :  a == 2  →  ~C
p3 :  C                →  S
p4 :  P                →  %[~P]/(72)[~P]/(72)[~P]/(72)[~P]/(72)[~P]
```

The axiom represents a flower bud ~B, preceded by a module A which controls its development. In the first derivation step, production p_1 creates a stem F, initiates petals P, changes its own state to A(2), and inserts a % into the string in preparation for bud removal. The resulting string is F[P]A(2)%~B. In the second derivation step, production p_2 converts the module A into ~C representing the reproductive organs making up the central part of the flower. The modules %~B are removed by the action of the cut symbol. Five petals ~P are positioned by production p_4, which also introduces a module % so that the petals will be removed in the next step. In the third step, the flower center C will be converted to a seed S by production p_3.

The cut symbol provides a simple mechanism for removing a sequence of modules from an L-system string. It can be used to capture natural transformations within a plant, as in the example above, or to model traumatic effects such as pruning or branch death.

6.3 L-systems as a plant modelling mini-language

L-systems with turtle interpretation can be considered a mini-language for expressing plant models to be visualized on a computer screen. As illustrated by the examples in Section 5.10, the parametric extensions presented in this dissertation broaden the range of applications of this mini-language to include more standard computational problems. In this context, a comparison of parametric L-systems to standard computer languages was made, which led to the extensions of parametric L-systems presented in this section. These extensions incorporate variables local to productions, global variables for reading and writing, and new processing capabilities to create a more flexible and efficient mini-language.

6.3.1 Variables local to productions

The first feature of computer languages examined was the use of variables in a production, other than those specified as formal parameters in the predecessor. For

example, consider the following production from L-system 6 on page 69.

$$p_1 : \ \mathtt{A(s,w)} \ \rightarrow \ \mathtt{!(w)F(s)[\&(a0)B(s*r2,w*wr)]/(d)A(s*r1,w*wr)}$$

It calculates the value `w*wr` twice, once in module `B` and again in module `A`. This can be avoided by introducing a temporary variable local to the production, and setting its value in an assignment statement, as shown below.

$$p_1' : \ \mathtt{A(s,w)} \ : \{\mathtt{T = w*wr;}\} \ \rightarrow \ \mathtt{!(w)F(s)[\&(a0)B(s*r2,T)]/(d)A(s*r1,T)}$$

This production can be substituted for p_1 in L-system 6. Since there will be a total of 2^n `A`, `B`, and `C` modules in the string after the nth derivation step, the use of this production will result in 2^n calculations in the next step rather than 2^{n+1}.

In general, an *assignment statement* in a production p takes the form $t = \mathcal{E}(\Sigma)$; where Σ is the set of formal parameters for the L-system, $t \in \Sigma$, and $\mathcal{E}(\Sigma)$ is a correctly formed expression with formal parameters from Σ that have appeared in the predecessor of p. The condition and expressions in the successor of p may incorporate local variables previously defined in the production. Syntactically, the statements can be inserted into a production in the positions shown below:

$$\underline{\eta_l} < \underline{a} > \underline{\eta_r} : \ \{\alpha\} \ C \ \{\beta\} \ \rightarrow \underline{\chi}$$

where α is a list of assignment statements processed after the predecessor has been matched and the parameters bound, and β is a list of assignment statements processed only if the condition evaluates to true and the production is selected for application.

6.3.2 Global variables read by productions

The computation required in a derivation step may be further reduced by introducing *global variables* whose value can be accessed by the expressions in a production. Considering L-system 6 as a whole, it can be seen that modules of types `A`, `B`, and `C` synchronously change their respective width parameter using the same formula. This fact is exploited in the following L-system.

```
L-system 24:  A more efficient version of Honda's model for trees
#define r1 0.9       /* contraction ratio for the trunk */
#define r2 0.6       /* contraction ratio for branches */
#define a0 45        /* branching angle from the trunk */
#define a2 45        /* branching angle for lateral axes */
#define d  137.5     /* divergence angle */
#define wr 0.707     /* width decrease rate */
{W = 10;}
ω : A(1)
{W = W * wr;}
p1 : A(l) → !(W)F(l)[&(a0)B(l * r2)]/(d)A(l * r1)
p2 : B(l) → !(W)F(l)[−(a2)@VC(l * r2)]C(l * r1)
p3 : C(l) → !(W)F(l)[+(a2)@VB(l * r2)]B(l * r1)
```

The list of statements enclosed in braces appearing before the axiom is executed once at the start of the derivation process. In this case, the single assignment statement initializes the global variable W capturing branch width to its starting value of 10. The statement list that appears after the axiom is executed at the beginning of each derivation step. In this case, W is multiplied by the width decrease rate wr. This operation is performed only 1 time per derivation step, instead of the 2^n times it is performed when only local variables are used. In addition, this approach saves the storage of 2^{n+1} parameters in the next string. Note that the same cannot be done for the lengths, since length is not consistent throughout a string, even among modules of one type.

In addition to efficiency related uses, variables accessible to all productions may be employed to represent the environment in which the simulation takes place. For instance, such variables could represent the number of sunshine hours in a day or the concentration of a nutrient in the growth medium.

6.3.3 Global variables written by productions

In order to collect general information about a model, individual productions are allowed to alter the value of a global variable. For example, a global variable can be used to count the number of occurrences of a particular module type in a string. However, this type of operation alters the way parallelism in L-systems is understood and implemented. Strictly speaking, the definition of L-systems requires productions to be applied *simultaneously* in each derivation step. In this context, allowing productions to modify and write global variables leads to inconsistencies when the productions attempt to set the same variable to different values. The situation changes if productions are viewed as *asynchronous concurrent processes* [33, Chapter 4]. Various synchronization techniques are then available to resolve the write conflict by allowing only one process at a time to modify a shared variable. For example, its value may be read, changed, and written back inside a *critical section* in each production [33, page 82]. Of course, the mutual exclusion of critical sections implies that productions are no longer applied simultaneously within a derivation step. Nevertheless, even with no imposed order of production application, the result of a derivation may be deterministic, as is the case when a shared variable is used to count the number of times a particular production has been applied.

6.3.4 Mini-language processing cycle

Since global variables are used to collect information about the overall generation process, it is necessary to both initialize and print the variables at the appropriate stages of a derivation. To be able to perform these operations, the notion of derivation in L-systems must be modified; it can no longer simply be a sequence of parallel production applications (Figure 6.3a), but must become a cycle of alternating sequential and parallel computations (Figure 6.3b). Further analysis of the structure of L-system execution revealed the following useful points for inserting statement lists while retaining as much of the parallel nature of the process as possible.

1. before the L-system derivation begins, in order to initialize global variables representing the environment and variables used for counting items related to

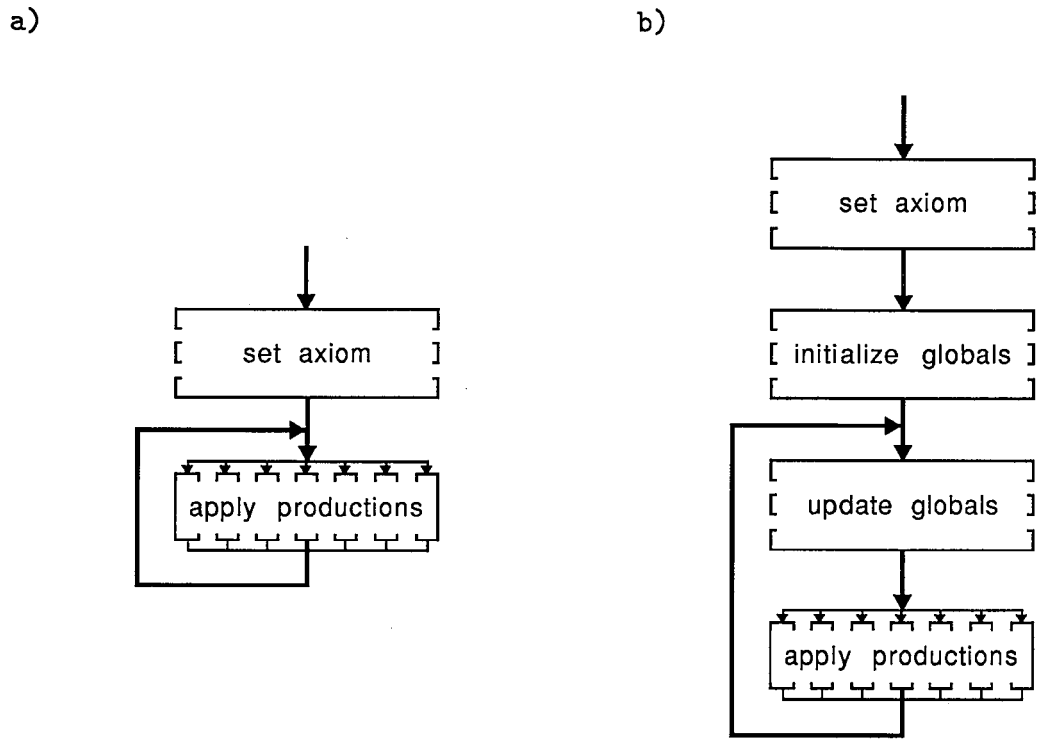


Figure 6.3: Operation of an L-system without global variables (a) and with global variables (b)

the number of derivation steps,

2. before each derivation step, in order to initialize variables used within a step and to modify environment variables over time,

3. at the time of production application, as described in Section 6.3.1,

4. at the end of each derivation step, to record current step results in global variables, and

5. at the end of the entire derivation process, to calculate and report overall statistics.

In the following example, global variables are used to compute the total length of the segments in the tree, the number of new branches in each step, and the total number of branches in the complete structure.

```
L-system 25:  Producing statistics for Honda's model for trees
#define r1 0.9                  /* contraction ratio for the trunk */
#define r2 0.6                  /* contraction ratio for branches */
#define a0 45                   /* branching angle from the trunk */
#define a2 45                   /* branching angle for lateral axes */
#define d  137.5                /* divergence angle */
#define wr 0.707                /* width decrease rate */
{W = 10;N = 1;L = 0;TB = 0;}
ω : A(1)
{W = W * wr;B = 0;}
p1 : A(s) : {L = L + s;B = B + 1;} → !(W)F(s)[&(a0)B(s * r2)]/(d)A(s * r1)
p2 : B(s) : {L = L + s;B = B + 1;} → !(W)F(s)[−(a2)@VC(s * r2)]C(s * r1)
p3 : C(s) : {L = L + s;B = B + 1;} → !(W)F(s)[+(a2)@VB(s * r2)]B(s * r1)
{
 print("Step=",N);
 N = N + 1;
 print("Current length=",L);
 print("New branches=",B);
 TB = TB + B;
}
{
 print("Total length=",L);
 print("Total branches=",TB);
}
```

The block of statements appearing immediately after the productions is processed at the end of each step. The statements in the final block are processed once at the end of the entire derivation. In this L-system, the branch width W is controlled globally as in L-system 24. The variables accumulating total segment length L and total number of branches TB are initialized to 0 at the start of the derivation. The new branch counter B is initialized to 0 at the start of each step. The statements

in the productions accumulate the number of branches and total length. At the end of each step, the appropriate output is produced and the step counter N and total branch counter TB are updated. After the derivation is complete, the total length and branch count are printed out.

In general, output procedures called at the end of each step or at the end of the derivation produce well-defined results. If they are called within productions, they do not; this is due to the parallel nature of L-systems. Since the productions are applied concurrently, the sequence of output messages is not predictable. However, most computers currently in use have a traditional single-processor architecture, requiring a sequential implementation of L-systems. The usual technique for performing the derivation step $\mu_i \Longrightarrow \mu_{i+1}$ is to scan the string μ_i from left to right, apply productions sequentially to consecutive modules, and append the resulting successors to the string μ_{i+1} [119, pages 107-108]. Since all modules of μ_i are considered before the next derivation step, the result is the same as if productions were applied in parallel. Models can take advantage of the sequential application of productions within each derivation step, and use constructs that rely on it.

6.3.5 Summary

These extensions enhance parametric L-systems as a plant modelling mini-language. The use of local and global variables can result in more legible L-systems that operate with greater efficiency in terms of both space requirements and processing time. In addition, the user has a wider range of output options; besides the graphical visualization, statistics on the operation of the model may be produced. This is particularly important for biological applications where the spatial model is used as a tool to study some other phenomena, such as the production of biomass.

