SIGGRAPH 2003 Course Notes

L-systems and Beyond

Organizer:

Przemyslaw Prusinkiewicz
Department of Computer Science
The University of Calgary

Instructors:

Pavol Federl
Department of Computer Science
The University of Calgary
2500 University Drive N.W.
Calgary, Alberta T2N 1N4, Canada
federl@cpsc.ucalgary.ca

Radomir Mech
SGI
1600 Amphitheatre Way
Mountain View, CA 94043
rmech@sgi.com

Radoslaw Karwowski
Department of Computer Science
The University of Calgary
2500 University Drive N.W.
Calgary, Alberta T2N 1N4, Canada
pwp@cpsc.ucalgary.ca

Przemyslaw Prusinkiewicz
Department of Computer Science
The University of Calgary
2500 University Drive N.W.
Calgary, Alberta T2N 1N4, Canada
pwp@cpsc.ucalgary.ca

Course Description

L-systems are a biologically-motivated formalism that can be used as a tool for modeling and visualizing biological structures, and as a computing technique gleaned from nature to solve other modeling problems. Their appeal lies in a compact, intuitive description of algorithms, and the possibility of using this description directly as an input to the modeling software. The course will present recent theoretical results, implementations, applications and research directions pertinent to L-systems and their extensions. The applications include, on one hand, the modeling and visualization of plants at different levels of abstraction and for a variety of purposes, and, on the other hand, geometric modeling of curves and surfaces. These applications are united by their treatment of the modeled objects as dynamical structures subject to local operations, and can be implemented using the same modeling software. The course will be of a particular interest to researchers and students working on geometric modeling and the modeling of nature.

Prerequisites

The course will assume basic knowledge of geometric modeling algorithms, in particular subdivision curves and surfaces, and of numerical methods for solving algebraic and (ordinary and partial) differential equations. Prior exposure to L-systems, fractals, and the modeling of plants is desirable, but not necessary.

Speaker Biographies

Pavol Federl is a research associate in the Department of Computer Science at the University of Calgary, where he also received his Ph.D. His graduate work was done under the direction of Dr. Przemyslaw Prusinkiewicz, and involved physically based simulations of fracture patterns. Dr. Federl's current research interests lie in the area of simulating growth in biological structures. He is also involved in the development of Virtual Laboratory (VLAB), an interactive and collaborative simulation environment.

Radoslaw Karwowski is a research associate in the Department of Computer Science at the University of Calgary. He holds an M.Sc. in Computational Physics from the University of Wroclaw, Poland, and a Ph.D. in Computer Science from the University of Calgary. From 1996 to 1998 he worked on the modeling and simulation of plant development in the Institute of Botany at the University of Wroclaw. His current research interests are in the domain of modeling languages and simulation methods applicable to plant modeling. He is a co-creator and developer of the L-system-based plant modeling software L-studio.

Radomir Mech is a technical staff member at SGI, Mountain View, CA. He received a Ph.D. in Computer Science from the University of Calgary in 1997, where he completed a dissertation under the direction of Dr. Przemyslaw Prusinkiewicz. His graduate work was devoted to L-system models of plants interacting with their environment. After his Ph.D., Dr. Mech joined the OpenGL Performer team at SGI, where he developed algorithms enhancing the visual quality of real-time rendering. Dr. Mech is the (co)author of several research papers, including three papers presented at SIGGRAPH. His current research spans real-time rendering and procedural modeling.

Przemyslaw Prusinkiewicz is a Professor of Computer Science at the University of Calgary. He holds an M.Sc. and Ph.D., both in Computer Science, from the Technical University of Warsaw. Before joining the faculty of the University of Calgary, he was a faculty member at the University of Regina, Canada, University of Science and Technology of Algiers, and Technical University of Warsaw. He was also a Visiting Professor at Yale University (1988), l'Ecole Polytechnique Federale de Lausanne (1990), and a Visiting Researcher at the University of Bremen (1989) and the Centre for Tropical Pest Management in Brisbane (1993, 1994, 1998). His research combines computer graphics with concepts rooted in biology, formal language theory, and mathematics. He originated a method for visualizing the structure and growth of plants based on L-systems, a mathematical model of development. He is the co-author of two books, "The Algorithmic Beauty of Plants", and "Lindenmayer Systems, Fractals, and Plants", and numerous papers in this area. He was the organizer of the 1992 SIGGRAPH course on fractals, and a speaker in over 15 other SIGGRAPH courses on fractals, procedural modeling, artificial life, and modeling of natural phenomena. Dr. Prusinkiewicz is the recipient of the 1997 ACM SIGGRAPH Computer Graphics Achievement Award.

Course Schedule

Part 1: Introduction to L-systems

1:45: Introduction to L-systems: theory, modeling, and graphics
Prusinkiewicz

Part 2: Plant modeling with L-systems

2:15 Foundations: simulating control processes in plants
Mech

2:45 Interlude: solving algebraic and differential equations with L-systems
Federl

3:15 Break

3:30 Advanced plant modeling: genes, physiology, and biomechanics
Prusinkiewicz

Part 3: Geometric modeling with L-systems

4:00 Application of L-systems to geometric modeling of curves
Prusinkiewicz

4:20 Extending L-systems to surfaces
Prusinkiewicz

Part 4: Implementations of L-systems

4:40 Designing and implementing an L-system-based language
Karwowski

5:00 Hardware implementation of L-systems
Mech

5:20 Questions and answers
Federl, Karwowski, Mech, Prusinkiewicz

Table of Contents

Structured Dynamical Systems

Przemyslaw Prusinkiewicz[1]
Department of Computer Science
University of Calgary

Abstract

This note introduces the notion of structured dynamical systems, and places L-systems in their context.

1 Basic definitions

Many natural phenomena can be modeled as *dynamical systems*. At any point in time, a dynamical system is characterized by its *state*. A state is represented by a set of *state variables*. For example, in the description of planetary motions around the sun, the set of state variables may represent positions and velocities of the planets. Changes of the state over time are described by a *transition function*, which determines the next state of the system as a function of its previous state and, possibly, the values of external variables (input to the system). This progression of states forms a *trajectory* of the system in its *phase space* (the set of all possible states of the system).

Mathematical objects with diverse properties can be considered dynamical systems. For instance, state variables may take values from a continuous or discrete domain. Likewise, time may advance in continuous or discrete steps. Examples of dynamical systems characterized by different combinations of these features are listed in Table 1.

Table 1: Some formalisms used to specify dynamical systems according to the discrete or continuous nature of time and state variables.

C: continuous, D: discrete.	ODE	Iterated Mappings	Finite Automata
Time	C	D	D
State	C	C	D

[1]Adapted from: J.-L Giavitto, C. Godin, O. Michel and P. Prusinkiewicz, *Computational Models for Integrative and Developmental Biology*, LaMI Rapport de Recherche N^o **72-2002**, March 2002, Section 2.

In simple cases, trajectories of dynamical systems may be expressed using mathematical formulas. For example, the ODE (ordinary differential equation) describing the motion of a mass on a spring has an analytical solution expressed by a sine function (linear spring, in the absence of friction and damping). In more complex cases, analytic formulas representing trajectories of the system may not exist, and the behavior of the system is best studied using computer simulations.

By their nature, simulations operate in discrete time. Models initially formulated in terms of continuous time must therefore be discretized. Strategies for discretizing time in a manner leading to efficient simulations have extensively been studied in the scope of simulation theory, *e.g.* [6].

Dynamical systems with apparently simple specifications may have very complex trajectories. This phenomenon is called *chaotic behavior*, *c.f.* [13], and is relevant to biological systems (for example, in populations models) [10, 11].

2 Structured dynamical systems

Many dynamical systems can be decomposed into parts. The advancement of the state of the whole system is then viewed as the result of the advancement of the state of its components. For example, a developing plant can be described in terms of its functional units (modules), such as apices, internodes, leaves, and flowers (reviewed in [15]). Similarly, the operation of a gene regulation network can be described in terms of interactions between individual genes, and gene activities in interacting cells (e.g. [12, 14, 18, 19]).

Formally, we use the term *structured dynamical system* to denote a dynamical system divided into component subsystems (units). The set of state variables of the whole system is the Cartesian product of the sets of state variables of the component subsystems. Accordingly, the state transition function of the whole system can be described as the product of the state transition functions of these subsystems. Similarly to non-structured systems, structured dynamical systems can be defined assuming continuous or discrete state variables and time. In addition, the components can be arranged in a continuous or discrete manner in space. Some of the formalisms resulting from different combinations of these features are listed in Table 2.

Time management is an important issue in the modeling and simulation of structured systems [8]. For example, state transitions may occur *synchronously* (simultaneously in all components) or *asynchronously*. Furthermore, efficient simulation techniques may assume different rates of time

Table 2: Some formalisms used to specify structured dynamical systems according to the continuous or discrete nature of space, time, and state variables of the components. The heading "Numerical Solutions" refers to explicit numerical solutions of partial differential equations and systems of coupled ordinary differential equations.

C: continuous D: discrete	PDE	Coupled ODE	Numerical Solutions	Cellular Automata
Space	C	D	D	D
Time	C	C	D	D
States	C	C	C	D

progression in different components [5].

In many cases, the transition function of each subsystem depends only on a (small) subset of the state variables of the whole system. If the components of the system are discrete (i.e., excluding partial differential equations, or PDEs), these dependencies can be depicted as a *directed graph*, with the nodes representing the subsystems and the arrows indicating the inputs to each subsystem. We say that this graph defines the *topology* of the structured dynamical system, and call *neighbors* the pairs of subsystems (directly) connected by arrows.

The topology of a structured dynamical system may reflect its *spatial organization*, in the sense that only physically close subsystems are connected. A dynamical system with this property is said to be *locally* defined. Locality is an important feature of systems that model physical reality, because physical means of information exchange ultimately have a local character (e.g., transport of signaling molecules between neighboring cells). On the other hand, physically-based models need not to be rigorously local. For example, when modeling plants, it may be convenient to assume that higher branches cast shadow on lower branches without simulating the local mechanism of light propagation through space.

When the number of components in a structured dynamical systems is large, the exhaustive listing of all connections between the components becomes impractical or infeasible. This limitation can be overcome in several ways. For example, if the components are arranged in a regular pattern, the neighbors of each component need not to be listed explicitly. This is the case of *cellular automata* (*e.g.* [17, 20]), in which cells are arranged in a

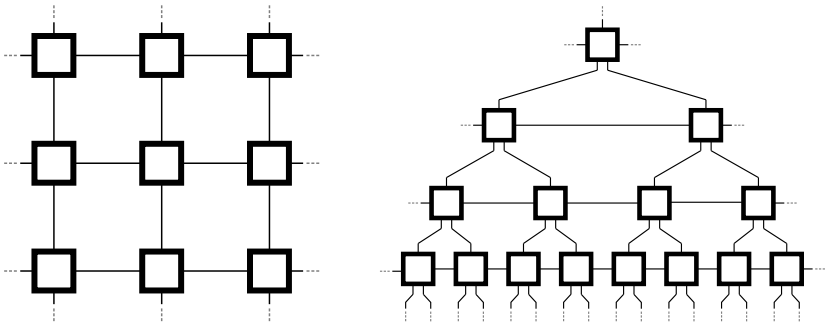


Figure 1: Two examples of regular patterns of connections in structured dynamical systems: Euclidean grid, in which each cell has four neighbors (von Neumann neighborhood, left), and hyperbolic grid, in which each cell has five neighbors (right).

square grid. The neighbor of each cell (excluding the cells, if present), can be referred to in a uniform way, using the directions North, South, East and West. Cellular automata operating in the hyperbolic plane [9] and *group-based fields* [1] are generalizations of this idea, allowing for a wider range of connecting patterns (Figure 1). Large structures can also be defined by *simulated development*, as it is discussed in the next section.

3 Dynamical systems with a dynamic structure

A developing multicellular organism can be viewed as a dynamical system in which not only the values of state variables, but also the *set* of state variables and the state transition function change over time. These phenomena can be captured using an extension of structured dynamic systems, in which the set of subsystems or the topology of their connections may dynamically change. We call these systems *dynamical systems with a dynamic structure* [1], or $(\mathrm{DS})^2$-systems in short.

For example, let us consider a multicellular plant structure, defined at the level of individual cells. A large network of interconnected cells can by gradually created by simulating the process of cell division. When a cell divides, the topology of this network is adjusted to:

- remove connections (neighborhood relations) between the mother cell and the rest of the organism,
- create connections between the daughter cells,

- insert connections between the daughter cells and the remainder of the system.

Other examples of natural processes that can be viewed as $(DS)^2$-systems include:

- chemical reactions, considered as interactions between molecules,
- developing plants, in which apical meristems create new meristems, branch segments, leaves, and flowers,
- developing ecosystems, in which plants and animals reproduce, interact with each other, and die.

The concept of $(DS)^2$-systems can also be extended to more abstract, mathematical constructs, such as:

- fractal generation using Koch construction,
- generation of curves and surfaces using subdivision and pyramid algorithms [4],
- numerical solution of PDEs using adaptive numerical methods, in which the discretization of space changes over time,
- parallel algorithms, in which the number of interconnected computing elements changes as the computation proceeds.

In the above examples, time is the independent variable that drives the system dynamics. It is also possible to consider $(DS)^2$-systems in which the independent variable has spatial character. These include multiresolution representations, in which the level of detail is controlled by the scale of observations.

From the computer science point of view, simulation of dynamical systems with a dynamical structure raises the problem of finding a programming paradigm (or the modeling language) well suited to the specification of such systems. The key difficulty is that, in $(DS)^2$-system, not only the values of variables that describe the system, but also the entire set of variables, and equations that relate them, change over time. If the number of these changes is small, they can be specified explicitly. In general, however, we need a formalism that supports these changes in an automatic manner. Table 3 presents some of the existing formalisms for handling systems that operate in discrete space.

Table 3: Some formalisms used for the modeling of (DS)^2, according to the underlying topology of space.

Topology	Multiset	Sequence	Array	Graph
Formalism	multiset rewriting	L-systems	cellular automata	map L-systems, graph grammars, MGS, VV-systems, Cellerator

In this table, the first line gives the type of the topology used to connect the subcomponents of a system. In a *multiset*, all elements are considered to be connected to each other. In a *sequence*, elements are ordered linearly; this case includes lists and extends to tree-like structures. *Array* structures represent regular neighborhoods; for example, the lattice shown in Figure 1. In this case, addition of new elements to the structure may only occur at its boundary. Finally, *graphs* are used to define arbitrary connections between discrete components. Formalisms for describing (DS)^2 operating on graphs are less well developed than the formalisms operating on multisets, sequences, and arrays. Current research includes MGS (acronym for *un Modèle Général de Simulation de système dynamique* [1, 2, 3]) and vertex-vertex systems, outlined later in these notes. Cellerator [16] is a recent example of a practical simulation system operating on dynamically reconfigured graphs.

4 Conclusions

Structured dynamical systems consist of interconnected components. In dynamical systems with a dynamic structure, or (DS)^2, the set of components and their connections may change over time. This raises the problem of characterizing these changes as a part of a (DS)^2 specification. One possibility is to assume that the (DS)^2 components exist in some topological space, and use proximity of components in this space to define their connections. These connections define, how the variables in one component are related to the variables in its neighbors. Within this general framework, several formalisms for modeling (DS)^2 exist. Specifically, L-systems [7] make it possible to model dynamically reconfigurable structures with linear or branching topology.

References

[1] J.-L. Giavitto and O. Michel. MGS: A programming language for the transformation of topological collections. Research Report 61-2001, CNRS - Université d'Evry Val d'Esonne, Evry, France, 2001.

[2] J.-L. Giavitto and O. Michel. Data structures as topological spaces. In *Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*, volume 2509, pages 137–150, 2002. Lecture Notes in Computer Science.

[3] J.-L. Giavitto and O. Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002.

[4] R. Goldman. *Pyramid algorithms. A dynamic programming approach to curves and surfaces for geometric modeling.* Morgan Kaufmann, San Francisco, 2003.

[5] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[6] W. Kreutzer. *System simulation: Programming styles and languages.* Addison-Wesley, Sydney, 1986.

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

[8] N. A. Lynch. *Distributed algorithms.* Morgan Kauffman, Los Altos, CA, 1996.

[9] M. Margenstern. Celular automata in the hyperbolic plane. Report 99-103, Groupe d'Informatique Fondemantale de Metz, Université de Metz, 1999.

[10] R. M. May. Biological population models obeying difference equations: Stable points, stable cycles, and chaos. *Journal of Theoretical Biology*, 51:511–524, 1975.

[11] R. M. May. Simple mathematical models with very complicated dynamics. *Nature*, 261:459–467, 1976.

[12] E. Mjolsness, D. Sharp, and J. Reinitz. A connectionist model of development. *Journal of Theoretical Biology*, 152:429–453, 1991.

[13] H.-O. Peitgen, H. Jürgens, and D. Saupe, editors. *Chaos and fractals. New frontiers of science*. Springer-Verlag, New York, 1992.

[14] J. Reinitz and D. Sharp. Mechanism of *eve* stripe formation. *Mechanisms of Development*, 49:133–158, 1995.

[15] P. M. Room, L. Maillette, and J. Hanan. Module and metamer dynamics and virtual plants. *Advances in Ecological Research*, 25:105–157, 1994.

[16] B. Shapiro and E. Mjolsness. Developmental simulations with Cellerator. In *Proceedings of the Second International Conference on Systems Biology*, pages 342–351, 2001.

[17] T. Toffoli and N. Margolus. *Cellular automata machines: A new environment for modeling*. MIT Press, Cambridge, Massachusetts, 1987.

[18] G. von Dassow, E. Meir, E. Munro, and G. Odell. Ingeneue manual v. 0.7. `http://www.beakerware.com/ingeneue/`.

[19] G. von Dassow, E. Meir, E. Munro, and G. Odell. The segment polarity network is a robust developmental module. *Nature*, 406:188–192, 2000.

[20] S. Wolfram. *A new kind of science*. Wolfram Media, Champaign, IL, 2002.

Introduction to Modeling with L-systems

Przemyslaw Prusinkiewicz[1]
Department of Computer Science
University of Calgary

Abstract

This note introduces L-systems as a programming language for developmental model construction. The examples presented here refer to a real biological structure, the vegetative segment of a cyanobacterium *Anabaena catenula*. Problems related to time management in developmental simulations are given particular attention.

1 Background

L-systems were introduced by A. Lindenmayer as a mathematical formalism for modeling multicellular organisms that form linear or branching filaments [11, 12]. The whole organism is treated as an assembly of discrete units, called *modules*. The nature of the modules is not predefined by the formalism. For example, they may represent individual cells in the models of lower organisms, or functional units, such as apical meristems, internodes, leaves and flowers, in the models of higher plants. Each module is represented by a symbol (a letter of the L-system *alphabet*), which specifies the module's *type*. In addition, a module can be characterized by one or more numerically-valued *parameters* [2, 13], which, together with the symbol, characterize the module's *state*. The resulting concept of *parametric L-systems* has been formalized in [5, 23], among others. Recent extensions make it possible to use parameters of compound data types, such as structures [7] and lists [3].

L-system models are inherently *dynamic*, which means that the form of an organism is viewed as a result of development: "an event in space-time, and not merely a configuration in space" [28]. The development of a

[1]Based in part on: P. Prusinkiewicz: Introduction to modeling using L-systems, in J. A. Kaandorp and J. E. Kübler (Eds.), *The Algorithmic Beauty of Seaweeds, Sponges, and Corals*, Springer, Berlin, 2001, pp. 91–93, and P. Prusinkiewicz, J. Hanan and R. Měch, An L-system-based plant modeling language, *Lecture Notes in Computer Science* **1779**, Springer, Berlin, pp. 395–410.

structure is described in terms of *rewriting rules* or *productions* that change the module's state, or replace a module by zero, one or several new modules. Productions are applied in parallel, which is intended to capture the simultaneous progress of time in all parts of the growing organism.

L-systems were originally introduced as a mathematical tool for reasoning about development. Soon after their introduction, however, they began to be used as a formal basis for constructing simulation modeling programs and modeling languages. The availability of these languages is one of the key practical advantages of L-systems in modeling applications, because diverse structures and processes can be modeled using the same L-system-based procedural modeling program with different inputs. The L-system-based languages have initially been almost indistinguishable from the underlying mathematical notation of L-systems, but, with the increased complexity of the modeling tasks, they gradually diverged from this notation.

The notion of L-systems is often presented in a formal manner. In contrast, we will present it in the context of a specific modeling application. This will allow us to see the relationship between a natural phenomenon and its L-system models, highlight the conceptual essence of L-systems, see the motivation behind their extensions, and trace the evolution of L-system-based modeling languages. In addition, the examples address the question of time management in simulations, which is essential to the animation of development.

2 Model organism: *Anabaena catenula*

Anabaena is one of the earliest examples of multicellular organisms with cellular specialization [1, 4]. Like many other cyanobacteria, it is capable both of aerobic photosynthesis and fixation of nitrogen from the atmosphere. These processes, however, are incompatible with each other, because the enzyme nitrogenase responsible for nitrogen fixation is destroyed by oxygen, a byproduct of photosynthesis. Some cyanobacteria deal with this incompatibility by dividing their activities over time: they photosynthesize in the daytime, and fix nitrogen in the night. *Anabaena*, in contrast, divides these activities in space, using two different types of cells: *vegetative cells* responsible for photosynthesis, and *heterocysts* responsible for nitrogen fixation. This division of tasks is the rationale for the multicellular structure of the organism.

Anabaena cells are arranged into filaments. In the absence of ammonia or nitrate in the medium, individual heterocysts are separated by sequences of approximately ten vegetative cells of varying sizes. The filaments grow by asymmetric division of vegetative cells. As this process moves the existing heterocysts apart, new heterocysts differentiate from vegetative cells roughly midway between those already present. Consequently, the regular spacing of heterocysts remains approximately constant while the filament grows.

The developmental process outlined above raises two questions, which have attracted the interest of biologists and modelers alike:

1. What mechanism controls the asymmetric division of vegetative cells, resulting in the observed sequence of cell sizes along the filament; and

2. What mechanism regulates the spacing of heterocysts?

In this note we will address the first question.

3 Context-free L-systems

Mitchison and Wilcox [18] proposed to explain the observed pattern of long and short cells in a vegetative segment of *Anabaena* as a direct result of a developmental process. To this end, they divided the cells that form the filament into two classes: long cells L and small cells S. Furthermore, they assumed that each vegetative cell has one of two possible polarities, which can be indicated by an arrow: $\overrightarrow{L}$, $\overleftarrow{L}$, and $\overrightarrow{S}$, $\overleftarrow{S}$. During the development, cells S elongate and change their state to L. Long cells divide, producing a cell L and a cell S.

Lindenmayer [16] observed that, taking polarities into account, the essence of the model of Mitchison and Wilcox can be written as a set of *rewriting rules*, or *productions*:

$$\overrightarrow{S} \longrightarrow \overrightarrow{L} \qquad \overleftarrow{S} \longrightarrow \overleftarrow{L} \qquad \overrightarrow{L} \longrightarrow \overleftarrow{L}\,\overrightarrow{S} \qquad \overleftarrow{L} \longrightarrow \overleftarrow{S}\,\overrightarrow{L} \tag{1}$$

These rules are the cornerstone of a DOL-system, the simplest type of L-systems [12] (see also [6, 26, 23, 27]). The acronym DOL stands for a **D**eterministic **L**-system with **0** interactions. This means that exactly one production applies to any symbol of the L-system alphabet, and the productions are *context-free*. The DOL-system productions have the form

$$\textit{predecessor} \longrightarrow \textit{successor},$$

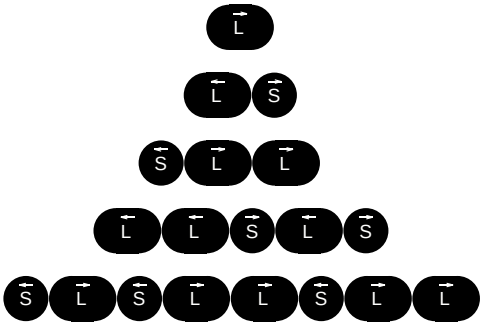


Figure 1: Developmental sequence of an *Anabaena* filament modeled with L-system 1.

where the *predecessor* is a single symbol of the L-system alphabet V, and the *successor* is a string of zero, one, or several symbols from the same alphabet.

Like all L-systems, DOL-systems operate on sequences of symbols called *strings* or *words*. In a single *derivation step*, each letter in the predecessor string is replaced by its *successor* using the applicable production from the *production set* P. The developmental process is simulated as a sequence of such derivation steps, beginning with a given initial string, called the *axiom*, and denoted ω. For example, Figure 1 shows the developmental sequence generated by L-system $\mathcal{G} = \langle V, \omega, P \rangle$ with alphabet $V = \{\overrightarrow{L}, \overleftarrow{L}, \overrightarrow{S}, \overleftarrow{S}\}$, axiom $\omega = \overrightarrow{L}$, and the production set P given by Equation 1.

The above model implies that the time between the formation and division of a short cell is twice as long as the time between the formation and division of a long cell. In reality, a short cell takes only about 20% longer to divide that a long cell. Lück and Lück [17], and Lindenmayer [15] incorporated this behavior into the model by introducing a sequence of state transitions to control the timing of cell division:

$$\begin{array}{ccccc} \overrightarrow{S} \longrightarrow \overrightarrow{L} & \overrightarrow{L} \longrightarrow \overrightarrow{A} & \overrightarrow{A} \longrightarrow \overrightarrow{B} & \overrightarrow{B} \longrightarrow \overrightarrow{C} & \overrightarrow{C} \longrightarrow \overleftarrow{L}\overrightarrow{S} \\ \overleftarrow{S} \longrightarrow \overleftarrow{L} & \overleftarrow{L} \longrightarrow \overleftarrow{A} & \overleftarrow{A} \longrightarrow \overleftarrow{B} & \overleftarrow{B} \longrightarrow \overleftarrow{C} & \overleftarrow{C} \longrightarrow \overleftarrow{S}\overrightarrow{L} \end{array} \tag{2}$$

Sequences of state transitions are not intuitively represented by the L-system formalism, but can be conceptualized using diagrams, similar to that shown in Figure 2. Similar diagrams have been used by Lindenmayer [10] and Lück and Lück [17]. Formally, they are a synchronously operating vari-

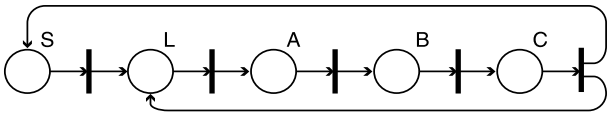


Figure 2: Petri net representation of L-system 2. A cell undergoes a sequence of transitions (vertical bars), which advance its state (circles). The last transition represents division of cell C into two children cells, S and L.

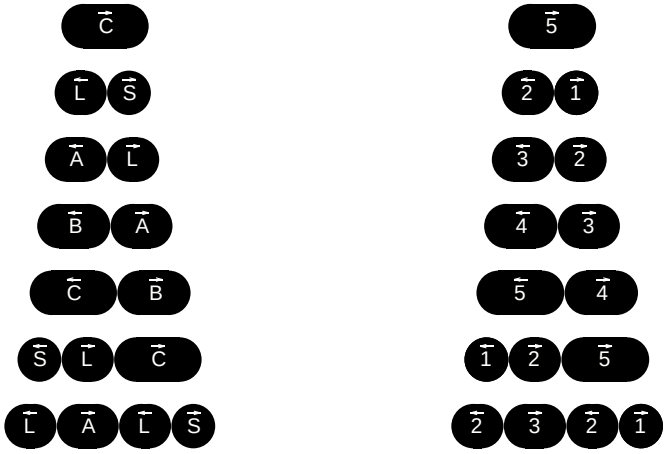


Figure 3: Improved developmental sequences of the *Anabaena* vegetative filaments. The models that take into account timing of the short and long cell divisions. Left: the sequence generated by non-parametric L-system 2. Right: the corresponding sequence generated by the parametric L-system 3.

ant of Petri nets [19, 25]. As illustrated in Figure 2, L-system 2 specifies a sequence of developmental stages, through which a cell progresses until it reaches the dividing state C. The children of cell C have different positions along this sequence, and therefore need different times to divide again. Specifically, the time to the next division is equal to 4 time units for a cell L and 5 time units for a cell S, thus resulting in the required ratio of 80%. This timing is also evident in the developmental sequence shown in Figure 3 on the right.

4 Parametric L-systems

Sequences of developmental stages significantly increase the size of the L-sysem alphabet and the size of the corresponding production set, and consequently complicate the specification of DOL-systems. *Parametric L-systems* [5, 23] offer a solution to this problem, by associating numerical parameters with L-system symbols. Context-free parametric productions have the form

$$predecessor : condition \longrightarrow successor,$$

where *predecessor* is a single module (L-system symbol with the associated parameters), *successor* is a sequence of modules, and *condition* is the logical expression that determines whether the production can be applied to a given module. For example, a parametric L-system equivalent to that in Equation 2 is given below:

$$\begin{array}{lcll} \vec{M}\,(s): & s<5 & \longrightarrow & \vec{M}\,(s+1) \\ \vec{M}\,(s): & s==5 & \longrightarrow & \overleftarrow{M}\,(2)\,\vec{M}\,(1) \\ \overleftarrow{M}\,(s): & s<0 & \longrightarrow & \overleftarrow{M}\,(s+1) \\ \overleftarrow{M}\,(s): & s==5 & \longrightarrow & \overleftarrow{M}\,(1)\,\vec{M}\,(2) \end{array} \tag{3}$$

The non-parametric L-system 2 and the parametric L-system 3 are equivalent because modules type in L-system 2 correspond one-to-one to parameter values in L-system 3:

$$S \leftrightarrow 1, \qquad L \leftrightarrow 2, \qquad A \leftrightarrow 3, \qquad B \leftrightarrow 4, \qquad C \leftrightarrow 5. \tag{4}$$

This correspondence is also illustrated by Figure 3, which compares the developmental sequences generated by both L-systems.