While providing more modelling power, these extensions must be used with care, to ensure that the models produced do not violate the parallel nature of the growth process under consideration. For instance, the updating of a global variable in a production rule and the use of the same variable in another production's condition can result in production application being dependent on the order in which the modules are considered. The analysis of appropriate restrictions for the use of global variables,

and of the possibility of automatic detection of these situations, is open for further study.

The introduction of further programming languages features such as procedural statements and user defined functions is an area open for further development. A study of the biological implications of the use of such constructs and their relation to parallelism would also be of interest.

Chapter Seven

CONCLUSIONS

7.1 Research contributions

L-systems were proposed in 1968 as a mathematical model of development at the cellular level. The formalism applied parallel rewriting rules to simulate divisions or state changes of the cells constituting an organism. Geometric interpretation of L-systems was introduced to allow the visualization of the models, and the range of application was extended to include more complex plants. Nevertheless, the discrete nature of the rewriting formalism imposed limitations on the models. For example, exact values of branching angles, lengths of internodes, and concentrations of chemicals in plant modules could not be expressed easily.

In this dissertation, I have presented an extension of L-systems that overcomes these limitations. The key concept is the association of numerical parameters with the symbols representing plant modules, so that their features can be easily quantified. Parametric L-system productions incorporate arithmetic expressions for updating parameter values during the rewriting process. Parameters may also appear in the logical expressions used to select the applicable productions in each step. The formal definitions of various classes of parametric L-systems parallel the usual definitions found in the theory of L-systems, progressing from context free to context sensitive. Deterministic, non-deterministic, and stochastic application of productions is possible in each case. Finally, the notion of bracketed strings is incorporated to allow the modelling of branching structures.

The practical value of parametric L-systems has been demonstrated using a series of examples. The simple model of the blue-green alga *Anabaena catenula* illustrates the use of parameters to capture cell states and geometric attributes. In the model of spruce cones, parameters are used to express constant values of angles and lengths with an accuracy within a fraction of a percent; such precision is necessary for the model to operate correctly. In the model of the sunflower, as well as in the models of trees, precise parameter values are also important, but they are no longer constant and change gradually according to growth functions captured by mathematical expressions in the productions. In the model of compound leaves, parameters control developmental delays, which have a critical impact on the resulting structures. The models of simple leaves and the bicubic model of a petal surface capitalize on the ease with which parameters can be modified. The models are controlled by intuitive parameters that allow the creation of a family of plant organ shapes, and produce sequences of surface shapes representing the leaf or petal in consecutive stages of development. In order to create smooth animations, simulating time-lapse photography of developing plants, the concept of age-controlled models is introduced and illustrated using the vegetative growth of *Anabaena* as an example.

The examples reviewed to this point were all expressed using context-free parametric L-systems, as they only require lineage mechanisms to control development. Context-sensitive parametric L-systems make it possible to capture endogenous control mechanisms as well. In the model of *Anabaena* with heterocysts, one of the parameters represents concentration of nitrogen compounds; their diffusion through the filament and decay in the vegetative cells is described by mathematical expressions incorporated in the productions. In the models of *Mycelis muralis*, context-sensitive productions are used to simulate the flow of the hormones controlling flowering sequence.

Compared to modelling with non-parametric L-systems, the use of parameters can simplify production rules, making a model such as that of *Mycelis muralis* easier to present and understand. Furthermore, parameters facilitate interactive manipulation of model attributes using graphical tools such as control panels, since it is easier

to change values of parameters than to change the format of productions. Consequently, models expressed in terms of parametric L-systems are more accessible to users without a detailed background in computer science.

Experimentation with parametric L-systems revealed room for further improvements, resulting in further extensions to the formalism. Sub-L-systems offer the possibility of hierarchically incorporating previously defined components into larger structures, for example petals into flowers, flowers into inflorescences, and inflorescences into an entire plant. Geometric scales of components can be easily harmonized within a complex model. Specification of qualitative changes, such as the transformation from bud to flower to fruit, is facilitated by the cut symbol which removes a large number of unwanted symbols representing obsolete structures. Extensions incorporating features of other programming languages, such as local and global variables, improve the operation of parametric L-systems as a plant modelling mini-language. Local variables increase the efficiency of production application by preventing the repetitive calculation of the same expression within a production. Global variables may capture aspects of the environment that affect development, or collect information about a model such as the number of flowers produced in a certain amount of time.

7.2 Impact of parametric L-systems

Although the first description of parametric L-systems only appeared in 1990 [120], this notion has already had an impact beyond the scope of this dissertation, resulting in several software and research projects. In the book *The Algorithmic Beauty of Plants* [122], the parametric L-system formalism was used to express a wide variety of models, from abstract fractal branching structures to realistic models of herbaceous plants and trees. Inspired by this book, J. Leech of the University of North Carolina reproduced many of its results using a plant modelling program called *lsys* that he based on the parametric L-system concept. This software is available over Internet and is widely distributed.

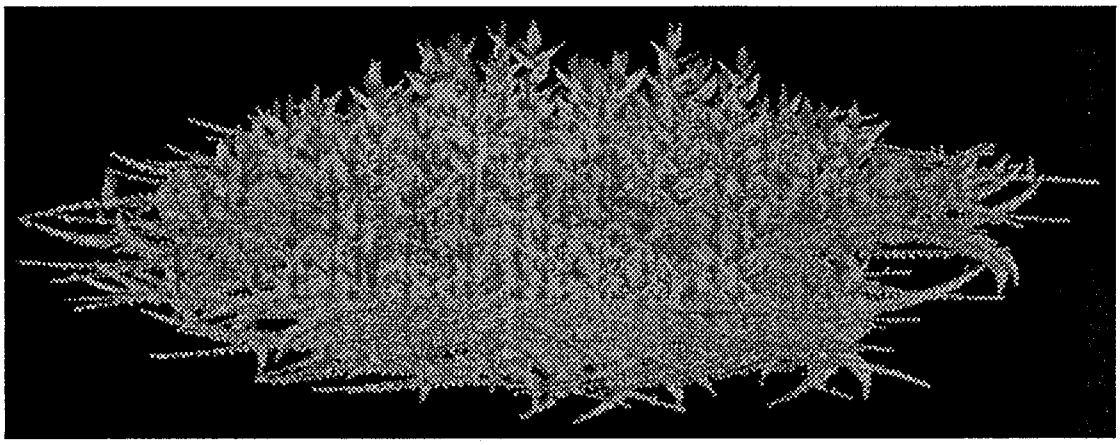


Figure 7.1: 16 day old *Physcomitrella patens* ©1991 F. D. Fracchia and N. W. Ashton

In the biological domain, N. Ashton and F. D. Fracchia [5] used parametric L-systems to model the development of a moss, *Physcomitrella patens*, from a single cell to a complete gametophyte (Figure 7.1). L-system rules were based on observations of the growth of a real moss, and the resulting visualization of development "provided an excellent means for verifying the underlying hypothesis of *P. patens* morphogenesis" [5].

For educational purposes, Dr. Prusinkiewicz, L. Mercer, and I designed and implemented an interactive display, *How Does Your Garden Grow?*. This project was commissioned by the Saskatchewan Science Centre to present basic concepts of plant structure and development to a school age audience (Figure 7.2). Pushbuttons are used to select from a variety of parametric L-system models in the garden. Parameters can then be manipulated using a dial, and 3D rotations controlled using a joystick.

The *Virtual Laboratory*, designed and implemented by L. Mercer [103, 104], combines a computer micro-world with a hypertext system in order to create a research and learning environment. This provides a means for organizing and performing simulation experiments in various domains. The main application of the system, which actually motivated the project, is a Virtual Laboratory in biology, centered on plant modelling software based on parametric L-systems. A sample screen is shown in Figure 7.3. Funding for this project was provided by Apple Computer, Inc., of Cupertino, California.

Figure 7.2: How Does Your Garden Grow?

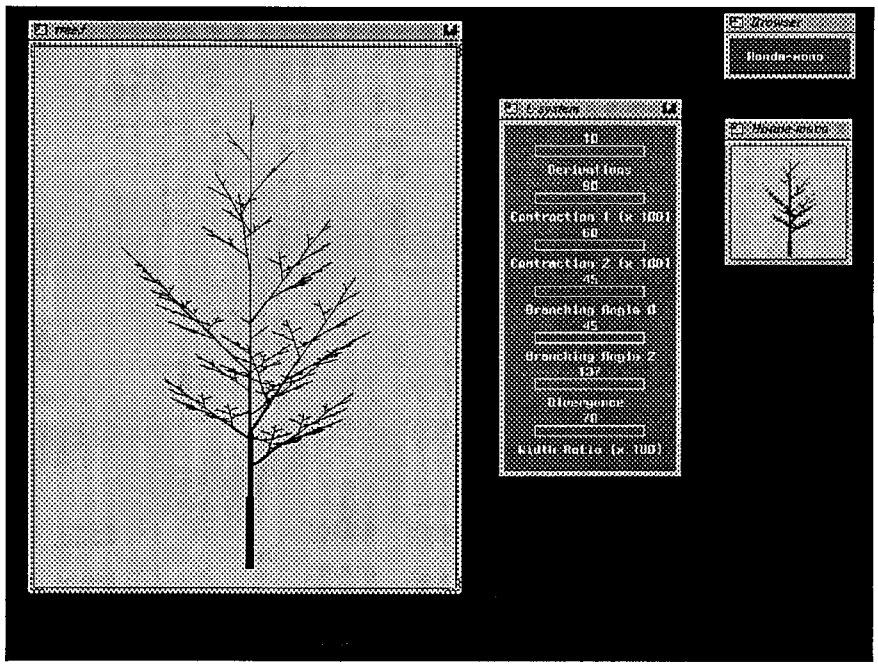


Figure 7.3: A virtual laboratory screen

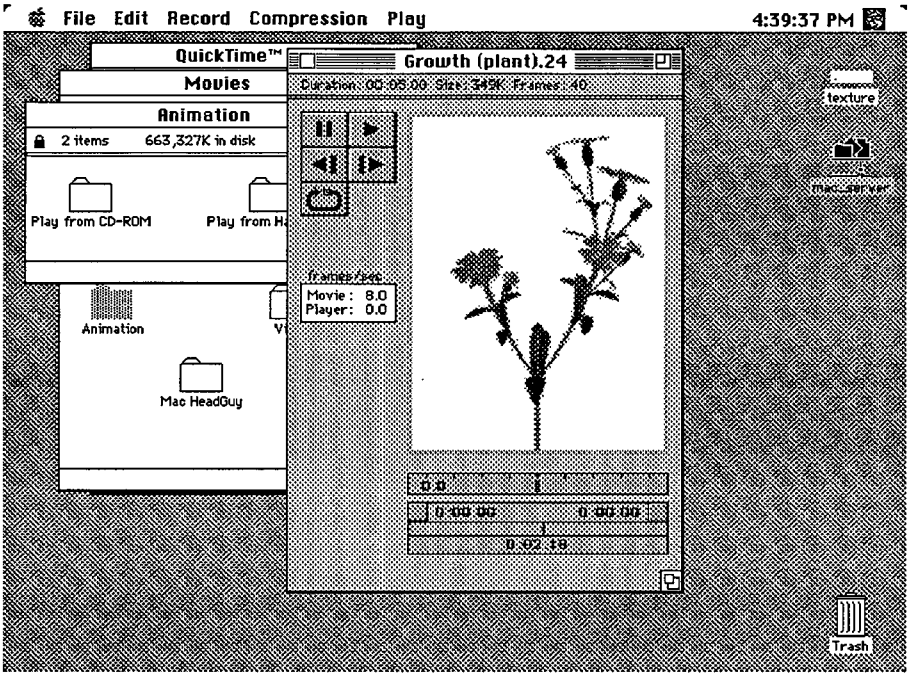


Figure 7.4: A frame from a QuickTime™ animation of rose campion growth

Apple's interest in the research presented in this dissertation also resulted in direct cooperation. Apple included animations of plant development produced with parametric L-systems in the initial release of their QuickTime™ software (Figure 7.4). QuickTime is a standard for multimedia processing that allows the recording and playback of real-time video and computer animation on a personal computer.

Further to this, I contributed several "exhibits" for the Virtual Museum designed by G. Miller *et al* [106] of Apple's Advanced Technology Group. The Virtual Museum is a software system that allows a "visitor" to "walk around" in a computer model of a museum. Exhibits in the museum's plant room are represented as three-dimensional flowers and leaves modelled using parametric L-systems. The individual exhibits consist of QuickTime movie animations of simulated plant growth and of the effects of changing model parameters, accompanied by text describing the models. A sample screen is shown in Figure 7.5.

In another collaboration, I incorporated parametric L-systems into MacBounce [105],

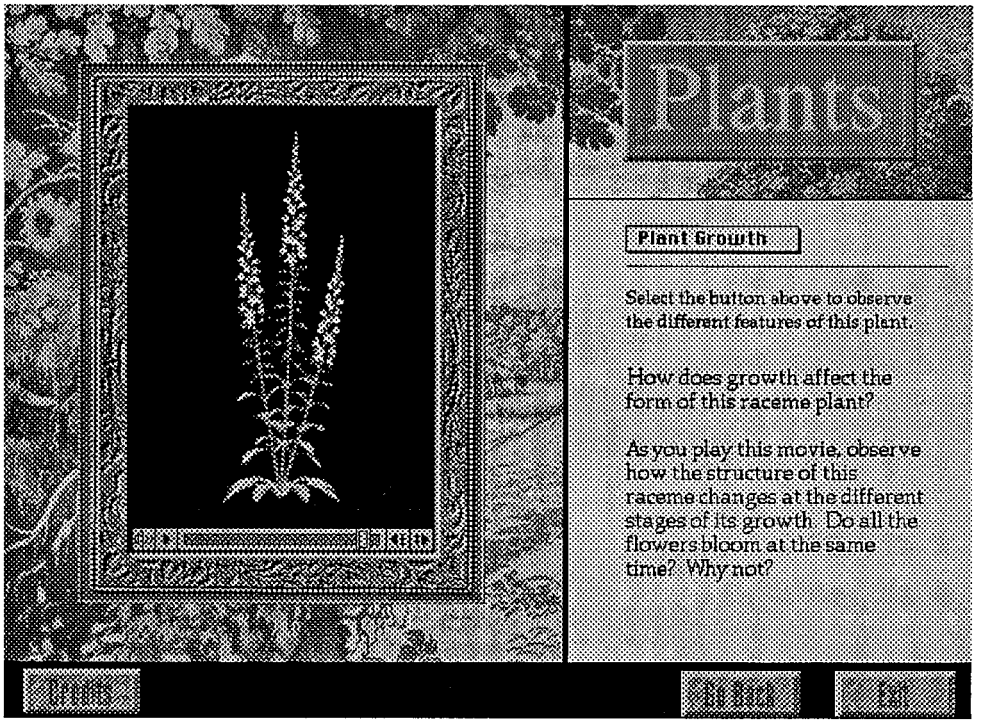


Figure 7.5: A Virtual Museum plant room exhibit ©1992 Apple Computer, Inc.

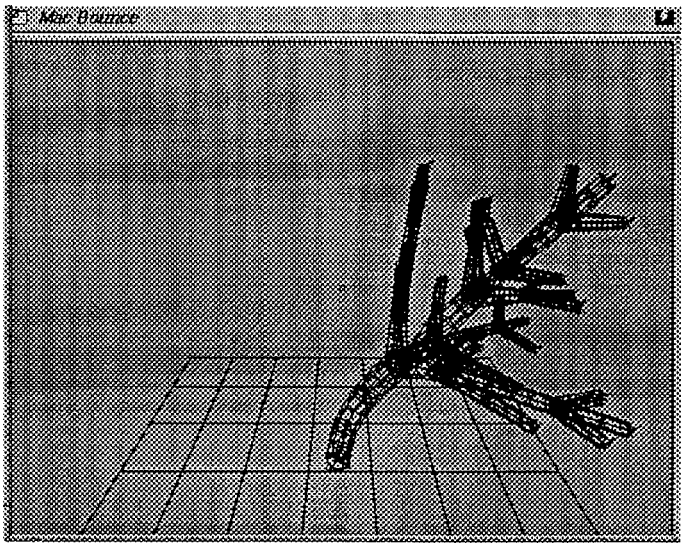


Figure 7.6: A simple branching structure under gravity

a physically-based modelling environment used in Apple's computer graphics research program. The resulting plants are susceptible to the effects of simulated gravity and wind, and to inter-object collisions. Rather than drawing line segments, the turtle inserts cross-like structures, composed of rigid bodies, at nodes. The nodes are connected by springs that are braced by torques. Parameters are used to specify the associated masses and constants. In each derivation step, the turtle's state at an existing node is updated by the physically-based model before productions are applied. A sample model is shown in Figure 7.6.

Plant models produced using parametric L-systems were also used in a physically-based modelling system developed by A. Snider [139]. His program operates on the final structure produced by the plant modelling program and applies physically-based techniques to enhance the resulting image, as shown in Figure 7.7. The disadvantage of this approach is that the structure cannot be grown.

Rather than incorporating parametric L-system models into a larger simulation

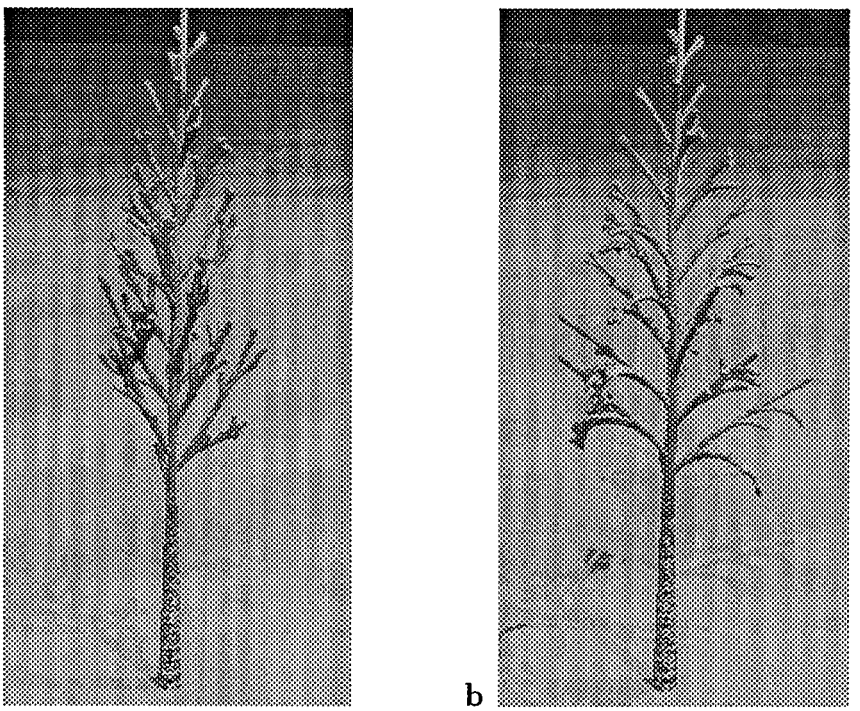


Figure 7.7: Parametric L-system trees with (a) and without (b) physically-based enhancements ©1992 A. Snider

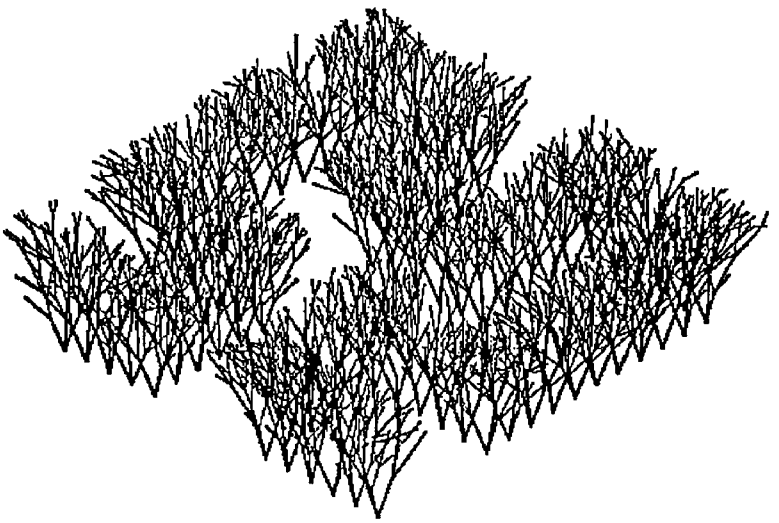


Figure 7.8: The Hilbert Hedge from [124]

environment, Prusinkiewicz and McFadzean [124] took the opposite approach, incorporating branch intersection testing into the modelling program based on parametric L-systems. This was used to simulate the effects of mechanical collisions between growing apices and objects in the environment including other branches. Figure 7.8 demonstrates the use of this technique to simulate the effects of pruning on a hedge. When a branch is clipped, a signal triggers the growth of dormant apices closer to the base of the branch.

7.3 Further research

The introduction of environmental effects presented at the end of the last section represents the initial stage of research aimed at complementing parametric L-systems with exogenous control mechanisms. The formalism presented in this dissertation allows limited simulation of these mechanisms. Global variables can be used to represent the environment, and sub-L-systems can simulate changes in the rules governing growth as a result of changes in the environment. The formalism should be examined to determine what can and cannot be modelled with it.

Other theoretical problems include the characterization of parametric L-systems

with regards to the traditional hierarchy of L-systems, and in relation to the restrictions that should be applied in the context of biological modelling. For instance, the modelling power of L-systems that employ subapical development to simulate branching structures should be investigated and compared to that of L-systems without such restrictions. Problems related to finding an L-system that models a given plant structure, called inference problems, are similar to those for standard L-systems. A D0L-system can be inferred automatically if a plant's development can be described by strictly lineage mechanisms; for interactive mechanisms the problem is harder [93]. However, a thorough analysis should be made to determine if parametric L-systems make inference problems easier to solve due to the presence of parameters.

Theoretical issues of a more general nature deserve attention as well. As illustrated by the L-systems that solved standard computational problems such as sorting and finding prime numbers, parametric L-systems can provide a model for parallel computation. The design and analysis of algorithms for this model is an open problem. The concept is also worth investigation as a basis for a class of programming languages based on monadic computation and concurrency.

Further extensions to the parametric L-system formalism can be made. The method for creating continuous animations should be extended to apply to context-sensitive L-systems. In addition, techniques for obtaining context information from an arbitrary number of branches should be worked out.

Hierarchical modelling using sub-L-systems has proven to be valuable in practice. However, this formalism requires further study to allow the harmonization of component time scales, particularly in the context-sensitive case. Examination of this problem may provide valuable insight into the use of parametric L-systems for modelling the asynchronous development of plant modules.

Another major area for further research is the use of the formalism for modelling a broader spectrum of plants. The modelling process itself can provide insight for biologists, as noted in the description of the work of Ashton *et al* [5] on page 132. The development of new realistic models is also important for future application in areas such as landscape design and commercial synthetic image production.

The creation of developmental models for both leaves and petals with topology

that changes over time remains a challenge. One approach to this problem would be to investigate the incorporation of the hierarchical B-spline refinement techniques proposed by Forsey and Bartels [45] into the developmental bicubic surface specification presented in this dissertation. An alternative approach, pursued by Hammel *et al* [60], uses parametric L-systems to build a skeleton around which an implicit surface contour is traced.

As a plant modelling mini-language, parametric L-systems can be employed to create developmental architectural models controlled by both lineage and endogenous mechanisms. The extension of parametric L-systems to model a broader range of environmental effects is a logical next step for further research. While some initial work has been done in this area, a variety of effects remain to be explored, including self-shadowing and plant-insect interactions. The latter are the focus of attention at the CSIRO Centre for Tropical Pest Management in Brisbane, Australia. The software developed in the scope of this dissertation has been chosen as the basis of their future modelling system. I am planning to participate in this project over the next few years and to continue my research, applying computer graphics as a tool to study the world around me.

Bibliography

[1] H. Abelson and A. A. diSessa. *Turtle geometry.* M.I.T. Press, Cambridge, 1982.