Parameters are useful in specifying various attributes of modules. For example, in the case of *Anabaena*, cell polarity can also be represented as a parameter. This results in the following L-system:

```
/* L-system 4 */
#define LEFT -1
#define RIGHT 1
Axiom: M(5,RIGHT)
M(t,p) : t<5 --> M(t+1,p)
M(t,p) : t==5 && p == LEFT --> M(1,LEFT)M(2,RIGHT)
M(t,p) : t==5 && p == RIGHT --> M(2,LEFT)M(1,RIGHT)
```

We have switched here from the mathematical notation to a programming language style of L-system specification. The particular language used has been devised for the L-system-based modeling program `cpfg` [22]. At this point, the differences between the mathematical notation and the corresponding `cpfg` specification are minimal, but the programming language style will allow us to more conveniently present further extensions to L-systems.

5 Modeling development in continuous time

All the L-systems discussed above advance time by unit interval per derivation step. In many applications, for example the animation of development, it is convenient to advance time by arbitrary user-defined increments. We can partially achieve this goal by reinterpreting the step-counting variable t in L-system 4 as a continuously-valued *developmental stage* or *physiological age* of the cell, and increment it by a constant dt. By applying this concept to L-system 4, and rescaling time so that the short cells have unit life span, we obtain:

```
/* L-system 5 */
#define LEFT -1
#define RIGHT 1
#define t_div 1.0       /* cell age at division */
#define t_s 0.0         /* initial age - short cell */
#define t_l 0.2         /* initial age - long cell */
#define dt 0.7          /* time increment per step */
Axiom: M(0,RIGHT)
M(t,p) : t+dt<t_div --> M(t+dt,p)
M(t,p) : t+dt>=t_div && p == LEFT -->
        M(t+dt-t_div+t_s,LEFT) M(t+dt-t_div+t_l,RIGHT)
M(t,p) : t+dt>=t_div && p == RIGHT -->
        M(t+dt-t_div+t_l,LEFT) M(t-t_div+t_s+dt,RIGHT)
```

Figure 4 shows developmental sequences generated by this L-system for two values of time increment, $dt_1 = 0.7$ and $dt_2 = 1.4$. A comparison of these sequences reveals that the results are correct for the smaller time increment dt_1, but incorrect for the larger increment dt_2. The reason for this discrepancy is presented in Figure 5, which places the branching process of cell division in the context of time increments used in both simulation. If

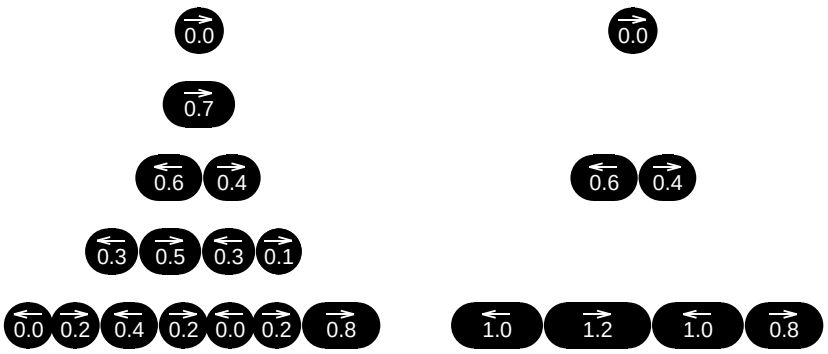


Figure 4: Developmental sequences generated by L-system 5 using time steps $dt_1 = 0.7$ (left) and $dt_2 = 1.4$ (right). The sequence on the left correctly captures the development of the filament, whereas sequence on the right does not, because it includes cells past their division age.

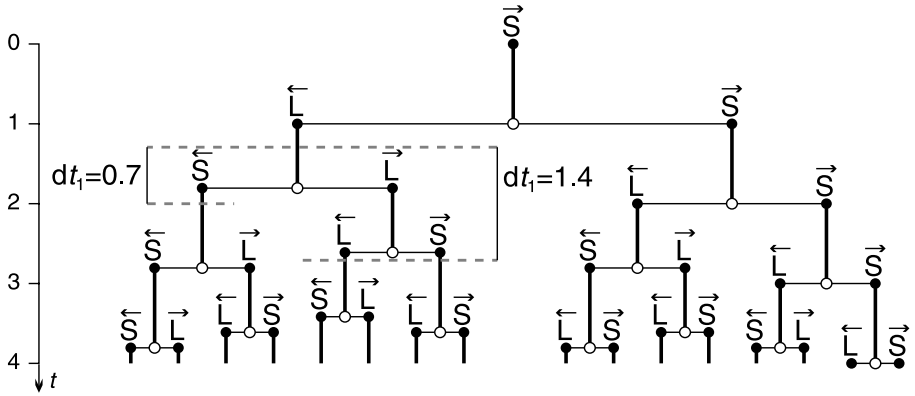


Figure 5: Development of an *Anabaena* filament considered in continuous time. Each cell divides at most once within any interval of duration $dt_1 =< 0.8$, but may divide more than once within an interval $dt_2 \geq 0.8$. Sample intervals $dt_1 = 0.7$ and $dt_2 = 1.4$ are shown.

the time increment dt is sufficiently small, for example $dt = 0.7$, each cell undergoes at most one division within an interval $(t, t+dt]$. This is correctly captured by L-system 5, according to which a cell either does not divide within a derivation step at all, or divides once. On the other hand, if the time increment dt is too large, for example $dt = 1.4$, some cells may undergo two divisions within an interval $(t, t + dt]$. This possibility is not accounted

for in L-system 5, which instead produces cells that exceed the age at which they should have divided.

6 Decomposition rules

Time steps of a duration longer than the minimum lifetime of a cell could be handled using productions that create more than two cells in a single derivation step. A more general and concise solution, however, is to assume that a cell will divide recursively, as long as the age of the descendants is greater than the division age t_{div} (Figure 6). Recursive application of productions is a concept characteristic of Chomsky grammars rather than L-systems. We combine these concepts by distinguishing two types of productions: L-system productions, applied in the breadth-first fashion, and Chomsky productions, applied recursively [24]. In the `cpfg` language, we refer to Chomsky productions as *decomposition rules*, and separate them from the ordinary L-system productions with the keyword `decomposition` [22]. A derivation step consists of the application of L-system productions to all modules in the string, followed by the recursive application of decomposition rules. A model of *Anabaena* development using decomposition rules is given by L-system 6.

```
/* L-system 6 */
#define LEFT -1
#define RIGHT 1
#define t_div 1.0       /* cell age at division */
#define t_s 0.0         /* initial age - short cell */
#define t_l 0.2         /* initial age - long cell */
#define dt 0.7          /* time increment per step */
Axiom: M(0,RIGHT)
M(t,p) --> M(t+dt,p)
decomposition:
M(t,p) : t>=t_div && p == LEFT -->
        M(t-t_div+t_s,LEFT) M(t-t_div+t_l,RIGHT)
M(t,p) : t>=t_div && p == RIGHT -->
        M(t-t_div+t_l,LEFT) M(t-t_div+t_s,RIGHT)
```

The use of decomposition rules improved the structure of L-system 6 with respect to L-system 5, by distinguishing between an ordinary L-system production, which advances time, and decomposition rules, which describe the

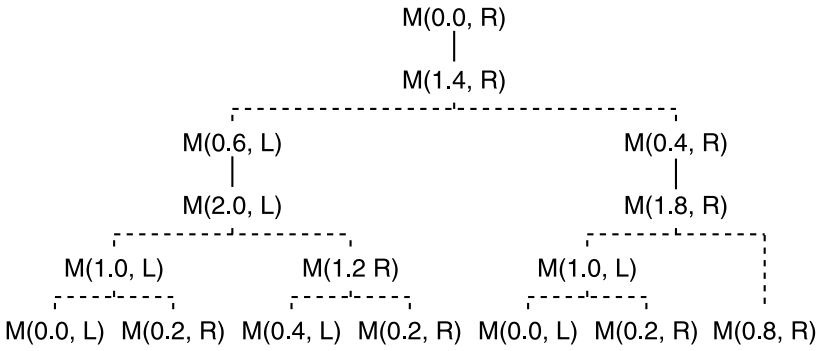


Figure 6: Development of an *Anabaena* filament simulated using L-system 6 with time step $dt = 1.4$. The application of the time-advancing L-system production (thick line) is followed by the recursive application of decomposition rules (thin lines).

structure of the descendants of each module. The underlying logical distinction between the relations "produces over time" and "is a part of" was formalized by Woodger and Tarski [29], and reviewed by Lindenmayer [14]. Figure 6 shows that L-system 6 works correctly for time increments exceeding the lifetime of a cell.

If we are only interested in the final state of the model at some time T, we can delegate all computation to the recursive application of decomposition rules by setting $dt = T$. This mode of operation is closely related to *timed L-systems*, defined in [23]. Timed L-systems make it possible to find the state of the developing structure at any time T, and consequently were the first variant of L-systems used to create "smooth" animations of plant development. As a limiting factor, the arbitrarily large time steps and recursive evaluation of L-systems are only possible in the deterministic context-free case, where components of the structure do not communicate with each other during the development, and always create the same lineage.

7 Interpretation rules

Up to now, we have not been concerned with the visualization of the models. This is consistent with the original definition of the L-system formalism, which only described ordering of cells in the filament, i.e., its topology. Nevertheless, in order to present generated models graphically, we need to

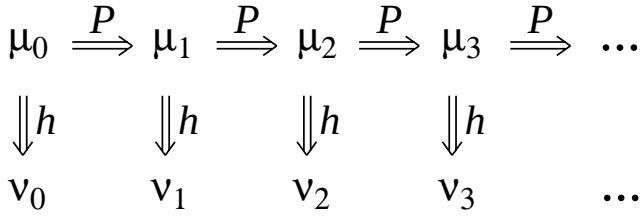


Figure 7: Generation of a developmental sequence using a `cpfg` model with interpretation rules. The progression of strings $\mu_0, \mu_1, \mu_2, \ldots$ results from the derivation steps $\stackrel{P}{\Longrightarrow}$ defined by productions and decomposition rules. The interpretation rules $\stackrel{h}{\Longrightarrow}$ map strings μ_i into the final strings ν_i that contain the graphical information.

assign a geometric interpretation to the modules. The problem is that the types of modules in terms of which a model is formulated and conceptualized (e.g., cells, apices, or leaves) is often very different from graphical primitives that are convenient to describe its geometry (e.g., lines, polygons, parametric surfaces). This problem was first addressed by Kurth [9], who introduced the notion of *interpretation rules*.

Interpretation rules provide a mechanism for complementing models with the geometric information needed for visualization purposes. They do not affect the sequence of strings $\mu_0, \mu_1, \mu_2, \ldots$ derived by productions and decomposition rules, but make it possible to temporarily replace modules in the derived strings by other modules or sequences of modules (Figure 7), which may have a predefined graphical interpretation. Formally, the interpretation rules are related to the notion of string *homomorphisms*, which have been studied in the mathematical theory of L-systems [6, 26]. The interpretation rules extend the concept of homomorphism, because they may operate on symbols with parameters, and can be applied in hierarchical and recursive manners. In that sense, they resemble decomposition rules.

In the `cpfg` language, the interpretation rules are specified using the same syntax as context-free productions and decomposition rules, following the keyword `homomorphism`. To see how they work, let us assume that the modeling program supports predefined modules `L(length,width)` and `R(length,width)`, which draw a round-cornered rectangles of given length and width. In addition, each module incorporates an arrow, pointing to the left or right, respectively. (In the actual `cpfg` implementation, modules `L` and `R` are defined in terms of more fundamental primitives, but this is

irrelevant to the basic concept.) We thus can define a complete L-system model of a vegetative segment of the Anabaena filament, by extending L-systems 6 with interpretation rules.

```
/* L-system 7 */
#define LEFT -1
#define RIGHT 1
#define t_div 1.0       /* cell age at division */
#define t_s 0.0         /* initial age - short cell */
#define t_l 0.2         /* initial age - long cell */
#define a 2.1673        /* exponential growth base */
#define WID 1.0         /* cell width */
#define dt 0.7          /* time increment per step */
Axiom: M(0,RIGHT)
M(t,p) --> M(t+dt,p)
decomposition:
M(t,p) : t>=t_div && p == LEFT -->
        M(t-t_div+t_s,LEFT) M(t-t_div+t_l,RIGHT)
M(t,p) : t>=t_div && p == RIGHT -->
        M(t-t_div+t_l,LEFT) M(t-t_div+t_s,RIGHT)
homomorphism:
M(t,p) : p==LEFT --> L(a^t,WID)
M(t,p) : p==RIGHT --> R(a^t,WID)
```

8 Geometric continuity

L-system 7 illustrates a nontrivial detail of the geometric interpretation of L-systems that can operate with arbitrarily small time steps: the need of preserving geometric continuity of the growing structure over time. In this specific example, we have assumed that the length of *Anabaena* cells increases exponentially with time: $l(t) = a^t$. Since the exponential function is continuous, the length of cells, and therefore the filament, will also be continuous functions of time in the intervals between cell divisions. In order to maintain the continuity during cell divisions as well, the length of a mother cell immediately before its division must be equal to the combined length of the daughter cells immediately after the division:

$$a^{t_{div}} = a^{t_s} + a^{t_l}$$

Given age values $t_{div} = 1.0$, $t_s = 0.0$ and $t_l = 0.2$, we obtain the basis a of the exponential growth by numerically solving the above equation: $a \approx 2.1673$. Thus, the value of a is a logical consequence of the timing of cell division and the exponential character of cell growth. The cell width WID has been set to $a^{t_s} = 1.0$, so that the smallest cells in the filament appear round. For a more general discussion of the problem of maintaining geometric continuity see [23].

In the above example, each cell keeps track of its age, and divides when it reaches the threshold age. An alternative approach is to use a cell's size itself as the independent variable. In this case, the continuity of the filament length is preserved in a more straightforward fashion, by explicitly partitioning the mother cell in the ratio $SMALLER : (1 - SMALLER)$. The resulting L-system is given below.

```
/* L-system 8 */
#define LEFT -1
#define RIGHT 1
#define x_div 2.1673    /* cell length at division */
#define SMALLER 0.4614  /* fraction of dividing cell length */
#define gr 1.7185       /* growth rate per simulation step */
#define WID 1.0         /* cell width */
Axiom: M(1.0,RIGHT)
M(x,p) --> M(x*gr,p)
decomposition:
M(x,p) : x>=x_div && p == LEFT
        --> M(x*SMALLER,LEFT) M(x*(1-SMALLER),RIGHT)
M(x,p) : x>=x_div && p == RIGHT
        --> M(x*(1-SMALLER),LEFT) M(x*SMALLER,RIGHT)
homomorphism:
M(x,p) : p==LEFT --> L(x,WID)
M(x,p) : p==RIGHT --> R(x,WID)
```

The constant values in L-system 8 have been chosen in terms of the constants of L-system 7, such that both L-systems produce the same developmental sequence. Specifically, $x_{div} = a^{t_{div}}$, $SMALLER = a^{t_s}/x_{div}$ and $gr = a^{dt}$.

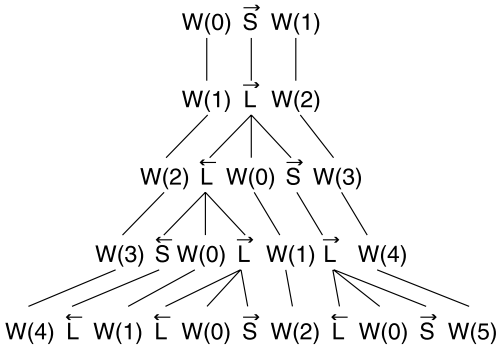


Figure 8: Developmental sequence of *Anabaena* filament (basic model, as in Figure 1), with explicitly represented cell walls. The parameter of cell walls indicated their age.

9 Context-sensitive L-systems

All models considered so far were united by one common tread: the fate of each module was determined by this module itself. A developmental process that proceeds this way is said to be controlled by *lineage* [12, 23, 20]. In reality, however, the fate of modules may also be controlled by *interactions* with their neighbors. In order to give a simple example of this fundamental concept, let us consider a variant of the *Anabaena* filament model, in which asymmetric cell divisions are controlled by properties of cell walls, rather than an attribute inherent in the individual cells. This variant of the model provides an insight into a plausible mechanism of polarity determination in nature.

The modified model treats the filament as a sequence of cells C separated by explicitly represented walls W. Similarly to cells, walls carry the age attribute. Figure 8 shows a developmental sequence corresponding to that of the basic *Anabaena* model (Figure 1). This sequence reveals that, during the asymmetric cell division, the shorter cell is always formed on the side of the older wall. It can be proven that this relationship between cell polarity and the age the walls that separate it from its neighbors also holds true for *Anabaena* models with improved timing (L-systems 2 to 8).

We now use this observation to a *context-sensitive* L-system [11] model, in which the asymmetry of cell division is controlled by the age of the neighboring walls (the context of the cell), rather than cell type or parameters transferred from mother to daughter cell by lineage. Context-sensitive para-

metric have the form

$$left\_context < predecessor > right\_context : condition \longrightarrow successor,$$

where $left\_context$ and $right\_context$ are strings of zero, one or more modules to the left and to the right of the strict *predecessor* [5, 23, 20]. During the derivation, the context modules may affect the applicability and outcome of a production application, but only the strict predecessor module is replaced by the successor. Using this notation, a context-sensitive variant of the *Anabaena* model based on L-system 5 is given below:

```
/* L-system 9 */
#define LEFT -1
#define RIGHT 1
#define t_div 1.0       /* cell age at division */
#define t_s 0.0         /* initial age - short cell */
#define t_l 0.2         /* initial age - long cell */
#define dt 0.7          /* time increment per step */
Axiom: W(0)M(0)W(dt)
W(t) --> W(t+dt)
M(t) : t<t_div --> M(t+dt)
W(tl) < M(t,p) > W(tr) : t>=t_div && tl<tr -->
        M(t-t_div+t_s+dt) W(t-t_div) M(t-t_div+t_l+dt)
W(tl) < M(t,p) > W(tr) : t>=t_div && tr>tl -->
        M(t-t_div+t_l+dt) W(t-t_div) M(t-t_div+t_s+dt)
```

In this model we have not used decomposition rules, and, consequently, the model does not operate properly for large time steps dt. The recursive cell division implemented using decomposition rules in L-systems 6 to 8 is not possible here, because it does not provide context information needed for cell divisions. More advanced time management techniques than the simple time slicing implemented by L-system 9 have been developed in the scope of simulation theory [8] and can be applied to L-systems as well [21], but their discussion is outside the scope of these introductory notes.

10 Conclusions

We have outlined basic types of L-systems introduced by Lindenmayer, context-free (DOL) and context-sensitive L-systems, and some of their extensions useful for model construction. These include the addition of attributes to L-system symbols (parametric L-systems), and decomposition

and interpretation rules. We have illustrated these concepts with a sequence of models of the vegetative segment of an *Anabaena catenula* filament. These examples also present a method for constructing developmental models that operate in arbitrarily small time steps.

References

[1] D. G. Adams. Heterocyst formation in cyanobacteria. *Current Opinoin in Microbiology*, 3:618–624, 2000.

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

[3] K. A. Erstad. L-systems, twining plants, Lisp. Master's thesis, University of Bergen, January 2002.

[4] J. W. Golden and H.-S. Yoon. Heterocyst formation in *Anabaena*. *Current Opinion in Microbiology*, 1:623–629, 1998.

[5] J. S. Hanan. *Parametric L-systems and their application to the modelling and visualization of plants*. PhD thesis, University of Regina, June 1992.

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

[7] R. Karwowski. *Improving the process of plant modeling: The L+C modeling language*. PhD thesis, University of Calgary, October 2002.

[8] W. Kreutzer. *System simulation: Programming styles and languages*. Addison-Wesley, Sydney, 1986.

[9] W. Kurth. *Growth grammar interpreter GROGRA 2.4: A software tool for the 3-dimensional interpretation of stochastic, sensitive growth grammars in the context of plant modeling. Introduction and reference manual*. Forschungszentrum Waldökosysteme der Universität Göttingen, Göttingen, 1994.

[10] A. Lindenmayer. Developmental systems and languages in their biological context. In G. T. Herman and G. Rozenberg, *Developmental systems and languages*. North-Holland, Amsterdam, 1975, pp. 1 – 40.

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

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

[13] 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.

[14] A. Lindenmayer. Theories and observations of developmental biology. In R. E. Butts and J. Hintikka, editors, *Foundational problems in special sciences*, pages 103–118. D. Reidel Publ. Co, Dordrecht-Holland, 1977.

[15] A. Lindenmayer. Algorithms for plant morphogenesis. In R. Sattler, editor, *Theoretical plant morphology*, pages 37–81. Leiden University Press, The Hague, 1978.

[16] 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.

[17] H. B. Lück and J. Lück. Cell number and cell size in filamentous organisms in relation to ancestrally and positionally dependent generation times. In A. Lindenmayer and G. Rozenberg, editors, *Automata, languages, development*, pages 109–124. North-Holland, Amsterdam, 1976.

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

[19] J. L. Peterson. *Petri net theory and the modeling of systems.* Prentice Hall, Englewood Cliffs, 1981.

[20] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Měch. Visual models of plant development. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages,* Vol. III*: Beyond words*, pages 535–597. Springer, Berlin, 1997.

[21] P. Prusinkiewicz, M. Hammel, and E. Mjolsness. Animation of plant development. Proceedings of SIGGRAPH 93 (Anaheim, California, August 1–6, 1993). ACM SIGGRAPH, New York, 1993, pp. 351–360.

[22] P. Prusinkiewicz, J. Hanan, and R. Měch. An L-system-based plant modeling language. In M. Nagl, A. Schürr, and M. Münch, editors, *Applications of graph transformations with industrial relevance*, Lecture Notes in Computer Science 1779, pages 395–410. Springer-Verlag, Berlin, 2000.

[23] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[24] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane. The use of positional information in the modeling of plants. Proceedings of SIGGRAPH 2001 (Los Angeles, California, August 12–17, 2001). ACM SIGGRAPH, New York, 2001, pp. 289–300.

[25] P. Prusinkiewicz and W. Remphrey. Characterization of architectural tree models using L-systems and Petri nets. In M. Lebrecque, editor, *L'Arbre - The Tree 2000. Papers presented at the 4th Internatinal Symposium on the Tree at the Montreal Botanical Garden, Aug. 20-25, 2000.*, pages 177–186. Isabelle Quentin and Institut de Recherche en Biologie Végétale, Montreal, 2001.

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

[27] G. Rozenberg and A. Salomaa. *Handbook of formal languages*. Springer, Berlin, 1997.

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

[29] J. H. Woodger. *The axiomatic method in biology*. University Press, Cambridge, 1937. With appendices by A. Tarski and W. F. Floyd.

L-systems: from the Theory to Visual Models of Plants

Przemyslaw Prusinkiewicz[1], Jim Hanan[2], Mark Hammel[1] and Radomir Mech[1]

[1] Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4
[2] CSIRO — Cooperative Research Centre for Tropical Pest Management, Brisbane, Queensland, Australia

1 Introduction

In 1968, Aristid Lindenmayer introduced a formalism for simulating the development of multicellular organisms, subsequently named L-systems [18]. This formalism was closely related to abstract automata and formal languages, and attracted the immediate interest of theoretical computer scientists. The vigorous development of the mathematical theory of L-systems was followed by its applications to the modeling of plants. These applications gained momentum after 1984, when Smith introduced state-of-the art computer graphics techniques to visualize the structures and processes being modeled [52]. Smith also attracted attention to the phenomenon of *data-base amplification*, or the possibility of generating complex structures from compact data sets, which is inherent in L-systems and forms the cornerstone of L-system applications to image synthesis. Subsequent developments (presented here from our personal perspective, without covering the fast-growing array of contributions from many other researchers) included:

- introduction of turtle interpretation of L-systems [32, 53] and refinement of a programming language based on L-systems [11, 43], which facilitated specification of the models for simulation purposes and promoted the use of L-systems as a language for describing models in publications;
- recognition of the fractal character of structures generated by L-systems, which related them to the dynamically developing science of fractals [32, 43, 38];
- increased interest in the application of computer simulations to the understanding of living processes and structures, related to the emergence of the field of Artificial Life;
- extension of the range of phenomena that can be modeled using L-systems, including, most recently, incorporation of environmental factors into the models [30, 41];
- increased understanding of the modeling process, providing a methodology for constructing models according to biological observations and measurements [45, 48].

In this paper, we revisit basic mechanisms that control plant development: lineage (cellular descent), captured by the class of context free L-systems, and endogenous interaction (transfer of information between neighboring modules in the structure), captured by context-sensitive L-systems (*c.f.* [22]). Within this framework, we present several models that have been developed after the survey of L-systems in [43].

The original version of this apper appeared in M. T. Michalewicz (Ed.): *Plants to Ecosystems. Advances in Computational Life Sciences*, CSIRO, Collingwood, Australia 1997, pp. 1–27.

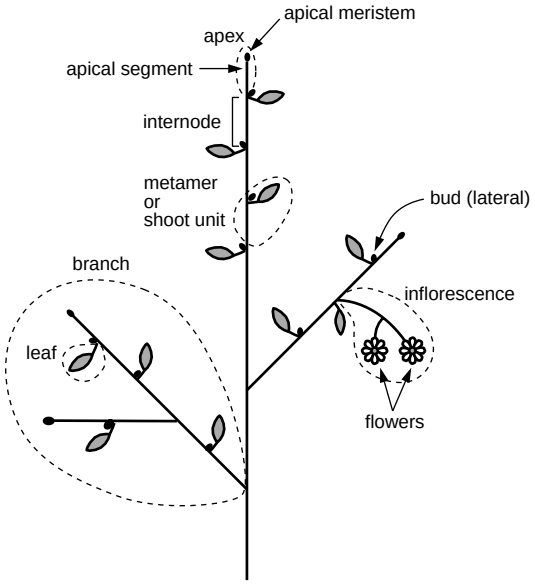


Figure 1: Selected modules and groups of modules (encircled with dashed lines) used to describe plants

2 The modular structure of plants

L-systems were originally introduced to model the development of simple *multicellular* organisms (for example, algae) in terms of division, growth, and death of individual cells [18, 19]. The range of L-system applications has subsequently been extended to higher plants and complex branching structures, in particular inflorescences [8, 9], described as configurations of modules in space. In the context of L-systems, the term *module* denotes any discrete constructional unit that is repeated as the plant develops, for example an internode, an apex, a flower, or a branch (Figure 1) [2, 12, 55]. The goal of modeling at the modular level is to describe the development of a plant as a whole, and in particular the emergence of plant shape, as the integration of the development of individual units.

3 Plant development as a rewriting process

The essence of development at the modular level can be conveniently captured by a parallel *rewriting system* that replaces individual *parent*, *mother*, or *ancestor* modules by configurations of *child*, *daughter*, or *descendant* modules. All modules belong to a finite *alphabet of module types*, thus the behavior of an arbitrarily large configuration of modules can be specified using a finite set of *rewriting rules* or *productions*. In the simplest case of *context-free*

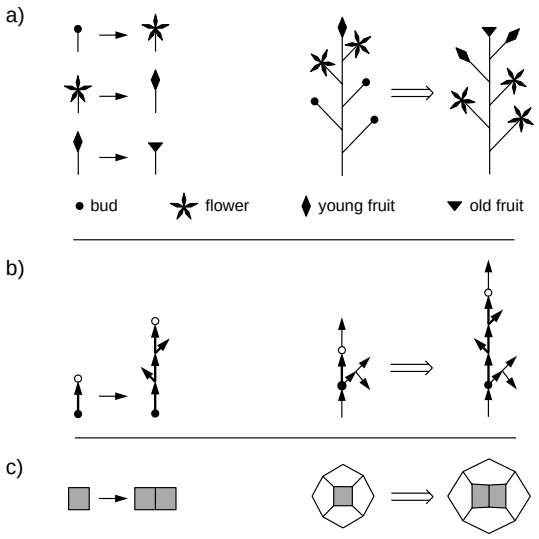


Figure 2: Examples of production specification and application: (a) development of a flower, (b) development of a branch, and (c) cell division.

rewriting, a production consists of a single module called the *predecessor* or the *left-hand side*, and a configuration of zero, one, or more modules called the *successor* or the *right-hand side*. A production p with the predecessor matching a given mother module can be applied by deleting this module from the rewritten structure and inserting the daughter modules specified by the production's successor.

Three examples of production application are shown in Figure 2. In case (a), modules located at the extremities of a branching structure are replaced without affecting the remainder of the structure. In case (b), productions that replace internodes divide the branching structure into a lower part (below the internode) and an upper part. The position of the upper part is adjusted to accommodate the insertion of the successor modules, but the shape and size of both the lower and upper part are not changed. Finally, in case (c), the rewritten structures are represented by graphs with cycles. The size and shape of the production successor does not exactly match the size and shape of the predecessor, and the geometry of the predecessor and the embedding structure had to be adjusted to accommodate the successor. The last case is most complex, since the application of a local rewriting rule may lead to a global change of the structure's geometry. Developmental models of cellular layers operating in this manner have been presented in [43, 4, 5, 7]. In this paper we focus on the rewriting of branching structures corresponding to cases (a) and (b).

Productions may be applied *sequentially*, to one module at a time, or they may be applied *in parallel*, with all modules being rewritten simultaneously in every *derivation step*. Parallel rewriting is more appropriate for the modeling of biological development, since development takes place simultaneously in all parts of an organism. A derivation step then corresponds to the progress of time over some interval. A sequence of structures obtained in consecutive derivation steps from a predefined *initial structure* or *axiom* is called a *developmental sequence*. It can be viewed as the result of a *discrete-time simulation* of development.