[2] S. G. Akl. *The design and analysis of parallel algorithms.* Prentice Hall, Englewood Cliffs, 1989.

[3] M. Aono and T. L. Kunii. Botanical tree image generation. *IEEE Computer Graphics and Applications*, 4(5):10–34, 1984.

[4] J. Arvo and D. Kirk. Modeling plants with environment-sensitive automata. In *Proceedings of Ausgraph '88*, pages 27–33, 1988.

[5] N. W. Ashton, F. D. Fracchia, and P. Prusinkiewicz. Computer modelling with L-systems as a tool for studying morphogenesis of the *Physcomitrella patens* gametophyte. In *Abstr. P/2. Mosses '91 International Symposium on the Physiology, Developmental Biology, Cell Biology and Genetics of Bryophytes. Heidelberg, April 11-13, 1991*, 1991.

[6] R. Baker and G. T. Herman. Simulation of organisms using a developmental model, parts I and II. *Int. J. of Bio-Medical Computing*, 3:201–215 and 251–267, 1972.

[7] R. W. Baker and G. T. Herman. CELIA - A cellular linear iterative array simulator. In *Proceedings of the Fourth Conference on Applications of Simulation*, pages 64–73, 1970.

[8] J. Banks and J. S. Carson, II. *Discrete-event system simulation.* Prentice-Hall international series in industrial and systems engineering. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[9] M. F. Barnsley. *Fractals everywhere.* Academic Press, San Diego, 1988.

[10] R. Bartels, J. Beatty, and B. Barsky, editors. *An introduction to splines for use in computer graphics and geometric modeling.* Morgan Kaufman, Los Altos, California, 1987.

[11] A. D. Bell. Computerized vegetative mobility in rhizomatous plants. In A. Lindenmayer and G. Rozenberg, editors, *Automata, languages, development*, pages 3–14. North-Holland, Amsterdam, 1976.

[12] A. D. Bell. On the astogeny of six-cornered clones: An aspect of modular construction. In J. White, editor, *Studies on plant demography: A festschrift for John L. Harper*, pages 187–207. Academic Press, London, 1985.

[13] A. D. Bell. The simulation of branching patterns in modular organisms. *Phil. Trans. Royal Society London, Ser. B*, 313:143–159, 1986.

[14] A. D. Bell, D. Roberts, and A. Smith. Branching patterns: the simulation of plant architecture. *Journal of Theoretical Biology*, 81:351–375, 1979.

[15] D. S. Berger. Modification of a simple fractal tree growth scheme: Implications on growth, variation, and evolution. *Journal of Theoretical Biology*, 152:513–529, 1991.

[16] J. Bloomenthal. Modeling the mighty maple. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985) in *Computer Graphics*, 19, 3 (July 1985), pages 305–311, ACM SIGGRAPH, New York, 1985.

[17] R. Borchert and H. Honda. Control of development in the bifurcating branch system of *Tabebuia rosea*: A computer simulation. *Botanical Gazette*, 145(2):184–195, 1984.

[18] R. Borchert and P. B. Tomlinson. Architecture and crown geometry in *Tabebuia rosea* (Bignoniaceae). *American Journal of Botany*, 71:958–969, 1984.

[19] C. C. Borel, A. W. S. Gerstl, and B. J. Powers. The radiosity method in optical remote sensing of structured 3D surfaces. *Remote Sensing of Environment*, 36:13–44, 1991.

[20] H. Bothe, H. Nelles, T. Kentemick, H. Papen, and G. Neuer. Recent aspects of heterocyst biochemistry and differentiation. In W. Wiessner, D. Robinson, and R. C. Starr, editors, *Compartments in Algal Cells and Their Interaction*, pages 218–232. Springer-Verlag, Berlin, 1984.

[21] C. Briere and R. Buis. Activation-inhibition model for branching of a filamentous organism. *Mathematical Biosciences*, 74:219–232, 1985.

[22] T. W. Chien. Graphical interpretation of the biological development modelled by VD0L-systems. Master's thesis, University of Western Ontario, 1989.

[23] T. W. Chien and H. Jürgensen. Parameterized L systems for modelling: Potential and limitations. Report 283, Department of Computer Science, University of Western Ontario, 1992.

[24] L. A. Cochrane and E. D. Ford. Growth of a Sitka spruce plantation: Analysis and stochastic description of the development of the branching structure. *Journal of Applied Ecology*, 15:227–244, 1978.

[25] D. Cohen. Computer simulation of biological pattern generation processes. *Nature*, 216:246–248, 1967.

[26] J. D. Corbit and D. J. Garbary. Computer simulation of the morphology and development of several species of seaweed using Lindenmayer systems. *Computers and Graphics*, in press, 1992.

[27] P. Dabadie, P. de Reffye, and P. Dinouard. Modélisation de la croissance et de l'architecture d'un bambou. In *Deuxième congrès international du bambou*, 1988.

[28] C. G. de Koster and A. Lindenmayer. Discrete and continuous models for heterocyst differentiation in growing filaments of blue-green bacteria. *Acta Biotheoretica*, 36:249–273, 1987.

[29] P. de Reffye. Modéle mathématique aléatorie et simulation de la croissance et de l'architecture du caféier robusta. Première partie, *Café-Cacao-Thé*, 25(2):83–104, 1981. Deuxième partie, *Café-Cacao-Thé*, 25(4):219–230, 1981. Troisième partie, *Café-Cacao-Thé*, 26(2):77–96, 1982. Quatrième partie, *Café-Cacao-Thé*, 27(1):3–20, 1983.

[30] P. de Reffye. *Modélisation de l'architecture des arbres tropicaux par des processus stochastiques.* PhD thesis, Université de Paris-Sud, Orsay, France, 1979.

[31] P. de Reffye, C. Edilin, J. Françon, M. Jaeger, and C. Puech. Plant models faithful to botanical structure and development. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988), in *Computer Graphics* 22,4 (August 1988), pages 151–158, ACM SIGGRAPH, New York, 1988.

[32] S. Demko, L. Hodges, and B. Naylor. Construction of fractal objects with iterated function systems. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985) in *Computer Graphics*, 19, 3 (July 1985), pages 271–278, ACM SIGGRAPH, New York, 1985.

[33] H. M. Dietel. *An introduction to operating systems.* Addison-Wesley, Reading, Massachusetts, 1983.

[34] C. Edelin. *L'architecture monopodiale; l'exemple de quelques arbres d'Asie tropicaux.* PhD thesis, Université des sciences et des techniques du Languedoc, Montpellier, France, 1984.

[35] M. Eden. A two-dimensional growth process. In J. Neeyman, editor, *Proceedings of Fourth Berkeley Symposium on Mathematics, Statistics, and Probability*, volume 4, pages 223–239, Berkeley, 1960. University of California Press.

[36] P. Eichhorst and W. J. Savitch. Growth functions of stochastic Lindenmayer systems. *Information and Control*, 45:217–228, 1980.

[37] R. O. Erickson. The geometry of phyllotaxis. In J. E. Dale and F. L. Milthrope, editors, *The growth and functioning of leaves*, pages 53–88. University Press, Cambridge, 1983.

[38] G. Eyrolles. *Synthèse d'images figuratives d'arbres par des méthodes combinatoires.* PhD thesis, Université de Bordeaux I, 1986.

[39] J. B. Fisher. Branching patterns and angles in trees. In T. J. Givnish, editor, *On the economy of plant form and function*, pages 493–523. Cambridge University Press, Cambridge, 1986. Proceedings of the Sixth Maria Moors Cabot Symposium, Harvard Forest, August, 1983.

[40] J. B. Fisher and H. Honda. Computer simulation of branching pattern and geometry in *Terminalia* (Combretaceae), a tropical tree. *Botanical Gazette*, 138(4):377–384, 1977.

[41] J. B. Fisher and H. Honda. Branch geometry and effective leaf area: A study of *Terminalia*-branching pattern, Parts I and II. *American Journal of Botany*, 66:633–655, 1979.

[42] J. D. Foley and A. Van Dam. *Fundamentals of interactive computer graphics.* Addison-Wesley, Reading, Massachusetts, 1982.

[43] E. D. Ford, A. Avery, and R. Ford. Simulation of branch growth in the *Pinaceae*: Interactions of morphology, phenology, foliage productivity, and the requirement for structural support, on the export of carbon. *Journal of Theoretical Biology*, 146:15–36, 1990.

[44] R. Ford and E. D. Ford. Structure and basic equations of a simulator for branch growth in the *Pinaceae*. *Journal of Theoretical Biology*, 146:1–13, 1990.

[45] D. R. Forsey and R. H. Bartels. Hierarchical B-spline refinement. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988) in *Computer Graphics*, 22, 4 (August 1988), pages 205–212, ACM SIGGRAPH, New York, 1988.

[46] A. Fournier. Prolegomenon. SIGGRAPH '87 Course Notes on the Modeling of Natural Phenomena, 1987.

[47] A. Fournier and D. A. Grindal. The stochastic modelling of trees. In *Proceedings of Graphics Interface '86 — Vision Interface '86*, pages 164–172, 1986.

[48] D. R. Fowler, J. Hanan, and P. Prusinkiewicz. Modelling spiral phyllotaxis. *computers & graphics*, 13(3):291–296, 1989.

[49] D. Frijters. Simulation by L-systems of branching patterns with copy relationships. In *Abstracts of the Conference on Formal Languages, Automata and Development*, Noordwijkerhout, The Netherlands, 1975.

[50] D. Frijters. Mechanisms of developmental integration of *Aster novae-angliae* L. and *Hieracium murorum* L. *Annals of Botany*, 42:561–575, 1978.

[51] D. Frijters. Principles of simulation of inflorescence development. *Annals of Botany*, 42:549–560, 1978.

[52] D. Frijters and A. Lindenmayer. A model for the growth and flowering of *Aster novae-angliae* on the basis of table (1,0)L-systems. In G. Rozenberg and A. Salomaa, editors, *L Systems*, Lecture Notes in Computer Science 15, pages 24–52. Springer-Verlag, Berlin, 1974.

[53] D. Frijters and A. Lindenmayer. Developmental descriptions of branching patterns with paracladial relationships. In A. Lindenmayer and G. Rozenberg, editors, *Automata, languages, development*, pages 57–73. North-Holland, Amsterdam, 1976.

[54] G. Y. Gardner. Simulation of natural scenes using textured quadric surfaces. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 22-27, 1984) in *Computer Graphics*, 18, 3 (July 1984), pages 11–20, ACM SIGGRAPH, New York, 1984.

[55] J. Gips. *Shape grammars and their uses; artificial perception, shape generation and computer aesthetics.* Birkhäuser-Verlag, Basel and Stuttgart, 1975.

[56] N. S. Goel, I. Rozehnal, and R. L. Thompson. A computer graphics based model for scattering from objects of arbitrary shape in the object region. *Remote Sensing of Environment*, 36(2):73–140, 1991.

[57] M. E. Gottlieb. The VT model: A deterministic model of angiogenesis and biofractals based on physiological rules. In *Proceedings of IEEE 17th Annual Northeast Bioengineering Conference*, pages 38–39, 1991.

[58] N. Greene. Voxel space automata: Modeling with stochastic growth processes in voxel space. Proceedings of SIGGRAPH '89 (Boston, Mass., July 31-August 4, 1989), in *Computer Graphics* 23,4 (August 1989), pages 175–184, ACM SIGGRAPH, New York, 1989.

[59] F. Hallé, R. A. A. Oldeman, and P. B. Tomlinson. *Tropical trees and forests: An architectural analysis.* Springer-Verlag, Berlin, 1978.

[60] M. S. Hammel, P. Prusinkiewicz, and B. Wyvill. Modelling compound leaves using implicit contours. In *Proceedings of CG International '92*, 1992. To appear.

[61] J. S. Hanan. PLANTWORKS: A software system for realistic plant modelling. Master's thesis, University of Regina, 1988.

[62] J. S. Hanan and P. Prusinkiewicz. From L-systems to a plant modelling language. In *Proceedings of the Third Annual Western Computer Graphics Symposium*, pages 16–20, 1991.

[63] J. L. Harper. The concept of population in modular organisms. In R. M. May, editor, *Theoretical Ecology: Principles and Applications, Second Edition*, chapter 4, pages 53–77. Blackwell Scientific, Oxford, 1981.

[64] J. L. Harper and A. D. Bell. The population dynamics of growth form in organisms with modular construction. In R. M. Anderson, B. D. Turner, and L. R. Taylor, editors, *Population Dynamics*, pages 29–52. Blackwell Scientific, Oxford, 1979. The 20th Symposium of the British Ecological Society, London, 1978.

[65] R. Henderson, E. D. Ford, and E. Renshaw. Morphology of the structural root system of Sitka spruce. 2. Computer simulation of rooting patterns. *Forestry*, 56(2):137–153, 1983.

[66] R. Henderson, E. D. Ford, E. Renshaw, and J. D. Deans. Morphology of the structural root system of Sitka spruce. 1. Analysis and quantitative description. *Forestry*, 56(2):121–135, 1983.

[67] R. Henderson and E. Renshaw. Spatial stochastic models and computer simulation applied to the study of tree root systems. In *COMPSTAT 1980*, pages 389–395, 1980.

[68] G. Herman, A. Lindenmayer, and G. Rozenberg. Description of developmental languages using recurrence systems. *Mathematical Systems Theory*, 8:316–341, 1975.

[69] G. T. Herman and W. H. Liu. The daughter of CELIA, the French flag, and the firing squad. *Simulation*, 21:33–41, 1973.

[70] G. T. Herman and G. Rozenberg. *Developmental systems and languages.* North-Holland, Amsterdam, 1975.

[71] G. T. Herman and G. L. Schiff. Simulation of multi-gradient models of organisms in the context of L-systems. *Journal of Theoretical Biology*, 54:35–46, 1975.

[72] P. Hogeweg. Simulating the growth of cellular forms. *Simulation*, 31(3):90–96, September 1978.

[73] P. Hogeweg and B. Hesper. A model study on biomorphological description. *Pattern Recognition*, 6:165–179, 1974.

[74] H. Honda. 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, 1971.

[75] H. Honda and J. B. Fisher. Tree branch angle: Maximizing effective leaf area. *Science*, 199:888–890, 1978.

[76] H. Honda and J. B. Fisher. Ratio of tree branch lengths: The equitable distribution of leaf clusters on branches. *Proceedings of the National Academy of Sciences USA*, 76(8):3875–3879, 1979.

[77] H. Honda, P. B. Tomlinson, and J. B. Fisher. Computer simulation of branch interaction and regulation by unequal flow rates in botanical trees. *American Journal of Botany*, 68(4):569–585, 1981.

[78] H. Honda, P. B. Tomlinson, and J. B. Fisher. Two geometrical models of branching of botanical trees. *Annals of Botany*, 49:1–11, 1982.

[79] R. Hunt. *Plant growth analysis.* Studies in Biology 96. Edward Arnold, London, 1978.

[80] R. Hunt. *Plant growth curves – the functional approach to plant growth analysis.* Edward Arnold, London, 1982.

[81] G. Van Iterson. *Mathematische und mikroskopish-anatomische Studien über Blattstellungen.* Gustav Fischer, Jena, 1907.

[82] M. Jaeger. *Représentation et simulation de croissance des végétaux.* PhD thesis, Université Louis Pasteur de Strasbourg, 1987.

[83] J. M. Janssen and A. Lindenmayer. Models for the control of branch positions and flowering sequences of capitula in *Mycelis muralis* (L.) Dumont (Compositae). *New Phytologist*, 105:191–220, 1987.

[84] J. A. Kaandorp. Interactive generation of fractal objects. In G. Maréchal, editor, *Proceedings of EUROGRAPHICS '87*, pages 181–197, 1987.

[85] J. A. Kaandorp. Modelling growth forms of sponges with fractal techniques. In A. J. Crilly, R. A. Earnshaw, and H. Jones, editors, *Fractals and Chaos*, pages 181–188. Springer Verlag, 1991.

[86] Y. Kawaguchi. A morphological study of the form of nature. Proceedings of SIGGRAPH '82 (Boston, Massachusetts, July 26-30, 1982), in *Computer Graphics* 16,3 (July 1982), pages 223–232, ACM SIGGRAPH, New York, 1982.

[87] M. W. Krueger. *Artificial Reality.* Addison-Wesley, Reading, Massachusetts, 1983.

[88] A. Lindenmayer. Mathematical models for cellular interactions in development, Parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[89] A. Lindenmayer. Developmental systems without cellular interactions, their languages and grammars. *Journal of Theoretical Biology*, 30:455–484, 1971.

[90] A. Lindenmayer. Adding continuous components to L-systems. In G. Rozenberg and A. Salomaa, editors, *L Systems*, Lecture Notes in Computer Science 15, pages 53–68. Springer-Verlag, Berlin, 1974.

[91] A. Lindenmayer. Developmental algorithms: Lineage versus interactive control mechanisms. In S. Subtelny and P. B. Green, editors, *Developmental order: Its origin and regulation*, pages 219–245. Alan R. Liss, New York, 1982.

[92] A. Lindenmayer. Positional and temporal control mechanisms in inflorescence development. In P. W. Barlow and D. J. Carr, editors, *Positional controls in plant development.* University Press, Cambridge, 1984.

[93] A. Lindenmayer and H. Jürgensen. Grammars of development: Discrete state models for growth, differentiation, and gene expression in modular organisms. Report 285, Department of Computer Science, University of Western Ontario, 1991.

[94] A. Lindenmayer and P. Prusinkiewicz. An annotated bibliography of plant modeling and growth simulation. In C. Langton, editor, *Artificial Life: Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems held September, 1987, in Los Alamos, New Mexico*, pages 625–643. Addison–Wesley, Redwood City, 1989.

[95] A. Lindenmayer and G. Rozenberg, editors. *Automata, languages, development.* North-Holland, Amsterdam, 1976.

[96] J. Lück, H. B. Lück, and M. Bakkali. A comprehensive model for acrotonic, mesotonic and basitonic branchings in plants. *Acta Biotheoretica*, 38:257–288, 1990.

[97] N. Macdonald. *Trees and networks in biological models.* J. Wiley & Sons, New York, 1983.

[98] B. B. Mandelbrot. *The fractal geometry of nature.* W. H. Freeman, San Francisco, 1982.

[99] R. Marshall, R. Wilson, and W. Carlson. Procedure models for generating three-dimensional terrain. Proceedings of SIGGRAPH '80 (Seattle, Washington, July 14-18, 1980) in *Computer Graphics*, 14, 3 (July 1980), pages 154–162, ACM SIGGRAPH, New York, 1980.

[100] J. J. McConnell. *Botanical image generation using attributed graph grammars for modeling growth.* PhD thesis, Worcester Polytechnic Institute, 1989.

[101] P. Meakin. A new model for biological pattern formation. *Journal of Theoretical Biology*, 118:101–113, 1986.

[102] H. Meinhardt. *Models of biological pattern formation.* Academic Press, New York, 1982.

[103] L. Mercer. The virtual laboratory. Master's thesis, University of Regina, 1991.

[104] L. Mercer, P. Prusinkiewicz, and J. Hanan. The concept and design of a virtual laboratory. In *Proceedings of Graphics Interface '90*, pages 149–155. CIPS, 1990.

[105] G. S. P. Miller. MacBounce: A dynamics-based modeller for character animation. In N. M. Thalmann and D. Thalmann, editors, *Proceedings of Computer Animation '91*, pages 149–168. Springer-Verlag, 1991.

[106] G. S. P. Miller, E. Hoffert, S. E. Chen, E. Patterson, D. Blackketter, S. Rubin, S. A. Applin, D. Yim, and J. Hanan. The Virtual Museum: Interactive 3D navigation of a multimedia database. *The Journal of Visualisation and Animation*, 1992. To appear.

[107] G. J. Mitchison and Michael Wilcox. Rules governing cell division in *Anabaena*. *Nature*, 239:110–111, 1972.

[108] R. A. Morelli, R. E. Walde, E. Akstin, and C. W. Schneider. L-system representation of speciation in the red algal genus *Dipterosiphonia* (Ceramiales, Rhodomelaceae). *Journal of Theoretical Biology*, 149:453–465, 1991.

[109] F. K. Musgrave. About the cover: Natura Ex Machina II. *IEEE Computer Graphics and Applications*, 10(6):5–7, 1990.

[110] K. J. Niklas. Computer simulations of early land plant branching morphologies: canalization of patterns during evolution? *Paleobiology*, 8(3):196–210, 1982.

[111] T. Nishida. KOL-systems simulating almost but not exactly the same development — the case of Japanese cypress. *Memoirs Fac. Sci., Kyoto University, Ser. Bio*, 8:97–122, 1980.

[112] P. Oppenheimer. Fractals, computers and DNA. In *Proceedings of Graphics Interface '86 — Vision Interface '86*, pages 254–259, 1986.

[113] P. Oppenheimer. Real time design and animation of fractal plants and trees. *Computer Graphics*, 20(4):55–64, 1986.

[114] S. Papert. *Mindstorms: Children, computers and powerful ideas*. Basic Books, New York, 1980.

[115] R. Y. Pinter and S. S. Pinter. Efficient processing of L-systems on the Connection Machine, or parallel breadth-first expansion. Manuscript, Computer Science Department, Yale University, 1989.

[116] P. Prusinkiewicz. Graphical applications of L-systems. In *Proceedings of Graphics Interface '86 — Vision Interface '86*, pages 247–253. CIPS, 1986.

[117] P. Prusinkiewicz. Applications of L-systems to computer imagery. In H. Ehrig, M. Nagl, A. Rosenfeld, and G. Rozenberg, editors, *Graph grammars and their application to computer science; Third International Workshop*, pages 534–548. Springer-Verlag, Berlin, 1987. Lecture Notes in Computer Science 291.

[118] P. Prusinkiewicz and M. S. Hammel. Continuous Animation Using L-systems, Video Tape, University of Calgary, 1991.

[119] P. Prusinkiewicz and J. Hanan. *Lindenmayer systems, fractals, and plants*, volume 79 of *Lecture Notes in Biomathematics*. Springer-Verlag, Berlin, 1989.

[120] P. Prusinkiewicz and J. Hanan. Visualization of botanical structures and processes using parametric L-systems. In D. Thalmann, editor, *Scientific Visualization and Graphics Simulation*, pages 183–201. J. Wiley & Sons, 1990.

[121] P. Prusinkiewicz and J. Hanan. L-systems: From formalism to programming languages. In G. Rozenberg and A. Salomaa, editors, *In the footsteps of L: L-wise studies to the memory of Aristid Lindenmayer 1925–1989*. Springer-Verlag, Berlin, 1992. In press, 19 pages.

[122] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990. With J. Hanan, D. Fracchia, D. Fowler, M. de Boer, and L. Mercer.

[123] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan. Developmental models of herbaceous plants for computer imagery purposes. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988), in *Computer Graphics* 22,4 (August 1988), pages 141–150, ACM SIGGRAPH, New York, 1988.

[124] P. Prusinkiewicz and D. McFadzean. Modelling plants in environmental context. In *Proceedings of the Fourth Annual Western Computer Graphics Symposium*, pages 47–51, 1992.

[125] W. T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985) in *Computer Graphics*, 19, 3 (July 1985), pages 313–322, ACM SIGGRAPH, New York, 1985.

[126] W. R. Remphrey, B. R. Neal, and T. A. Steeves. The morphology and growth of *Arctostaphylos uva-ursi* (bearberry): An architectural model simulating colonizing growth. *Canadian Journal of Botany*, 61(9):2451–2458, 1983.

[127] W. R. Remphrey and G. R. Powell. Crown architecture of *Larix laricina* saplings: Quantitative analysis and modelling of (nonsylleptic) order 1 branching in relation to development of the main stem. *Canadian Journal of Botany*, 62(9):1904–1915, 1984.

[128] W. R. Remphrey and G. R. Powell. Crown architecture of *Larix laricina* saplings: Sylleptic branching on the main stem. *Canadian Journal of Botany*, 63(7):1296–1302, 1985.