For example, Figure 3 illustrates the development of a stylized compound leaf including two module types, the *apices* (represented by

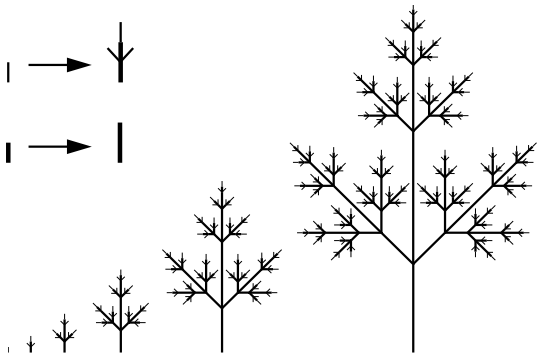


Figure 3: Developmental model of a compound leaf, modeled as a configuration of apices and internodes

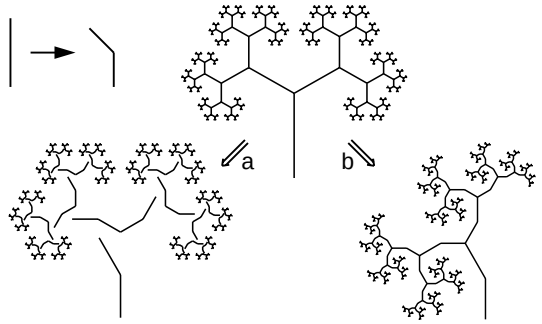


Figure 4: A comparison of the Koch construction (a) with a rewriting system preserving the branching topology of the modeled structures (b). The same production is applied in both cases, but the rules for incorporating the successor into the structure are different.

thin lines) and the *internodes* (thick lines). An apex yields a structure that consists of two internodes, two lateral apices, and a replica of the main apex. An internode elongates by a constant scaling factor. In spite of the simplicity of these rules, an intricate branching structure develops from a single apex over a number of derivation steps.

It is interesting to contrast simulation of development using rewriting rules with the well known Koch construction for generating fractals [29, page 39]. The essence of the Koch construction is the replacement of straight line segments by sets of lines. Their positions, orientations, and scales are determined by the position, orientation, and scale of the segment being replaced (Figure 4a). In contrast, in models of plants, the position and orientation of each module is determined by the chain of modules beginning at the base of the structure and extending to the module under consideration. For example, when the internodes bend, the subtended branches are rotated and displaced to maintain the connectivity of the structure (Figure 4b). Thus, development is simulated as a parallel application of productions, followed by a sequential connection of the child structures.

Rewriting processes maintaining the connectivity of branching structures can be defined directly in the geometric domain, but a more convenient approach is to express the generating rules and the resulting structures symbolically, using a string notation. A sequential geometric interpretation of these strings from the left (plant base) to right (branch extremities) automatically captures proper positioning of the higher branches on the lower ones. The rewriting

of branching structures in the string domain is the cornerstone of L-systems.

The basic notions of the theory of L-systems have been presented in many survey papers [22, 20, 21, 24, 25, 26] and books [43, 38, 13, 50, 51]. Consequently, we only describe parametric L-systems, which are a particularly convenient programming tool for expressing models of plant development. Our presentation closely follows the formalization introduced in [43, 39] (see also [11, 40]).

4 Parametric L-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 $M = V \times \Re^*$, where $\Re^*$ is the set of all finite sequences of parameters. The set of all strings of modules and the set of all nonempty strings are denoted by $M^* = (V \times \Re^*)^*$ and $M^+ = (V \times \Re^*)^+$, respectively.

The real-valued *actual* parameters appearing in the words have a counterpart in the *formal* parameters, which may occur in the specification of L-system productions. 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 arithmetic operators $+$, $-$, $*$, $/$; the exponentiation operator $\wedge$, the relational operators $<$, $<=$, $>$, $>=$, $==$; the logical operators !, &&, || (not, and, or); and parentheses $()$. The expressions can also include calls to standard mathematical functions, such a natural logarithm, sine, floor, and functions returning random variables. The operation symbols and the rules for constructing syntactically correct expressions are the same as in the C programming language [17]. For clarity of presentation, however, we sometimes use Greek letters and symbols with subscripts in print. 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 sets of all correctly constructed logical and arithmetic expressions with parameters from Σ are noted $\mathcal{C}(\Sigma)$ and $\mathcal{E}(\Sigma)$.

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

- V is 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*,
- $P \subset (V \times \Sigma^*) \times \mathcal{C}(\Sigma) \times (V \times \mathcal{E}(\Sigma)^*)^*$ is a finite *set of productions*.

The symbols : and $\rightarrow$ are used to separate the three components of a production: the *predecessor*, the *condition*, and the *successor*. Thus, a production has the format

$$pred : cond \rightarrow succ.$$

For example, a production with predecessor $A(t)$, condition $t > 5$ and successor $B(t+1)CD(t \wedge 0.5, t-2)$ is written as

$$A(t) : t > 5 \rightarrow B(t+1)CD(t \wedge 0.5, t-2). \qquad (1)$$

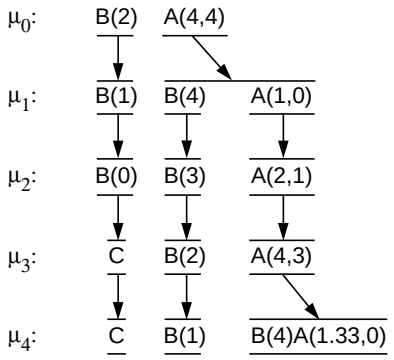


Figure 5: The initial sequence of strings generated by the parametric L-system specified in equation (2)

A production in a 0L-system *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 the formal parameters in the production.

A matching production can be *applied* to the module, creating a string of modules specified by the production successor. The actual parameter values are substituted for the formal parameters according to their position. For example, production (1) above matches a module $A(9)$, since the letter A in the module is the same as in the production predecessor, there is one actual parameter in the module $A(9)$ and one formal parameter in the predecessor $A(t)$, and the logical expression $t > 5$ is true for t equal to 9. The result of the application of this production is a parametric word $B(10)CD(3, 7)$.

If a module a produces a parametric word χ as the result of a production application in an 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 L-system is given below.

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

It is assumed that a module replaces itself if no matching production is found in the set P. The words obtained in the first few derivation steps are shown in Figure 5.

Productions in parametric 0L-systems are *context-free*, *i.e.*, applicable regardless of the context in which the predecessor appears. A *context-sensitive* extension is necessary to model information exchange between neighboring modules. In general, a context-sensitive production has the format

$$lc < pred > rc \quad : cond \rightarrow succ,$$

where symbols $<$ and $>$ separate the three components of the predecessor: a string of modules without brackets lc called the *left context*, a module $pred$ called the *strict predecessor*, and a well-nested bracketed string of modules rc called the *right context*. The remaining components of the production are the condition $cond$ and the successor $succ$, defined as for parametric 0L-systems.

A sample context-sensitive production is given below:

$$A(x) < B(y) > C(z) : x + y + z > 10 \rightarrow E((x+y)/2)F((y+z)/2). \quad (3)$$

The left context is separated from the strict predecessor by the symbol $<$. Similarly, the strict predecessor is separated from the right context by the symbol $>$. Production 3 can be applied to the module $B(5)$ that appears in a parametric word

$$\cdots A(4)B(5)C(6)\cdots \quad (4)$$

since the sequence of letters A, B, C in the production and in parametric word (4) are the same, the numbers of formal parameters and actual parameters 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 $A(4)$ and $C(6)$ will be replaced by other productions in the same derivation step.

Productions in 2L-systems use context on both sides of the strict predecessor. 1L-systems are a special case of 2L-systems in which context appears only on one side of the productions.

When no production explicitly listed as a member of the production set P matches a module in the rewritten string, we assume that an appropriate *identity production* belongs to P and replaces this module by itself. Under this assumption, a parametric L-system $G = \langle V, \Sigma, \omega, P \rangle$ is called *deterministic* if and only if for each module $A(t_1, t_2, \ldots, t_n) \in V \times \Re^*$ the production set includes exactly one matching production. Within this paper we only consider deterministic L-systems.

5 The turtle interpretation of L-systems

Strings generated by L-systems may be interpreted geometrically in many different ways. Below we outline the *turtle interpretation* of L-systems, introduced by Szilard and Quinton [53], and extended by Prusinkiewicz [32, 33] and Hanan [11, 10]. A tutorial exposition is included in [43], and subsequent results are presented in [11]. The summary below is based on [43, 39, 33, 15].

After a string has been generated by an L-system, it is scanned sequentially from left to right, and the consecutive symbols are interpreted as commands that maneuver a LOGO-style turtle [1, 31] in three dimensions. The turtle is represented by its *state*, which consists of turtle *position* and *orientation* in the Cartesian coordinate system, as well as various attribute values, such as current *color* and *line width*. The position is defined by a vector $\vec{P}$, and the orientation is defined by three vectors $\vec{H}$, $\vec{L}$, and $\vec{U}$, indicating the turtle's *heading* and the directions to the *left* and *up* (Figure 6a). These vectors have unit length, are perpendicular to each other, and satisfy the equation $\vec{H} \times \vec{L} = \vec{U}$. Rotations of the turtle are expressed by the equation:

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

where $\mathbf{R}$ is a 3×3 rotation matrix [6]. Changes in the turtle's state are caused by interpretation of specific symbols, each of which may be followed by parameters. If one or more parameters are present,

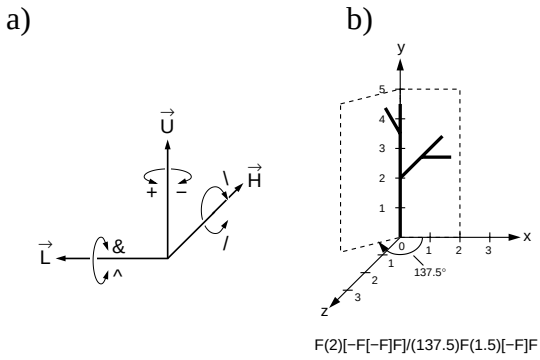


Figure 6: a) Controlling the turtle in three dimensions. b) Example of the turtle interpretation of a string.

the value of the first parameter affects the turtle's state. If the symbol is not followed by any parameter, default values specified outside the L-system are used. The following list specifies the basic set of symbols interpreted by the turtle.

Symbols that cause the turtle to move and draw

$F(s), G(s)$ Move forward a step of length s and draw a line segment from the original to the new position of the turtle.

$f(s), g(s)$ Move forward a step of length s without drawing a line.

$@O(r)$ Draw a sphere of radius r at the current position.

Symbols that control turtle orientation in space (Figure 6a)

$+(\theta)$ Turn left by angle θ around the $\vec{U}$ axis.

$-(\theta)$ Turn right by angle θ around the $\vec{U}$ axis.

$\&(\theta)$ Pitch down by angle θ around the $\vec{L}$ axis.

$\wedge(\theta)$ Pitch up by angle θ around the $\vec{L}$ axis.

$/(\theta)$ Roll left by angle θ around the $\vec{H}$ axis.

$\backslash(\theta)$ Roll right by angle θ around the $\vec{H}$ axis.

$|$ Turn $180°$ around the $\vec{U}$ axis. This is equivalent to $+(180)$ or $-(180)$.

Symbols for modeling structures with branches

$[$ Push the current state of the turtle (position, orientation and drawing attributes) 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 and orientation of the turtle are changed.

Symbols for creating and incorporating surfaces

$\{$ Start saving the subsequent positions of the turtle as the vertices of a polygon to be filled.

$\}$ Fill the saved polygon.

$\sim X(s)$ Draw the surface identified by symbol X, scaled by s, at the turtle's current location and orientation. Such a surface is usually defined as a bicubic patch [33, 10].

Symbols that change the drawing attributes

$\#(w)$ Set line width to w, or increase the value of the current line width by the default width increment if no parameter is given.

$!(w)$ Set line width to w, or decrease the value of the current line width by the default width decrement if no parameter is given.

$;(n)$ Set the index of the color map to n, or increase the value of the current index by the default colour increment if no parameter is given.

$,(n)$ Set the index of the color map to n, or decrease the value of the current index by the default colour decrement if no parameter is given.

A sample string and its interpretation are shown in Figure 6b. The default length of lines represented by symbols F without a parameter is 1, and the default magnitude of the angles represented by symbols $+$ and $-$ is $45°$.

6 Examples of parametric D0L-system models

This section presents selected examples that illustrate the operation of deterministic 0L-systems (D0L-systems) with turtle interpretation and their application to the modeling of plants. Many other examples are included in [11, 43, 39].

6.1 Fractal generation

Fractal curves provide a convenient means for illustrating the basic principle of L-system operation [32, 43, 38, 44]. For example, the following L-system generates the well-known snowflake curve [29, 54].

$$\begin{array}{lll} \omega: & F(1)-(120)F(1)-(120)F(1) & \\ p_1: & F(s) \rightarrow F(s/3)+(60)F(s/3) & \\ & \quad -(120)F(s/3)+(60)F(s/3) & \end{array}$$

The axiom $F(1)-(120)F(1)-(120)F(1)$ draws an equilateral triangle, with edges of unit length. Production p_1 replaces each line segment with a polygonal shape, as shown at the top of Figure 7. Productions for symbols $+$ and $-$ are not listed, which means that the corresponding modules will be replaced by themselves during the derivation. The same effect could have been obtained by explicit inclusion of productions:

$$\begin{array}{ll} p_2: & +(a) \rightarrow +(a) \\ p_3: & -(a) \rightarrow -(a) \end{array}$$

The axiom and the figures obtained in the first three derivation steps are shown at the bottom of Figure 7.

6.2 Simulation of development

The next L-system generates the developmental sequence of the stylized compound leaf model presented in Figure 3.

$$\begin{array}{lllll} \omega: & !(1)F(1,1) & & & \\ p_1: & F(s) & \rightarrow & G(s)[-!(1)F(s)][+!(1)F(s)]G(s)!(1)F(s) \\ p_2: & G(s) & \rightarrow & G(2*s) \\ p_3: & !(w) & \rightarrow & !(3) \end{array}$$

The structure is built from two module types, apices F (represented by thin lines) and internodes G (thick lines). In both cases the parameter s determines the length of the line representing the module. An apex yields a structure that consists of two internodes, two lateral apices, and a replica of the main apex (production p_1). An internode elongates by a constant scaling factor (production p_2). Production p_3 is used to make the lines representing the internodes wider (3 units of width) than the lines representing the apices (1 unit). The branching angle associated with symbols $+$ and $-$ is set to $45°$ by a global variable outside the L-system.