[129] W. R. Remphrey, T. A. Steeves, and B. R. Neal. The morphology and growth of *Arctostaphylos uva-ursi* (bearberry): An architectural analysis. *Canadian Journal of Botany*, 61(9):2430–2450, 1983.

[130] E. Renshaw. Computer simulation of Sitka spruce: Spatial branching models for canopy growth and root structure. *IMA Journal of Mathematics Applied in Medicine and Biology*, 2:183–200, 1985.

[131] P. M. Room. Plant architecture and how biological control agents affect the dynamics of weeds. In E. S. Delfosse, editor, *Proceedings of VI International Symposium on Biological Control of Weeds*, pages 89–102. Agriculture Canada, 1985.

[132] G. Rozenberg. T0L systems and languages. *Information and Control*, 23:357–381, 1973.

[133] G. Rozenberg and A. Salomaa. *The mathematical theory of L-systems.* Academic Press, New York, 1980.

[134] A. Salomaa. *Formal languages.* Academic Press, New York, 1973.

[135] M. Shebell. Modeling branching plants using attribute L-systems. Master's thesis, Worcester Polytechnic Institute, 1986.

[136] A. R. Smith. Plants, fractals, and formal languages. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 22-27, 1984) in *Computer Graphics*, 18, 3 (July 1984), pages 1–10, ACM SIGGRAPH, New York, 1984.

[137] A. R. Smith. About the cover: Reconfigurable machines. *Computer*, 11(7):3–4, 1978.

[138] B. H. Smith. The optimal design of a herbaceous body. *The American Naturalist*, 123(2):197–211, 1984.

[139] A. Snider. Manuscript. Master's thesis, University of Regina, 1992.

[140] G. Stiny. *Pictorial and formal aspects of shape and shape grammars.* Birkhäuser-Verlag, Basel and Stuttgart, 1975.

[141] A. L. Szilard. Growth functions of Lindenmayer systems. Technical Report 4, Computer Science Department, University of Western Ontario, 1971.

[142] A. L. Szilard and R. E. Quinton. An interpretation for DOL systems by computer graphics. *The Science Terrapin*, 4:8–13, 1979.

[143] d'Arcy Thompson. *On growth and form.* University Press, Cambridge, 1952.

[144] A. M. Turing. The chemical basis of morphogenesis. *Phil. Trans. Royal Society London, Ser. B*, 237:37–72, 1952.

[145] S. Ulam. On some mathematical problems connected with patterns of growth of figures. In *Proceedings of symposia in applied mathematics, 14*, pages 215–224. American Mathematical Society, 1962.

[146] S. Ulam. Patterns of growth of figures: Mathematical aspects. In G. Kepes, editor, *Module, Proportion, Symmetry, Rhythm*, pages 64–74. Braziller, New York, 1966.

[147] X. G. Viennot, G. Eyrolles, N. Janey, and D. Arquès. Combinatorial analysis of ramified patterns and computer imagery of trees. Proceedings of SIGGRAPH '89 (Boston, Mass., July 31-August 4, 1989), in *Computer Graphics* 23,4 (August 1989), pages 31–40, ACM SIGGRAPH, New York, 1989.

[148] P. M. B. Vitányi. Development, growth and time. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 431–444. Springer-Verlag, Berlin, 1986.

[149] H. Vogel. A better way to construct the sunflower head. *Mathematical Biosciences*, 44:179–189, 1979.

[150] J. von Neumann. *Theory of self-reproducing automata.* University of Illinois Press, Urbana, 1966. Edited by A. W. Burks.

[151] K. W. Waite. Modelling natural branching structures. *Computer Graphics Forum*, 7(2):105–115, 1988.

[152] D. M. Waller and D. A. Steingraeber. Branching and modular growth: Theoretical models and empirical patterns. In J. B. C. Jackson, L. W. Buss, and R. E. Cook, editors, *Population Biology and Evolution of Clonal Organisms*, pages 225–257. Yale University Press, 1985.

[153] S. A. Ward and R. H. Halstead. A syntactic theory of message passing. *Journal of the ACM*, 27(2):365–383, April 1980.

[154] T. A. Witten and L. M. Sander. Diffusion-limited aggregation. *Phys. Rev. B*, 27:5686–5697, 1983.

[155] D. Wood. Time-delayed OL languages and sequences. *Information Sciences*, 8:271–281, 1975.

[156] D. Wood. Generalized time-delayed OL languages. *Information Sciences*, 12:151–155, 1977.

[157] B. Wyvill. *An Interactive Graphics Language.* PhD thesis, University of Bradford, 1975.

[158] T. Yokomori. Stochastic characterizations of EOL languages. *Information and Control*, 45:26–33, 1980.

Appendix A

USER'S VIEW OF THE IMPLEMENTATION

The parametric L-system formalism has been implemented in a program called the *continuous plant and fractal generator* or cpfg. In keeping with the philosophy of the Virtual Laboratory [103, 104], the program was designed to produce a visualization of a parametric L-system specification read in from a file, and to support subsequent interaction by re-reading the file after changes have been made. These changes can be introduced using either a standard text editor or virtual control panels.

This appendix describes the cpfg program from a user's perspective. It includes a general overview and descriptions of the command line arguments, interactive features, and sample input files.

A.1 Overview

Cpfg produces a visualization of a parametric L-system model using Logo-style "turtle" geometric interpretation. The user interacts with the program by:

- specifying the file names for input data on the command line,
- rotating the modelled object interactively using a mouse-controlled cursor, and
- controlling execution of the program using a menu.

In the initial call to cpfg, the user specifies the names of up to three files (for samples see Section A.3).

- The *L-system file* contains the parametric L-system that describes the model. There is a main L-system, possibly followed by one or more sub-L-systems. Each L-system is composed of an identifier, an optional seed for the random number generator if the L-system is stochastic, a derivation length, a list of symbols to be ignored when context matching, the axiom, the set of productions, and statement lists to be interpreted during the derivation process.

- The *view file* contains viewing, rendering, and drawing parameters, including the names of files that specify any surfaces to be included in the model.

- The *animation file* contains parameters controlling frame by frame production of images for animation purposes. This file is optional.

In addition, the command line may contain output file names for rayshade or PostScript output (2D only), and a specification of the initial string length, which defaults to 150,000.

The cpfg program generates a string by performing the given number of derivation steps for the input L-system model. The window and viewing parameters are set according to the bounding box of the modelled object, either as specified in the view file by the user, or as calculated by the program, so that the image will be positioned in the center of the cpfg window. Finally, the string is interpreted as described in Section 4.5 to create an image on the screen.

A.2 User interaction

After the initial image is displayed, further interaction with the program is controlled using the mouse. In order to rotate the object, the left mouse button is pressed with the cursor anywhere in the cpfg window. While the left button is down, any movement of the cursor will rotate the object around axes through the center of its bounding volume. Left to right movements cause rotations around the vertical axis

parallel to the screen, and up and down movements cause rotations around the horizontal axis parallel to the screen. In order to select commands from the pop-up menus, the right mouse button is pressed with the cursor anywhere in the program's window. There are two menus available to the user. Initially, the user has access to the main menu, which includes the following items:

- *New model*, which rereads both the L-system and view files, generates a new string, resets the window and viewing parameters, and interprets the string to create a new image,
- *New L-system*, which rereads the L-system file, generates a new string, and interprets it to create a new image, but does not recalculate the bounding box,
- *New view*, which rereads the view file, resets the window and viewing parameters, and re-interprets the existing string to create a new image,
- *Animate mode*, which puts `cpfg` in animate mode, activating the animation menu, and
- *Exit*, which ends the process.

By editing the originally specified data files using a text editor or a virtual control panel, the user can change the model and have the result visualized by selecting either *New model*, *New view*, or *New L-system*.

The second user menu becomes available when the program is in animate mode. This mode is useful for producing developmental sequences of plants and as a debugging tool when developing L-system models. During the animation process, the L-system string is interpreted after each derivation step, the resulting image constituting the current frame in the animation. The animation process begins with the input of parameters, including specifications of the first frame and last frame of the animation, from an *animation file*. A sample of this file is described in Section A.3.3.

After reading the parameters, the animation routine derives the L-system string for the number of steps specified by the "first frame" parameter, then interprets it to create the initial image. At this point, the user can control the animation process by

selecting from the animation menu, which contains the same items as the main menu plus the following items.

- *Step* causes the execution of the next derivation step; the resulting string is interpreted to display the next frame.
- *Run* causes the animation to proceed frame by frame until the "last frame" is reached,
- *Forever*, has the same effect as *Run*, except that the animation is cycled with the "first frame" following the "last frame".
- *Stop* causes the animation to pause once the current frame has been displayed.
- *Rewind* causes the "first frame" of the animation to be displayed.
- *Clear* removes the image from the window.
- *New animate* rereads the animate file and proceeds to the new "first frame".
- *Don't animate* causes the program to leave animate mode and removes these items from the menu.

The animation parameters may be modified by editing the animation parameter file in a separate window and then selecting "New animate" to cause the animation parameter file to be reread.

A.3 Sample input files

This section provides examples of the data files for the model of *Anabaena catenula* described in Section 5.8.

A.3.1 L-system file

The syntax of L-system productions follows the specifications in Definitions 4.1, 4.5, 4.8, and Section 6.3. Spaces and tabs are allowed anywhere in the L-system

specification except within keywords. Multi-line productions are specified by starting each continuation line with a tab character. Expressions are defined as in Section 4.1; operator precedence is defined in Table 4.1, and the following function calls have been implemented: sin, cos, tan, acos, asin, atan, floor, ceil, trunc, abs, exp, log, and ran.

File name: anabaena.l

```
Lsystem: 1
derivation length: 200
ignore: f~C
Axiom: -(90)F(0,0,900)F(4,1,900)F(0,0,900)
F(s,t,c) : t==1 && s>=6 --> F(s/3*2,2,c)f(1)F(s/3,1,c)
F(s,t,c) : t==2 && s>=6 --> F(s/3,2,c)f(1)F(s/3*2,1,c)
F(h,i,k) < F(s,t,c) > F(o,p,r) : s>3.9 || c>.40 -->
        F(s+.1,t,c+0.25*(k+r-3*c))
F(h,i,k) < F(s,t,c) > F(o,p,r) : s<=3.9 && c<=.40 -->
        f(s*2)~C(s*2)F(0,0,900)f(s*2)
C(w) --> C(w+.025)
endlsystem
```

A.3.2 View file

A view file contains drawing, viewing, and rendering parameters including the names of surface specification files for any surfaces to be included in the image. The parameters can appear in any order but must be entered on separate lines.

File name: anabaena.v

```
angle factor: 12
initial color: 16
color increment: 32
initial line width: 15.0
line width increment: 1.0
```

```
viewpoint: 0,0,30
view reference point: 0,0,0
twist: 0
projection: parallel
front distance: -10000.0
back distance: 10000.0
scale factor: 0.8
z buffer: on
cue range: 0
shade mode: 3
light direction: 1.0,0.0,0.5
diffuse reflection: 10
tropism direction: 1.0,-3.0,0.0
initial elasticity: 0.0
elasticity increment: 0.0
surface ambient: .1
surface diffuse: .75
surface: C cyst.s 1.2
```

The first five parameters in this example (up to the **line width increment**) specify initial values for the turtle's state and environment, as do **tropism direction**, **initial elasticity**, and **elasticity increment** (as described in Chapter 3).

The three-dimensional viewing process is controlled by the **viewpoint**, **view reference point**, **twist**, **projection**, **front distance**, and **back distance** parameters. All these parameters have the standard IRIS graphics library definition. The final image is scaled relative to the window size by the **scale factor**.

Rendering is controlled by the parameters **z buffer**, **cue range**, **shade mode**, **light direction**, **diffuse reflection**, **surface ambient**, and **surface diffuse**.

The **surface:** lines are optional and a variable number can be included. The first parameter is the *id* symbol associated with a $\sim$ in the L-system generated string to identify a surface to be drawn by the turtle. The next parameter identifies the surface specification file name to be associated with the *id* symbol. The final parameter is a

scaling factor to be applied to the control points as they are read in.

A.3.3 Animation file

This file controls program operation in animate mode. The first three parameters are flags and can be either on or off.

File name: `anabaena.a`

```
double buffer: on
clear between frames: on
scale between frames: off
swap interval: 5
first frame: 1
last frame: 170
```

If the "double buffer" flag is on, the next frame is not drawn onto the screen but into a separate buffer. Once the drawing process is complete, this buffer is exchanged with the screen buffer, creating a smooth flow of images from frame to frame. If the "double buffer" flag is off, the next frame is drawn over top of the existing frame. The screen will be cleared between frames if the "clear between frames" parameter is on. If the "scale between frames" flag is on, the bounding box is determined and the image is scaled to fit the window before each frame is displayed. This has the effect of preventing the image from growing out of the window, but does not reveal relative size differences between frames. It is useful when studying fractals, but is typically not used in the case of plant development simulations. The "swap interval" parameter controls the speed of the animation by introducing a minimum time delay, expressed in screen refresh cycles, between the display of frames. The "first frame" parameter specifies the number of derivation steps to be performed before displaying the first frame. The "last frame" parameter specifies the number of the derivation step which is to be considered the last frame of the animation sequence. A value of -1 indicates that the L-system derivation length should be used to specify the last frame.

A.3.4 Surface specification file

The first section of a surface specification file contains information about the surface as a whole. This includes the extreme values of x, y, and z for the surface, the geometry parameters, and a scaling parameter giving the size to be considered as equivalent to the turtle's step size. This is followed by groups of nine lines, each describing one component patch. Each group consists of a patch name, patch-specific rendering information, three lines of patch neighbourhood information and four lines of patch control points, each line representing one row of four points. Further details of surface specification can be found in [61, Section 3.5].

File name: `cyst.s`

```
-75.00 75.00   -75.00 75.00   0.00 65.99
CONTACT POINT  X: 0.00 Y: 0.00 Z: 0.00
END POINT  X: 0.00 Y: 0.00 Z: 0.00
HEADING  X: 0.00 Y: 1.00 Z: 0.00
UP  X: 0.00 Y: 0.00 Z: 1.00
SIZE: 150.00
Patch_0
TOP COLOR: 0 DIFFUSE: 0.00 BOTTOM COLOR: 0 DIFFUSE: 0.00
AL: ~ A: ~ AR: ~
L: ~ R: ~
BL: ~ B: Patch_1 BR: ~
64.3 -32.27 0.0  70.7 -20.1  0.0  75.0 14.2  0.0  66.1 32.3 0.0
50.0 -55.90 0.0  50.0   0.0 27.6  50.0 30.0 45.1  50.0 58.0 0.0
30.0 -68.73 0.0  20.0 -23.7 66.0  20.0 30.0 40.4  30.0 68.3 0.0
 0.0 -75.00 0.0   0.0 -33.5 50.9   0.0 50.0 60.2   0.0 75.0 0.0
Patch_1
TOP COLOR: 0 DIFFUSE: 0.00 BOTTOM COLOR: 0 DIFFUSE: 0.00
AL: ~ A: Patch_0 AR: ~
L: ~ R: ~
BL: ~ B: ~ BR: ~
```

0.0	-75.0	0.0	0.0	-33.5	50.9	0.0	50.0	60.9	0.0	75.0	0.0
-30.0	-68.7	0.0	-20.0	-23.7	42.1	-20.0	30.0	53.2	-30.0	68.7	0.0
-50.0	-55.9	0.0	-50.0	0.0	38.1	-50.0	30.9	42.7	-50.0	55.9	0.0
-69.4	-26.4	0.0	-75.0	-11.3	0.0	-75.0	16.0	0.0	-64.2	36.3	0.0

Appendix B

IMPLEMENTATION CONSIDERATIONS FOR PARAMETRIC L-SYSTEMS

This appendix addresses issues related to my implementation of the parametric L-system formalism. The *continuous plant and fractal generator*, or cpfg, is a program written in C that reads L-system specifications from a file and creates a visualization of the model using turtle interpretation.

In the first stage of processing, the cpfg program generates a string by performing the given number of derivation steps for the input L-system model. In the next stage, the bounding box is determined if it has not been supplied by the user. The generated string is interpreted as described in Section 4.5, except that no drawing is performed. Instead, after each module is interpreted, the position of the turtle is processed to determine its effect, if any, on the bounding box. Once this process is complete, the window and viewing parameters are set appropriately. Finally, the string is re-interpreted to create an image on in the program's window.

B.1 Program organization

An overview of the cpfg program organization is presented in Figure B.1. The `control` module accepts input file names as command line arguments, reads in the files, and handles user interaction through mouse-activated menus, as described in

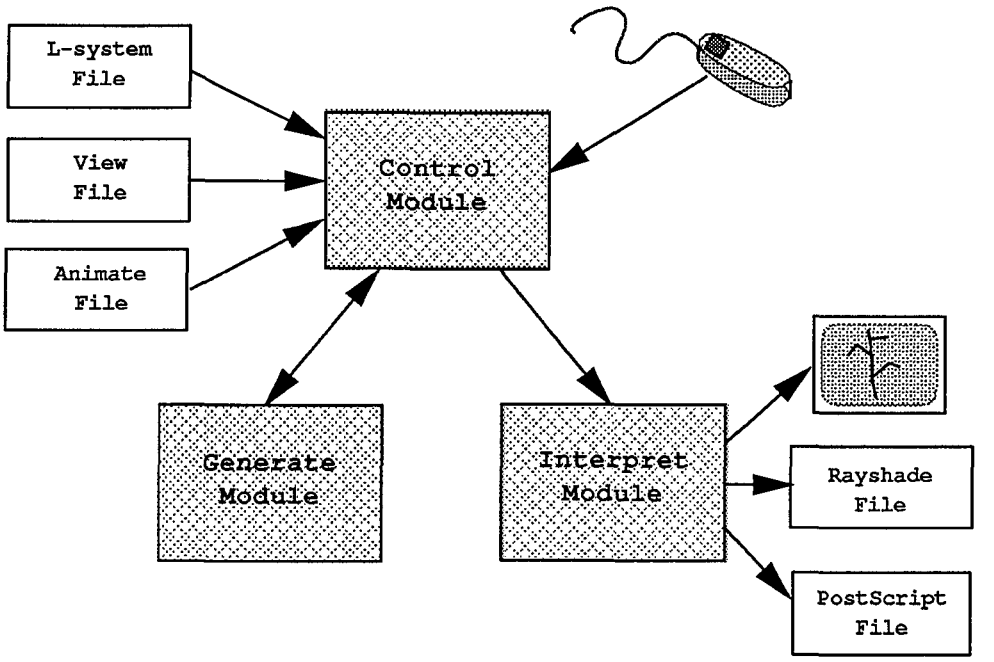


Figure B.1: `Cpfg` system overview

Appendix A. It calls the `generate` module to create the next stage in a model's development, and calls the `interpret` module to create the screen display or output in *rayshade* or PostScript format. The central data structure common to all three modules is the parametric string that represents the object being modelled. The `control` module initializes the current string, the `generate` module carries out the L-system derivation, and the `interpret` module applies turtle interpretation to the string to produce output.

The parametric string is represented in a straightforward fashion as a one-dimensional character array. A module is represented as a single letter followed by a comma-separated list of floating point parameters, enclosed in parenthesis. If a module has no parameters, no characters are stored after the letter. The floating point numbers are stored byte by byte in their internal form. For example, assuming that the internal representation of a floating point number takes 4 bytes of storage, a parametric word `A(9,.0125)BC(3.1546)` would be stored as

Byte	Contents	
1	A	
2	(	
3–6	□□□□	floating point representation of 9
7	,	
8–11	□□□□	floating point representation of .0125
12	)	
13	B	
14	C	
15	(	
16–19	□□□□	floating point representation of 3.1546
20	)	

Storing the parameters in this form allows faster access than storing them as ASCII numbers, while avoiding the storage-allocation and pointer-space overhead of a more complex data structure, such as a linked list. These are important considerations, as some models may require upwards of a million modules in later stages of development.

B.2 The control module

Once the names of files to be read and written have been determined from the command line, the `cpfg` program creates the menus and opens a window. The files are then read and the `generate` and `interpret` modules are called to display the model. The program then enters an event loop in order to respond to user command input via the mouse. This may result in files being reread, and new visualizations being created. Further details of the interactive aspects of the interface can be found in Appendix A.

Input procedures for the view file and animation file are straightforward. Sample files can be found in Sections A.3.2 and A.3.3, respectively. The L-system file is parsed by code generated using `lex` and `yacc`, which creates a list of L-systems, comprised of the main L-system and any sub-L-systems used in the model. A sample L-system file can be seen in Section A.3.1. The internal representation is closely related to the

file format; each L-system is composed of the following:

- the L-system identifier, used by the sub-L-system mechanism when selecting the rules to use for the particular sub-L-system reference (see Section 6.1),
- the derivation length,
- the list of characters to be ignored during context matching (see Section 2.5),
- an optional seed value for the random number generator, for stochastic L-systems,
- the axiom, a list of modules with parameter expressions in postfix form, and
- the set of productions, each production composed of the following:
  - a list of formal modules representing the left context in right to left order,
  - a formal module representing the strict predecessor,
  - a list of formal modules representing the right context in left to right order,
  - a pre-condition statement list,
  - a condition expression,
  - a post-condition statement list,
  - a list of formal modules representing the successor in left to right order,
  - a probability expression for productions in stochastic L-systems, and
  - a symbol table local to the production, containing all variable names defined in the predecessor and context modules, as well as those defined on the left hand side of any assignment statements in the statement lists.

  All expressions are stored in postfix order for easy evaluation. The symbol table associated with each production keeps track of the current binding of the variables during production application.
- lists of statements to be processed at specific times in the derivation cycle, as described in Section 6.3:

- at the start of the derivation process,
- at the start of each derivation step,
- at the end of each derivation step, and
- at the end of the derivation process,

- the global symbol table, containing global variable names defined on the left hand side of any assignment statements in the above lists.

Once the L-system has been read in from the file, two strings are created: the current string and the result string. The sub-L-system stack is initialized to contain the main L-system. The current string is initialized to contain the axiom with all expressions evaluated. It is then passed to the `generate` module to undergo a derivation, and the resulting string is passed back. If several derivation steps are required, the result string becomes the current string and the process is repeated until the desired number of derivations has been performed. At this point the final result string is passed to the `interpret` module. If animation is requested by the user, the interpretation is performed after each derivation step.

B.3 The generate module

As `cpfg` is implemented on a sequential machine, a parallel derivation is simulated by processing the string from left to right, one module after another. The production list for the L-system on the top of the sub-L-system stack is used to determine the matching production for each module (see Definitions 4.2 and 4.9). If an applicable production is found, the successor modules are appended to the result string after their expressions have been evaluated using the current values of variable names in the local and global symbol tables. If no production is found to match, the module is appended to the result string, as if an identity production for that module was included in the production set. If the letter of the module being processed is equal to a ?, the new sub-L-system identifier is pushed onto the stack, and processing continues. If the letter of the module being processed is equal to a $, the sub-L-system stack is popped, making the previous sub-L-system productions available. This derivation

process continues at the next module in the string, until all characters in the current string have been processed and the result of this derivation step has been produced.

B.4 The interpret module

The turtle interpretation process described in Section 4.5 can be summarized as follows. A three-dimensional LOGO-style turtle interprets the string modules one by one, from left to right. The state of the turtle is represented by its position and orientation, the colour and width of the lines that it draws, and its elasticity. Certain symbols are identified as commands that control this state, allowing the turtle to be rotated, moved, and to have its drawing attributes changed. The amount of change in the turtle's state is determined by the value of the interpreted module's first parameter if it is present, or by drawing environment parameters if not, in conjunction with the current scaling factor, which is initially set to 1. When the turtle is moved, it may also draw a line segment connecting consecutive positions in space. At the end of each move, tropism effects are simulated by modifying the turtle's orientation according to its current elasticity and an environment parameter which specifies the tropism direction.

In order to model branching structures, parametric L-system strings include brackets which enclose the substring to be considered as a branch. In the interpretation process, the left bracket causes the turtle's state to be pushed onto a stack. The "branch" is then drawn by interpreting the enclosed substring. When the right bracket is reached, the stack is popped, returning the turtle to its previous state and drawing of the parent branch is resumed.