6.3 Exploration of parameter space

Parametric L-systems provide a convenient mathematical framework for exploring the range of forms that can be captured by the same structural model with varying attributes (constants in the productions). Such *parameter space explorations* motivated some of the earliest computer simulations of biological structures: the models of sea shells devised by Raup and Michelson [46, 47] and the models of trees proposed by Honda [14] to study factors that determine overall tree shape. Parameter space exploration may reveal an unexpected richness of forms that can be produced by even the simplest models. For example, Figure 8 shows nine branching structures selected from a continuum generated by the following parametric D0L-system:

$$\begin{array}{lllll} \omega: & A(100, w_0) & & & \\ p_1: & A(s,w) & : s >= \mathit{min} & \rightarrow & !(w)F(s) \\ & & & & [+(\alpha_1)/(\varphi_1)A(s*r_1, w*q \wedge e)] \\ & & & & [+(\alpha_2)/(\varphi_2)A(s*r_2, w*(1-q) \wedge e)] \end{array}$$

The single non-identity production p_1 replaces apex A by an internode F and two new apices A. The angle values α_1, α_2, φ_1, and φ_2 determine the orientation of these apices with respect to the subtending internode. Parameters s and w specify internode length and

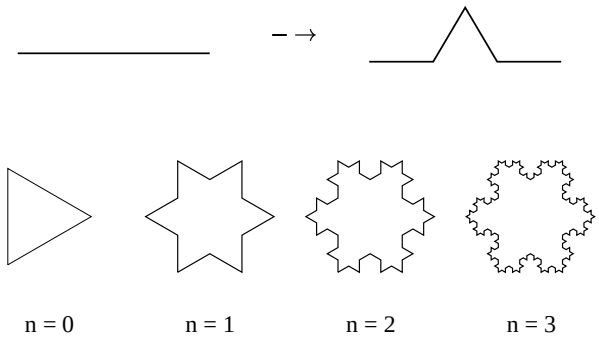


Figure 7: Visual interpretation of the production for the snowflake curve, and the curve after n = 0, 1, 2, and 3 derivation steps

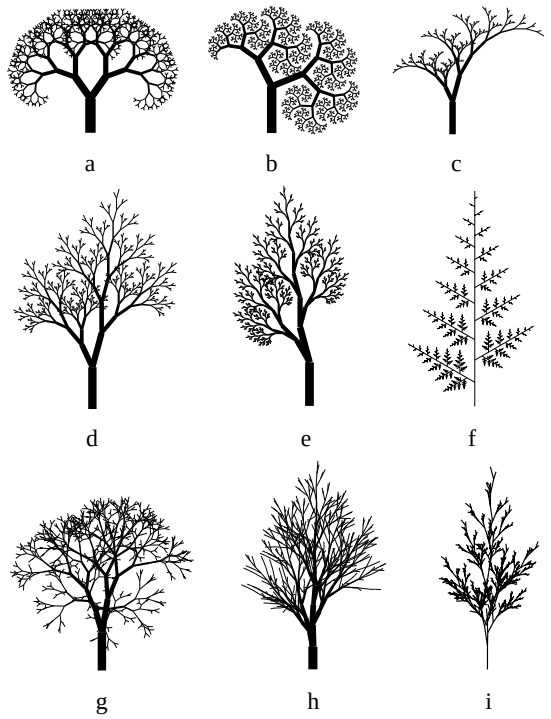


Figure 8: Sample structures generated by a parametric D0L-system with different values of constants

Table 1: The values of constants used to generate Figure 8

Figure	r_1	r_2	α_1	α_2	φ_1	φ_2	w_0	q	e	*min*	n
a	.75	.77	35	-35	0	0	30	.50	.40	0.0	10
b	.65	.71	27	-68	0	0	20	.53	.50	1.7	12
c	.50	.85	25	-15	180	0	20	.45	.50	0.5	9
d	.60	.85	25	-15	180	180	20	.45	.50	0.0	10
e	.58	.83	30	15	0	180	20	.40	.50	1.0	11
f	.92	.37	0	60	180	0	2	.50	.00	0.5	15
g	.80	.80	30	-30	137	137	30	.50	.50	0.0	10
h	.95	.75	5	-30	-90	90	40	.60	.45	25.0	12
i	.55	.95	-5	30	137	137	5	.40	.00	5.0	12

width. The constants r_1 and r_2 determine the gradual decrease in internode length that occurs while traversing the tree from its base towards the apices. The constants w_0, q, and e control the width of branches. The initial stem width is specified by w_0 in the second parameter of the axiom module A. For $e = 0.5$, the combined area of the descendant branches is equal to the area of the mother branch, as postulated by Leonardo da Vinci [29, page 156] (see also [28, pages 131–135]). The value q specifies the differences in width between descendant branches originating at the same vertex. Finally, the condition prevents formation of branches with length less then the threshold value *min*. The values of constants corresponding to each structure are collected in Table 1. The final column headed n indicates the number of derivation steps.

6.4 Modeling mesotonic and acrotonic structures

In spite of their apparent diversity, the structures generated by L-system of Section 6.3 share a common developmental pattern: in each derivation step, every apex gives rise to an internode terminated by a pair of new apices. This is a simple instance of *subapical branching*, a common developmental pattern in plants, in which new branches are initiated only near the apices of the existing axes. As a consequence of this pattern, the lower branches, being created first, have more time to develop than the branches further up, and a *basitonic* structure (more developed near the base than near the top) results (Figure 9a). In nature, however, one also finds *mesotonic* and *acrotonic* structures, in which the most developed branches are located near the middle or the top of the mother branch (Figures 9 b and c). As observed by Frijters and Lindenmayer [9], and formalized by Prusinkiewicz and Kari [42], arbitrarily large mesotonic and acrotonic structures cannot be generated by non-parametric deterministic 0L-systems with subapical branching. In contrast, *parametric* D0L-systems can generate such structures. For example, the following parametric D0L-system generates the mesotonic structure shown in Figure 9b.

$$\begin{array}{lll} \omega: & FA(0) & \\ p_1: & A(v) & \rightarrow [-FB(v)][+FB(v)]FA(v+1) \\ p_2: & B(v): v > 0 & \rightarrow FB(v-1) \end{array}$$

The axiom ω defines the initial structure as an internode F terminated by an apex A. In each derivation step, the apex A adds a new segment F to the main axis and initiates a pair of branches FB (production p_1). The value of parameter v assigned to the lateral apices B describes the maximum length to which each branch will grow (production p_2). This value is incremented acropetally (*i.e.*, in the ascending order of branches) by production p_1, yielding a sequence of branches of increasing length. This sequence is broken in the upper part of the structure, where the branches still grow. Consequently, the younger branches near the top are shorter than the older ones further down, and a mesotonic overall structure results.

A detailed discussion of the generation of mesotonic and acrotonic structures using a construct similar to parametric L-systems has been presented by Lück, Lück, and Bakkali [27].

6.5 The shedding of branches

The natural processes of plant development often involve shedding, or programmed removal of selected modules from the growing structure. In order to simulate shedding, Hanan [11] extended the formalism of L-systems with the *cut symbol* $\%$, which causes the removal of the remainder of the branch that follows it. For example, in the absence of other productions, the derivation step given below takes place:

$$a[b\%[cd]e[\%f]]g[h[\%i]j]k \Longrightarrow a[b]g[h[]j]k$$

A simple example of an L-system incorporating the cut symbol is given below:

$$\begin{array}{lllll} \omega: & A & & & \\ p_1: & A & & \rightarrow & F(1)[-X(3)B][+X(3)B]A \\ p_2: & B & & \rightarrow & F(1)B \\ p_3: & X(d) & : d > 0 & \rightarrow & X(d-1) \\ p_4: & X(d) & : d == 0 & \rightarrow & U\% \\ p_5: & U & & \rightarrow & F(0.3) \end{array}$$

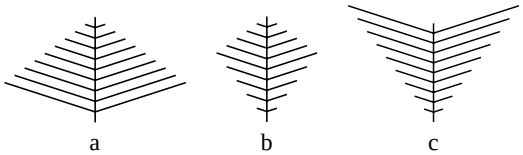


Figure 9: Schematic representation of a basitonic (a), mesotonic (b), and acrotonic (c) branching pattern. From [42].

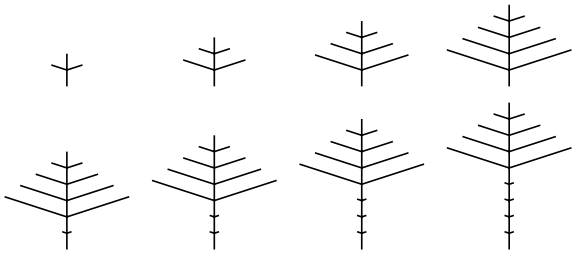


Figure 10: A developmental sequence generated by the L-system specified in Section 6.5. The images shown represent derivation steps 2 through 9.

According to production p_1, in each derivation step the apex of the main axis A produces an internode F of unit length and a pair of lateral apices B. Each apex B extends a branch by forming a succession of internodes F (production p_2). After three steps from branch initiation (controlled by production p_3), production p_4 inserts the cut symbol $\%$ and an auxiliary symbol U at the base of the branch. In the next step, the cut symbol removes the branch, while symbol U inserts a marker $F(0.3)$ indicating a "scar" left by the removed branch. The resulting developmental sequence is shown in Figure 10. The initial steps capture the growth of a basitonic structure. Beginning at derivation step 6, the oldest branches are shed, creating an impression of a tree crown of constant shape and size moving upwards. The crown is in a state of dynamic equilibrium: the addition of new branches and internodes at the apices is compensated by the loss of branches further down.

The state of dynamic equilibrium can be easily observed in the development of palms, where new leaves are created at the apex of the trunk while old leaves are shed at the base of the crown (Figure 11). Since both processes take place at the same rate, an adult palm carries an approximately constant number of leaves. This phenomenon has an interesting physiological explanation: palms are unable to gradually increase the diameter of their trunk over time, thus the flow of water and nutrients through the trunk can support only a crown of constant size.

7 Examples of context-sensitive L-system models

In this section we consider the propagation of control information through the structure of the developing plant (*endogenous information flow* [34]), which is captured by context-sensitive productions in the framework of L-systems. The conceptual elegance and expressive power of context-sensitive productions are among the most important assets of L-systems in modeling applications.

7.1 Development of a mesotonic structure

As outlined in Section 6.4, arbitrarily large mesotonic and acrotonic structures cannot be generated using deterministic 0L-systems without parameters [42]. The proposed mechanisms for modeling these structures can be divided into two categories: those using parameters to characterize the growth potential or vigor of individual apices, such as the L-system discussed in Section 6.4 and those postulating control of development by signals [8, 16]. The following L-system simulates the development of the mesotonic structure shown in Figure 12 using an acropetal (upward moving) signal.

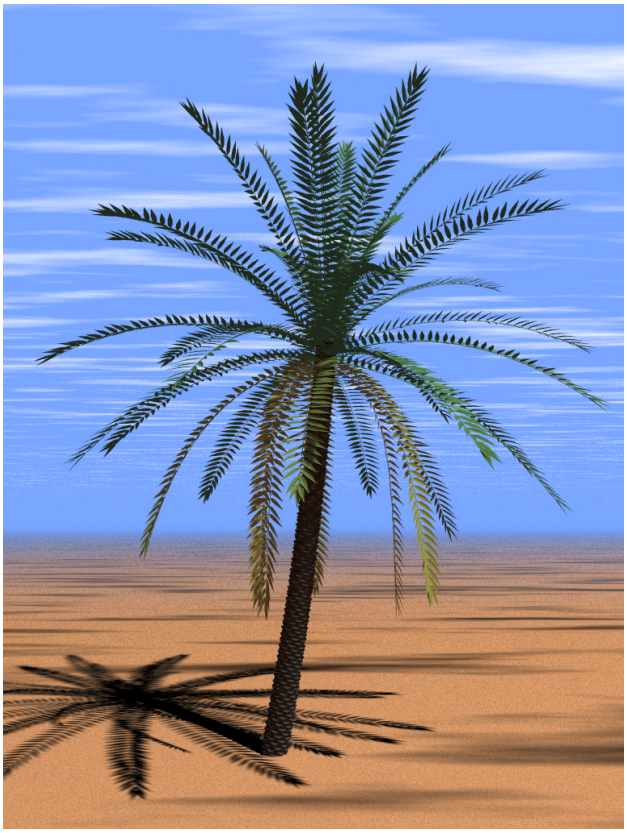


Figure 11: A model of the date palm (*Phoenix dactylifera*). This image was created using an L-system with the general structure specified in Section 6.5.

$$
\begin{array}{lll}
\#\text{define} & m \;\; 3 & /*\,\text{plastochron} - \text{main axis}\,*/ \\
\#\text{define} & n \;\; 4 & /*\,\text{plastochron} - \text{branc h}*/ \\
\#\text{define} & u \;\; 4 & /*\,\text{propagation rate} - \text{main axis}\,*/ \\
\#\text{define} & v \;\; 2 & /*\,\text{propagation rate} - \text{branc h}*/
\end{array}
$$

$$\text{ignore}: \quad +-/$$

$$
\begin{array}{ll}
\omega: & S(0)F(1,0)A(0) \\
p_1: & A(i) \quad : i < m-1 \rightarrow A(i+1) \\
p_2: & A(i) \quad : i == m-1 \rightarrow \\
 & \quad [+(60)F(1,1)B(0)]F(1,0)/(180)A(0) \\
p_3: & B(i) \quad : i < n-1 \rightarrow B(i+1) \\
p_4: & B(i) \quad : i == n-1 \rightarrow F(1,1)B(0) \\
p_5: & S(i) \quad : i < u+v \rightarrow S(i+1) \\
p_6: & S(i) \quad : i == u+v \rightarrow \varepsilon \\
p_7: & S(i) < F(l,o) : (o == 0)\&\&(i == u-1) \rightarrow \\
 & \quad \#F(l,o)!S(0) \\
p_8: & S(i) < F(l,o) : (o == 1)\&\&(i == v-1) \rightarrow \\
 & \quad \#F(l,o)!S(0) \\
p_9: & S(i) < B(j) \rightarrow \varepsilon
\end{array}
$$

The above L-system operates under the assumption that the context-sensitive production p_9 takes priority over p_3 or p_4. The ignore statement lists symbols that should not be taken for consideration for context-matching purposes. The axiom ω describes the initial structure as an internode F terminated by an apex A. A signal S

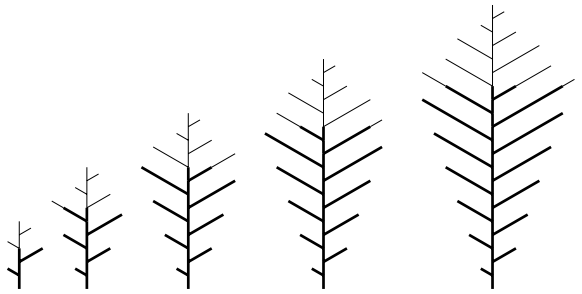


Figure 12: Development of a mesotonic branching structure controlled by an acropetal signal. Wide lines indicate the internodes reached by the signal. The stages shown correspond to derivation lengths 12, 24, 36, 48, and 60.

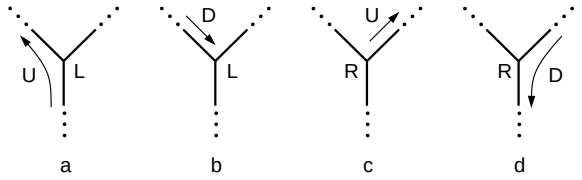


Figure 13: Insect's behavior at a branching point. An upward-moving insect U that approaches a branching point L is directed to the left daughter branch (a). A downward moving insect D that approaches a branching point marked L changes this marking to R, returns to state U, and enters the right branch (b and c). A downward moving insect D approaching a branching point R continues its downward motion (d).

is placed at the base of this structure. According to productions p_1 and p_2, the apex A periodically produces a lateral branch and adds an internode to the main axis. The period (called the *plastochron* of the main axis) is controlled by the constant m. Productions p_3 and p_4 describe the development of the lateral branches, where new segments F are added with plastochron n. Productions p_5 to p_8 describe the propagation of the signal through the structure. The signal propagation rate is u in the main axis, and v in the branches. Production p_9 removes the apex B when the signal reaches it, thus terminating the development of the corresponding lateral branch. Figure 12 shows that, for the values of plastochrons and signal propagation rates specified be the #define statements, the lower branches have less time to grow than the higher branches, and a mesotonic structure develops as a result.

A similar mechanism, based on the pursuit of apices by acropetal signals, has been proposed to model basipetal flowering sequences [43, 16, 23]. These sequences are characterized by the appearance of the first flower near the top of a plant, and a subsequent downward propagation of the flowering zone.

7.2 Attack of a plant by an insect

More complex information flow is considered in the next example. A hypothetical insect explores a growing branching structure and feeds on its apices. The insect always moves along the branches (*i.e.*, it does not jump or drop from one branch to another) and therefore can be treated as an endogenous signal. The insect's behavior at a branching point depends on its direction of motion and the state of the branching point, as explained in Figure 13. In a nutshell, the insect attempts to traverse the entire developing structure using the depth-first strategy. A context-sensitive L-system that integrates plant growth with the behavior of the insect is given below.

```
#define l_L   3     /* length of the left branch */
#define l_R   5     /* length of the right branch */
#define d     5     /* plastochron */
#define w     40    /* delay */
```

$$
\begin{array}{lllll}
\omega & : & W(w)FA(l_L, d) & & \\
p_1: & F & < A(n,m) & : m > 0 & \\
 & & & \rightarrow A(n, m-1) & \\
p_2: & F & < A(n,m) & : n > 0 \;\&\&\; m == 0 & \\
 & & & \rightarrow FA(n-1, d) & \\
p_3: & F & < A(n,m) & : n == 0 \;\&\&\; m == 0 & \\
 & & & \rightarrow L[+FA(l_L, d)][-FA(l_R, d)] & \\
p_4: & W(t) & : t > 0 & \rightarrow W(t-1) & \\
p_5: & W(t) & : t == 0 & \rightarrow U & \\
p_6: & U & < F & \rightarrow FU & \\
p_7: & U & & \rightarrow \varepsilon & \\
p_8: & UL & < + & \rightarrow +U & \\
p_9: & U & < A(n,m) & \rightarrow D & \\
p_{10}: & F & > D & \rightarrow DF & \\
p_{11}: & D & & \rightarrow \varepsilon & \\
p_{12}: & L & > [+D] & \rightarrow UR & \\
p_{13}: & UR & < - & \rightarrow -U & \\
p_{14}: & R & > [\,][-D] & \rightarrow D &
\end{array}
$$

Productions p_1 to p_3 describe the development of a simple branching structure. Starting with a single axis specified by axiom ω, the apex A appends a sequence of branch segments F to the current axis (productions p_1 and p_2), then initiates a pair of new lateral apices (production p_3) that recursively repeat the same pattern. Parameter m is used to count the derivation steps between the creation of consecutive segments F. Parameter n determines the remaining number of segments to be produced before the next branching occurs. The total number of segments in an axis is defined by constants l_L (for the main axis and the branches issued to the left) and l_R (for the branches issued to the right). A newly created branching point is marked by symbol L (production p_3).

After a delay of w steps introduced by production p_4, production p_5 places an insect in the state U at the base of the branching structure. This insect moves upwards, one branch segment per derivation step (productions p_6 and p_7), until it encounters the branching point marker L. The insect is then directed to the left daughter branch (production p_8). After crossing a number of segments and, possibly, further branching points, the insect eventually reaches an apex A. As specified by production p_9, this apex is then removed from the structure, thus stopping further growth of its axis, and the state of the insect is changed from U (moving upwards) to D (moving downwards). The downward movement is simulated by productions p_{10} and p_{11}. Returning to a branching point marked L, the insect changes this mark to R to indicate that the left branch has been already explored, reverts its own state to U, and enters the right branch (productions p_{12} and p_{13}). Coming back from that branch, the insect continues its downward movement (production p_{14}) until it reaches another branching point marked L and enters an unexplored right branch, or until it completes the traversal of the entire structure at its base.

A sequence of images obtained using a straightforward extension of the above L-system is shown in Figure 14. In this case, the insect feeds on the apices of a three-dimensional structure, and a branch that no longer carries any apices wilts.

Similar models can be constructed assuming different traversing

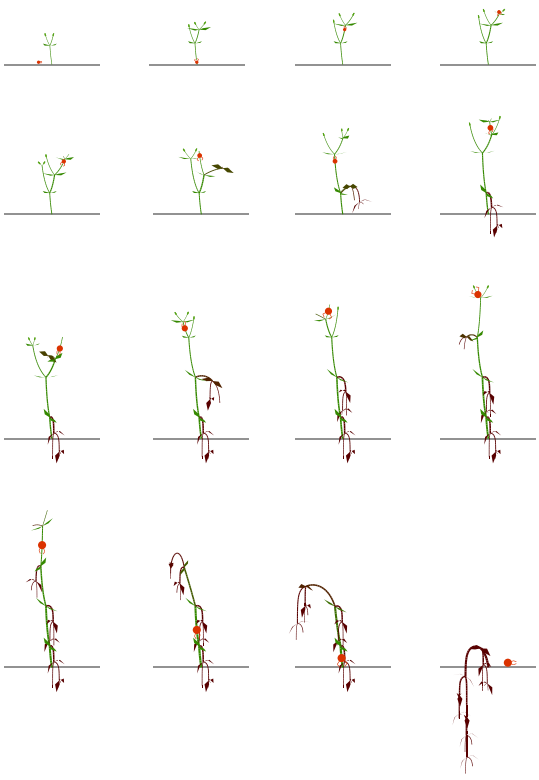


Figure 14: Simulation of the development of a plant attacked by an insect

and feeding strategies for one or many insects (which may interact with each other). Prospective applications of such models include simulation studies of insects used for weed control and of the impact of insects on crop plants [48, 49].

7.3 Development controlled by resource allocation

In the previous examples, discrete information was transferred between the modules of a developing structure. A signal (or insect) was either present or absent at any particular point, and affected the structure in an "all-or-nothing" manner, by removing the apices at the ends of branches. In nature, however, developmental processes are often controlled in a more modulated way, by the quantity of substances (*resources*) exchanged between the modules. For example, the growth of plants depends on the amount of water and minerals absorbed by the roots and carried acropetally (upwards), and by the amount of photosynthates produced by the leaves and transported basipetally. An early developmental model of branching structures making use of quantitative information flow was proposed by Borchert and Honda [3]. Below we restate the essence of this model using the formalism of L-systems, then we extend it to simulate interactions between the shoot and the roots in a growing plant.

Borchert and Honda postulated that the development of a branching structure is controlled by a flow or *flux* of substances, which propagate from the base of the structure towards the apices and supply them with materials needed for growth. When the flux reaching an apex exceeds a predefined threshold value, the apex bifurcates and initiates a lateral branch; otherwise it remains inactive. At branching points the flux is distributed according to the types of the supported internodes (straight or lateral) and the number of apices in the corresponding branches. These numbers are accumulated by messages that originate at the apices and propagate towards the base of the plant. Thus, development is controlled by a cycle of alternating acropetal and basipetal information flow.

An L-system that implements these mechanisms is given below.

#define α_1 10 /* branching angle - straight segment */
#define α_2 32 /* branching angle - lateral segment */
#define σ_0 17 /* initial flux */
#define η 0.89 /* controls input flux changes */
#define λ 0.7 /* flux distribution factor */
#define v_{th} 5.0 /* threshold flux for branching */

ignore: $+\ -/$

ω : $N(1)I(0,2,0,1)A$
p_1 : $N(k) < I(b,m,v,c) : b == 0\ \&\&\ m == 2$
$\rightarrow I(b,1,\sigma_0 * 2 \wedge (k-1) * (\eta \wedge k), c)$
p_2 : $N(k) > I(b,m,v,c) : b == 0\ \&\&\ m == 2 \rightarrow N(k+1)$
p_3 : $I(b_l,m_l,v_l,c_l) < I(b,m,v,c) : m_l == 1\ \&\&\ b == 1$
$\rightarrow I(b,m_l,v_l - v_l * (1-\lambda) * ((c_l - c)/c), c)$
p_4 : $I(b_l,m_l,v_l,c_l) < I(b,m,v,c) : m_l == 1\ \&\&\ b == 2$
$\rightarrow I(b,m_l,v_l * (1-\lambda) * (c/(c_l - c)), c)$
p_5 : $I(b,m,v,c) < A : m == 1\ \&\&\ v > v_{th}$
$\rightarrow /(180)[-(\alpha_2)I(2,2,v*(1-\lambda),1)A]$
$+(\alpha_1)I(1,2,v*\lambda,1)A$
p_6 : $I(b,m,v,c) > A : m == 1\ \&\&\ v <= v_{th} \rightarrow I(b,2,v,c)$
p_7 : $I(b,m,v,c) > [I(b_2,m_2,v_2,c_2) =]I(b_1,m_1,v_1,c_1) :$
$m == 0\ \&\&\ m_1 == 2\ \&\&\ m_2 == 2$
$\rightarrow I(b,2,v,c_1 + c_2)$
p_8 : $I(b,m,v,c) : m == 1 \rightarrow I(b,0,v,c)$
p_9 : $I(b_l,m_l,v_l,c_l) < I(b,m,v,c) : m_l == 2\ \&\&\ m == 2$
$\rightarrow I(b,0,v,c)$

This L-system operates on three types of modules: apices A, internodes I, and an auxiliary module N. The internodes are visualized as lines of unit length. Each internode has four parameters:

- **segment type** b, where 0 denotes base of the tree, 1 – a straight segment, and 2 – a lateral segment;
- **message type** m, where 0 denotes no message currently carried by the internode, 1 – an acropetal message (flux), and 2 – a basipetal message (apex count);
- **flux value** v, and
- **apex count** c.

All internodes are visualized as lines of unit length.

At the beginning of a developmental cycle, indicated by the presence of a basipetal message ($m = 2$) in the basal internode ($b = 0$), production p_1 calculates an input flux value. The expression used for this purpose, $v = \sigma_0 2^{(k-1)\eta^k}$, was introduced by Borchert and Honda to simulate a sigmoid increase of flux penetrating the base of a plant over time. The progress of time is captured by production p_2, which increments the current cycle number k in module N.

Productions p_3 and p_4 simulate acropetal flux propagation and distribute it between the straight segment and the lateral segment. If both the straight and lateral branch support the same number of

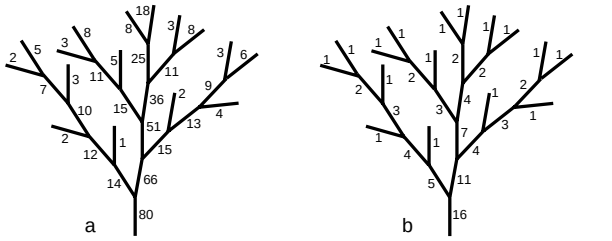


Figure 15: The structure generated by L-system of Section 7.3 at completion of the fifth developmental cycle. The numbers indicate the flow values v rounded to the nearest integer (a), and the numbers of apices c in the branches supported by each internode (b).

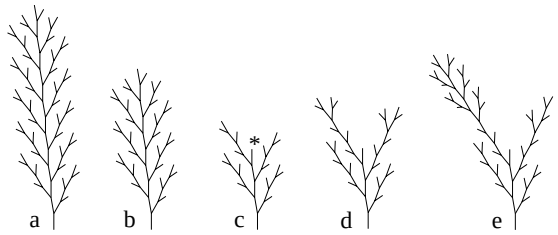


Figure 16: Development of a branching structure simulated using an L-system implementation of the model by Borchert and Honda. (a) Development not affected by pruning; (b, c) the structure immediately before and after pruning; (d, e) the subsequent development of the pruned structure. Based on [3].

apices, the straight segment will obtain a predefined fraction λ of the flux v_l reaching the branching point; the lateral segment will obtain the remainder, $(1-\lambda)v_l$. If a lateral branch supports c apices and its sister straight branch supports c_s apices, the flux reaching the lateral branch is further multiplied by the ratio c/c_s. The number c_s is not directly available to the lateral branch, but it can be calculated as the difference between the number of apices supported by this branch and its mother, $c_s = c_l - c$. In total, the flux directed towards the lateral branch is equal to $v_l(1-\lambda)(c/(c_l-c)$ (production p_3). The remaining flux reaches the straight segment. The parameter c denotes, in this case, the number of apices supported by the straight segment, and the resulting expression is $v_l - v_l(1-\lambda)((c_l-c)/c)$ (production p_4).

Productions p_5 and p_6 control the addition of new segments to the structure. According to production p_5, if the internode preceding an apex A reaches a sufficient flux $v > v_{th}$, the apex will create two new internodes I terminated by apices A. The new segments are assigned an initial message type $m = 2$, which triggers the basipetal signal propagation needed to update the count of apices supported by each segment. Alternatively, if the flux reaching an apex is not sufficient for bifurcation ($v \leq v_{th}$), the supporting internode itself starts the propagation of the basipetal signal (production p_6).

Production p_7 adds the number of apices supported by the daughter branches (c_1 and c_2), and propagates the result to the mother internode. Both input numbers must be available ($m_1 = 2$ and $m_2 = 2$) before basipetal message propagation takes place.

The remaining productions reset the message value m to zero, after the flux values have been transferred acropetally (p_8) or the apex count has been passed basipetally (p_9).

The initial state of the model is determined by the axiom ω. The value of the parameter to module N sets the current cycle number to 1. The initial structure consists of a single internode I terminated by an apex A. The message type indicates the presence of a basipetal message ($m = 2$) which triggers the application of productions p_1 and p_2, initiating the first full developmental cycle. The state of the structure after 35 derivation steps (completion of the fifth developmental cycle) is shown in Figure 15.

A remarkable feature of Borchert and Honda's model is its ability to simulate the response of a plant to its environment. Specifically, after a branch has been pruned, the model redirects the fluxes to the remaining branches and accelerates their growth to compensate for the loss. A sequence of structures that illustrates this phenomenon is shown in Figure 16. In accordance with [3], the L-system used in this case extends the L-system discussed above with parameters and productions needed to capture the effect of aging. Consequently, a branch that was unable to grow for a given number of developmental cycles dies: it loses the ability to develop further and stops taking any fluxes.