If a polygon is to be drawn, consecutive positions of the turtle are stored in a polygon list pointed to by the top entry of the polygon stack. Whenever a { module is encountered, a new current polygon is pushed on the stack. It is popped when a } is encountered. These polygons can be rendered as wireframes, filled with constant colour, or Gouraud shaded.

Externally defined bicubic surfaces are incorporated into an image by interpretation of a $\sim$ followed by a module which identifies the particular surface to be drawn.

The identifying module may include a scaling factor as its first parameter. The surface parameters are read in from a "surface definition file" and include control points and geometric and neighbourhood information. A sample file is presented in Section A.3.4.

Special purpose interpretation routines can be incorporated using the "black box" mechanism. The user-supplied code must be compiled into the program, as described in Section 3.4. The programmer is free to use module parameters in any way, and has access to the turtle's state, as well as the viewing and drawing parameters read in from the view file.

When a sub-L-system reference, ?, is encountered, the current scaling factor is multiplied by the new sub-L-system scaling factor, and the result is pushed on a sub-L-system scale stack. When the end of sub-L-system character, $, is interpreted, the scaling factor from the top of the sub-L-system stack is divided out of the current scaling factor, and the stack is popped.

The `interpret` module is responsible for creating both the screen image and output files in various formats. In order to perform these functions without duplicating large amounts of code, a rendering routine dispatch table was implemented. The table is a data structure holding pointers to routines for rendering operations; turtle state updates are handled by the main interpretation routine. For instance, if PostScript output is required, the `control` module passes the dispatch table for the PostScript routines to the `interpret` module. If screen output is required, the dispatch table for Iris GL rendering routines is passed. For each interpreted symbol, the `interpret` module calls the function in the appropriate slot of the current dispatch table. The dispatch table provides access to the following routines:

- Setup — handles any required preprocessing, such as opening of output files, and generating header information specific to the desired output format,
- StartNode — processes the start of a line segment,
- EndNode — processes the end of a line segment,
- StartBranch — processes the start of a branch,

- EndBranch — processes the end of a branch,
- StartPolygon — initializes a polygon,
- EndPolygon — takes all vertices and draws the polygon according to the shade mode specified in the view file (see Section A.3.2),
- SetColour — sets rendering colour,
- SetLineWidth — sets width of subsequently drawn segments,
- Circle — draws a circle at the current position of the turtle,
- Sphere — draws a sphere at the current position of the turtle,
- BlackBox — handles rendering specific to a blackbox function,
- PredefinedSurface — draws a predefined surface,
- LdefinedSurface — draws an L-system defined surface,
- FinishUp — handles any required post-processing, such as closing files.

Either the StartNode or EndNode routine can be responsible for drawing the segment, depending on the type of rendering. The advantage of using the EndNode routine to do the drawing is that segments can be drawn with widths that differ at either end.

Appendix C

COLOUR PLATES

This appendix collects colour plates matching selected black and white figures in the text.

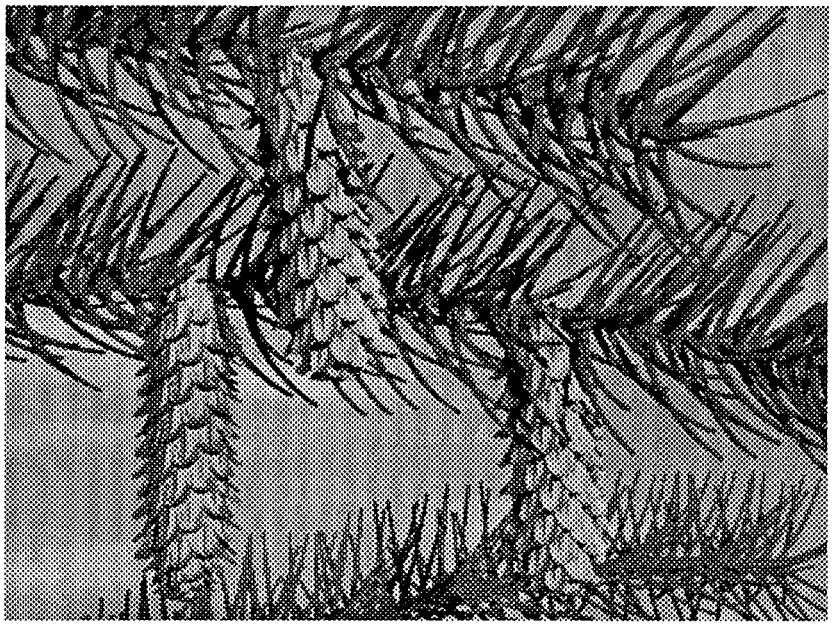


Figure C.1: Spruce cones ©1990 D. R. Fowler and J. Hanan

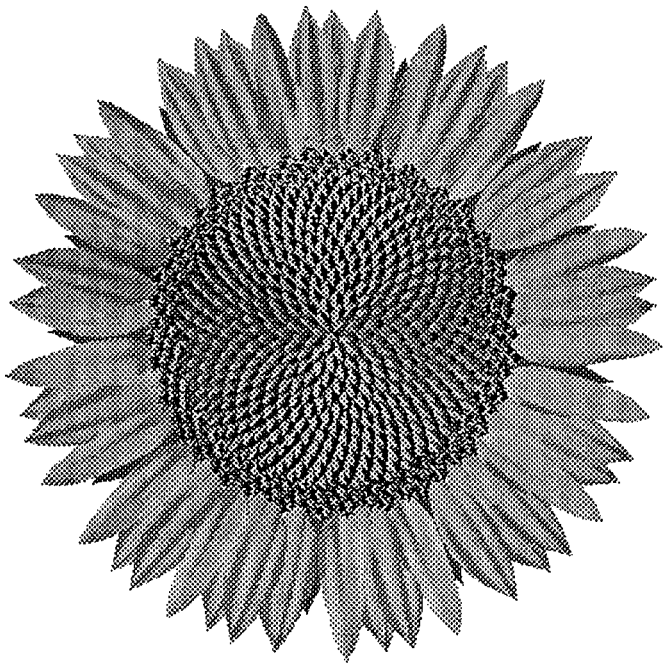


Figure C.2: A sunflower head

Figure C.3: Sunflower field ©1990 D. R. Fowler, N. Fuller, J. Hanan, and A. Snider

Figure C.4: Rose campion flower development ©1991 P. Prusinkiewicz and M. Hammel

Figure C.5: *Mycelis muralis* ©1987 P. Prusinkiewicz and J. Hanan

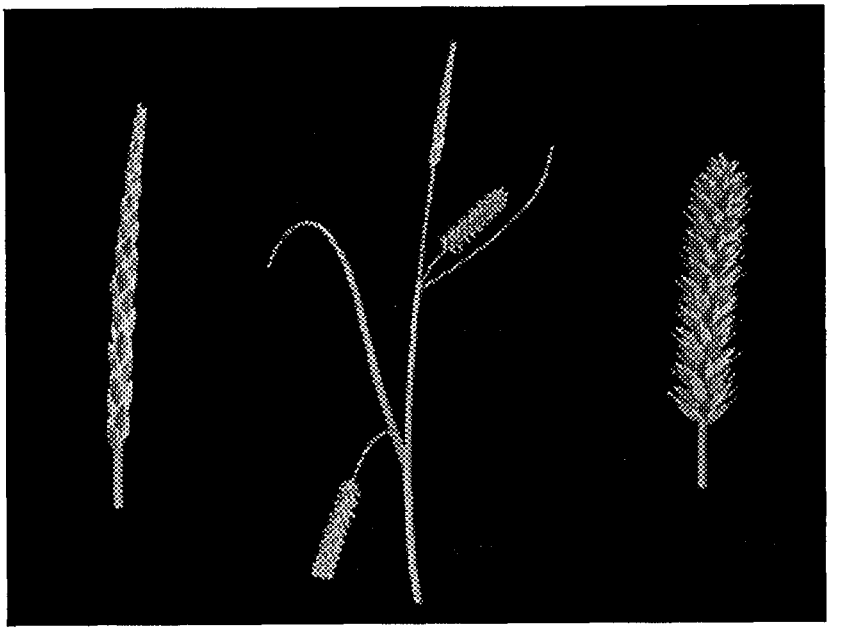


Figure C.6: *Carex laevigata* ©1989 J. Hanan and P. Prusinkiewicz

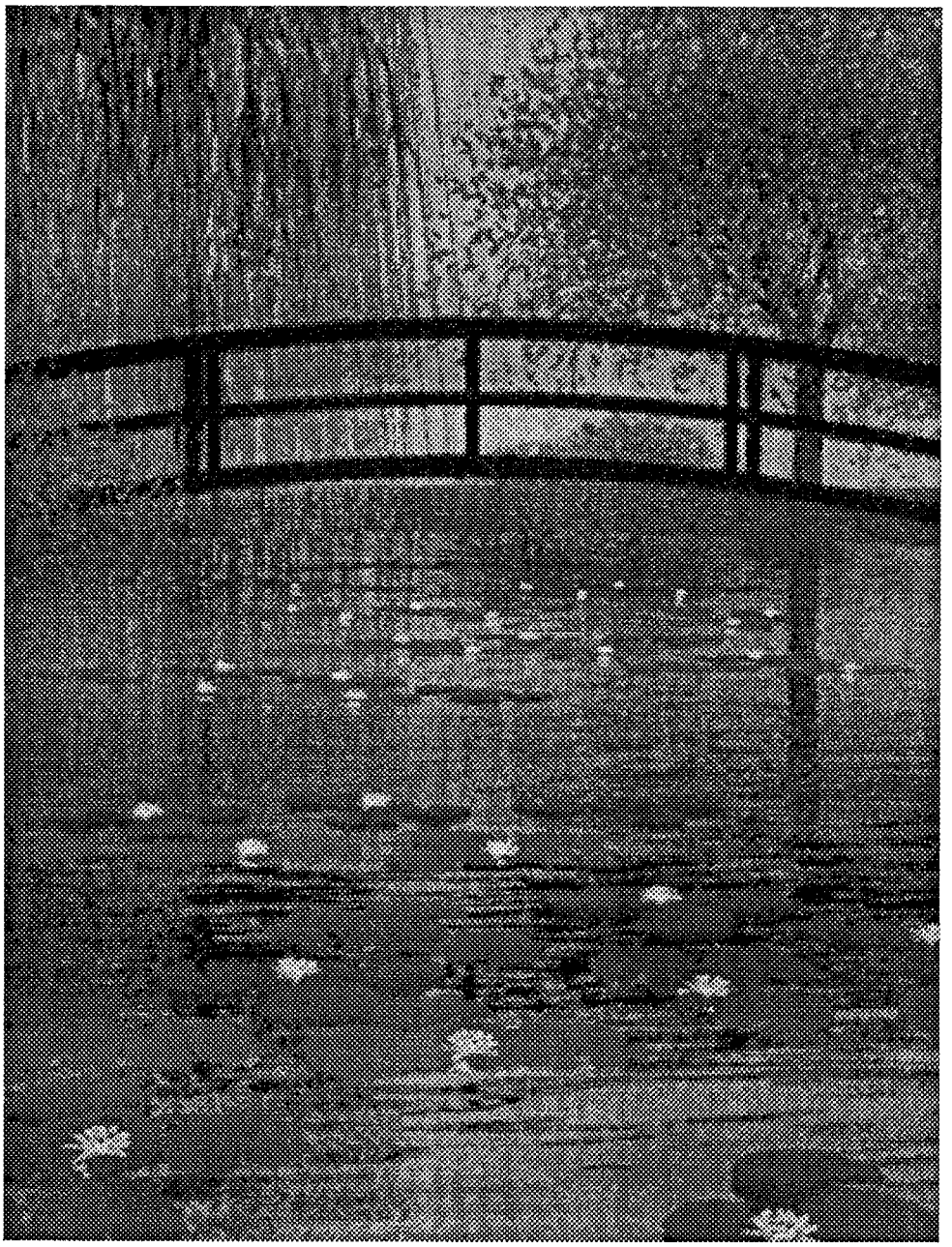


Figure C.7: Water-lilies ©1990 D. R. Fowler, J. Hanan, P. Prusinkiewicz, and N. Fuller