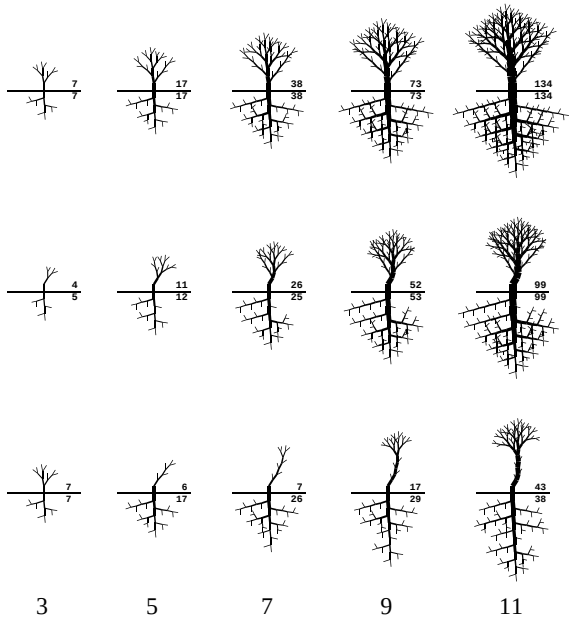


Figure 17: Application of the Borchert and Honda's model to the simulation of a complete plant, showing development unaffected by pruning (top row), affected by pruning during the third cycle of development (middle row), and affected by pruning during the fifth cycle of development (bottom row). The numbers of live apices in the shoot and root are indicated above and below the ground level. The numbers at the base of the figure indicate the number of completed developmental cycles.

Similar behavior is shown in Figure 17. In this case, two structures representing the shoot and the root of a plant are generated simultaneously. The flux penetrating the root at the beginning of a developmental cycle is assumed to be proportional to the number of apices in the shoot; reciprocally, the flux penetrating the shoot is proportional to the number of apices in the root. These assumptions form a crude approximation of plant physiology, whereby the photosynthates produced by the shoot fuel the development of the root, and water and mineral compounds gathered by the root are required for the development of the shoot. The model also assumes an increase of internode width over time and a gradual rotation of a lateral segment to the straight segment position, after the straight segment has

been lost. The developmental sequence shown in the top row of Figure 17 is unaffected by pruning. The shoot and the root develop in concert. The next two rows illustrate development affected by a loss of branches. The removal of a shoot branch slows down the development of the root; on the other hand, the large size of the root, compared to the remaining shoot, fuels a fast re-growth of the shoot. Eventually, the plant is able to redress the balance between the size of the shoot and the root. This is a non-obvious consequence of the model, which illustrates the usefulness of L-systems in predicting the global behavior of plants, given the specification of their components.

8 Conclusions

L-system models integrate local processes, taking place at the level of individual modules, into developmental patterns and structures of entire plants. Consequently, they address the central problem of morphogenesis: the description and understanding of mechanisms through which living organisms acquire their form. This aspect of modeling motivated the original biological applications of L-systems investigated by Lindenmayer and his collaborators, and continues to play a key role in current biological research using L-systems. The emergence of global forms and developmental patterns is also important in the application of L-systems to computer graphics, because it makes it possible to create realistic representations of growing plants using relatively easy to specify, compact sets of data.

In principle, the mathematical formulation of L-systems should also make it possible to address biologically relevant questions in the form of a deductive theory of plant development. The results of this theory could be potentially more general than simulations, which are inherently limited to case studies. Unfortunately, construction of such a theory still seems quite remote. One reason is the lack of a precise mathematical description of plant form. This is not of crucial importance in simulations, where the results are evaluated visually, but impedes the formulation of theorems and proofs. Another difficulty is the discrepancy between studies on the theory of L-systems and the needs of biological modeling. Most theoretical results are pertinent to non-parametric 0L-systems that operate on non-branching strings without geometric interpretation (for examples, see [50]). In contrast, L-system models of biological phenomena often involve parameters, interactions between modules, and geometric features of the modeled structures. We hope that further development of the theory of L-systems will bridge this gap.

9 Acknowledgements

An overview of L-systems was the subject of several invited lectures and tutorials presented recently by P. Prusinkiewicz. Consequently, this paper includes sections of previous surveys [36, 37], and coincides with [35]. The idea of using L-systems to simulate the interaction between plants and insects was proposed by Peter Room. The reported research has been sponsored by grants and graduate scholarships from the Natural Sciences and Engineering Council of Canada.

References

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

[2] A. Bell. *Plant form: An illustrated guide to flowering plants.* Oxford University Press, Oxford, 1991.

[3] 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.

[4] M. J. M. de Boer. *Analysis and computer generation of division patterns in cell layers using developmental algorithms*. PhD thesis, University of Utrecht, 1989.

[5] M. J. M. de Boer, F. D. Fracchia, and P. Prusinkiewicz. A model for cellular development in morphogenetic fields. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, pages 351–370. Springer-Verlag, Berlin, 1992.

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

[7] F. D. Fracchia, P. Prusinkiewicz, and M. J. M. de Boer. Animation of the development of multicellular structures. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation '90*, pages 3–18, Tokyo, 1990. Springer-Verlag.

[8] 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.

[9] 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.

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

[11] J. S. Hanan. *Parametric L-systems and their application to the modelling and visualization of plants.* PhD thesis, University of Regina, June 1992.

[12] J. L. Harper and A. D. Bell. The population dynamics of growth forms in organisms with modular construction. In R. M. Anderson, B. D. Turner, and L. R. Taylor, editors, *Population dynamics*, pages 29–52. Blackwell, Oxford, 1979.

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

[14] 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.

[15] M. James, J. Hanan, and P. Prusinkiewicz. CPFG version 2.0 user's manual. Manuscript, Department of Computer Science, University of Calgary, 1993, 50 pages.

[16] 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.

[17] B. W. Kernighan and D. M. Ritchie. *The C programming language. Second edition.* Prentice Hall, Englewood Cliffs, 1988.

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

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

[20] A. Lindenmayer. Developmental algorithms for multicellular organisms: A survey of L-systems. *Journal of Theoretical Biology*, 54:3–22, 1975.

[21] A. Lindenmayer. Algorithms for plant morphogenesis. In R. Sattler, editor, *Theoretical plant morphology*, pages 37–81. Leiden University Press, The Hague, 1978.

[22] 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.

[23] 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.

[24] A. Lindenmayer. Models for multicellular development: Characterization, inference and complexity of L-systems. In A. Kelemenová and J. Kelemen, editors, *Trends, techniques and problems in theoretical computer science*, Lecture Notes in Computer Science 281, pages 138–168. Springer-Verlag, Berlin, 1987.

[25] A. Lindenmayer and H. Jürgensen. Grammars of development: Discrete-state models for growth, differentiation and gene expression in modular organisms. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, pages 3–21. Springer-Verlag, Berlin, 1992.

[26] A. Lindenmayer and P. Prusinkiewicz. Developmental models of multicellular organisms: A computer graphics perspective. In C. G. Langton, editor, *Artificial Life*, pages 221–249. Addison-Wesley, Redwood City, 1988.

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

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

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

[30] R. Měch and P. Prusinkiewicz. Visual models of plants interacting with their environment. Proceedings of SIGGRAPH '96 (New Orleans, Louisiana, August 4–9, 1996). ACM SIGGRAPH, New York, 1996, pp. 397–410.

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

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

[33] 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.

[34] P. Prusinkiewicz. Visual models of morphogenesis. *Artificial Life*, 1:61–74, 1994.

[35] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Měch. L-systems: from the theory to visual models of plants. *Machine Graphics and Vision*, 5(1/2):365–392, 1996.

[36] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Měch. Visual models of plant development. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, Vol. III*: Beyond words*, pages 535–597. Springer, Berlin, 1997.

[37] P. Prusinkiewicz, M. Hammel, R. Měch, and J. Hanan. The artificial life of plants. In D. Terzopoulos, editor, *SIGGRAPH 1995 Course Notes on Artificial Life*, pages 1–1 – 1–38. ACM SIGGRAPH, 1995.

[38] P. Prusinkiewicz and J. Hanan. *Lindenmayer systems, fractals, and plants*, volume 79 of *Lecture Notes in Biomathematics*. Springer-Verlag, Berlin, 1989 (second printing 1992).

[39] 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, Chichester, 1990.

[40] P. Prusinkiewicz and J. Hanan. L-systems: From formalism to programming languages. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, pages 193–211. Springer-Verlag, Berlin, 1992.

[41] P. Prusinkiewicz, M. James, and R. Měch. Synthetic topiary. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994). ACM SIGGRAPH, New York, 1994, pp. 351–358.

[42] P. Prusinkiewicz and L. Kari. Subapical bracketed L-systems. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Graph grammars and their application to computer science; Fifth International Workshop*, Lecture Notes in Computer Science 1073, pages 550–564. Springer-Verlag, Berlin, 1996.

[43] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[44] P. Prusinkiewicz, A. Lindenmayer, and F. D. Fracchia. Synthesis of space-filling curves on the square grid. In H.-O. Peitgen, J. M. Henriques, and L. F. Penedo, editors, *Fractals in the fundamental and applied sciences*, pages 341–366. North-Holland, Amsterdam, 1991.

[45] P. Prusinkiewicz, W. Remphrey, C. Davidson, and M. Hammel. Modeling the architecture of expanding *Fraxinus pennsylvanica* shoots using L-systems. *Canadian Journal of Botany*, 72:701–714, 1994.

[46] D. M. Raup. Geometric analysis of shell coiling: general problems. *Journal of Paleontology*, 40:1178–1190, 1966.

[47] D. M. Raup and A. Michelson. Theoretical morphology of the coiled shell. *Science*, 147:1294–1295, 1965.

[48] P. M. Room, J. S. Hanan, and P. Prusinkiewicz. Virtual plants: new perspectives for ecologists, pathologists, and agricultural scientists. *Trends in Plant Science*, 1(1):33–38, 1996.

[49] P. M. Room, L. Maillette, and J. Hanan. Module and metamer dynamics and virtual plants. *Advances in Ecological Research*, 25:105–157, 1994.

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

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

[52] 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.

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

[54] H. von Koch. Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes. *Acta Mathematica*, 30:145–174, 1905.

[55] D. M. Waller and D. A. Steingraeber. Branching and modular growth: Theoretical models and empirical patterns. In J. B. C. Jackson and L. W. Buss, editors, *Population biology and evolution of clonal organisms*, pages 225–257. Yale University Press, New Haven, 1985.

Visual Models of Plants Interacting with Their Environment

Radomír Měch and Przemyslaw Prusinkiewicz[1]
University of Calgary

ABSTRACT

Interaction with the environment is a key factor affecting the development of plants and plant ecosystems. In this paper we introduce a modeling framework that makes it possible to simulate and visualize a wide range of interactions at the level of plant architecture. This framework extends the formalism of Lindenmayer systems with constructs needed to model bi-directional information exchange between plants and their environment. We illustrate the proposed framework with models and simulations that capture the development of tree branches limited by collisions, the colonizing growth of clonal plants competing for space in favorable areas, the interaction between roots competing for water in the soil, and the competition within and between trees for access to light. Computer animation and visualization techniques make it possible to better understand the modeled processes and lead to realistic images of plants within their environmental context.

CR categories: F.4.2 **[Mathematical Logic and Formal Languages]**: Grammars and Other Rewriting Systems: *Parallel rewriting systems*, I.3.7 **[Computer Graphics]**: Three-Dimensional Graphics and Realism, I.6.3 **[Simulation and Modeling]**: Applications, J.3 **[Life and Medical Sciences]**: Biology.

Keywords: scientific visualization, realistic image synthesis, software design, L-system, modeling, simulation, ecosystem, plant development, clonal plant, root, tree.

1 INTRODUCTION

Computer modeling and visualization of plant development can be traced back to 1962, when Ulam applied cellular automata to simulate the development of branching patterns, thought of as an abstract representation of plants [53]. Subsequently, Cohen presented a more realistic model operating in continuous space [13], Lindenmayer proposed the formalism of L-systems as a general framework for plant modeling [38, 39], and Honda introduced the first computer model of tree structures [32]. From these origins, plant modeling emerged as a vibrant area of interdisciplinary research, attracting the efforts of biologists, applied plant scientists, mathematicians, and computer scientists. Computer graphics, in particular, contributed a wide range of models and methods for synthesizing images of plants. See [18, 48, 54] for recent reviews of the main results.

One aspect of plant structure and behavior neglected by most models is the interaction between plants and their environment (including other plants). Indeed, the incorporation of interactions has been identified as one of the main outstanding problems in the domain of plant modeling [48] (see also [15, 18, 50]). Its solution is needed to construct predictive models suitable for applications ranging from computer-assisted landscape and garden design to the determination of crop and lumber yields in agriculture and forestry.

Using the information flow between a plant and its environment as the classification key, we can distinguish three forms of interaction and the associated models of plant-environment systems devised to date:

1. The plant is affected by global properties of the environment, such as day length controlling the initiation of flowering [23] and daily minimum and maximum temperatures modulating the growth rate [28].

2. The plant is affected by local properties of the environment, such as the presence of obstacles controlling the spread of grass [2] and directing the growth of tree roots [26], geometry of support for climbing plants [2, 25], soil resistance and temperature in various soil layers [16], and predefined geometry of surfaces to which plant branches are pruned [45].

3. The plant interacts with the environment in an information feedback loop, where the environment affects the plant and the plant reciprocally affects the environment. This type of interaction is related to *sighted* [4] or *exogenous* [42] mechanisms controlling plant development, in which parts of a plant influence the development of other parts of the same or a different plant through the space in which they grow. Specific models capture:

   – competition for *space* (including collision detection and access to light) between segments of essentially two-dimensional schematic branching structures [4, 13, 21, 22, 33, 34, 36];

   – competition between root tips for *nutrients* and *water* transported in soil [12, 37] (this mechanism is related to competition between growing branches of corals and sponges for nutrients diffusing in water [34]);

[1]Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4 (mech|pwp@cpsc.ucalgary.ca)

Published in the Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996). In *Computer Graphics* Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, New York, pp. 397–410.

– competition for *light* between three-dimensional shoots of herbaceous plants [25] and branches of trees [9, 10, 11, 15, 33, 35, 52].

Models of exogenous phenomena require a comprehensive representation of both the developing plant and the environment. Consequently, they are the most difficult to formulate, implement, and document. Programs addressed to the biological audience are often limited to narrow groups of plants (for example, poplars [9] or trees in the pine family [21]), and present the results in a rudimentary graphical form. On the other hand, models addressed to the computer graphics audience use more advanced techniques for realistic image synthesis, but put little emphasis on the faithful reproduction of physiological mechanisms characteristic to specific plants.

In this paper we propose a general *framework* (defined as a modeling methodology supported by appropriate software) for modeling, simulating, and visualizing the development of plants that bi-directionally interact with their environment. The usefulness of modeling frameworks for simulation studies of models with complex (emergent) behavior is manifested by previous work in theoretical biology, artificial life, and computer graphics. Examples include cellular automata [51], systems for simulating behavior of cellular structures in discrete [1] and continuous [20] spaces, and L-system-based frameworks for modeling plants [36, 46]. Frameworks may have the form of a general-purpose simulation program that accepts models described in a suitable mini-language as input, *e.g.* [36, 46], or a set of library programs [27]. Compared to special-purpose programs, they offer the following benefits:

- At the conceptual level, they facilitate the design, specification, documentation, and comparison of models.
- At the level of model implementation, they make it possible to develop software that can be reused in various models. Specifically, graphical capabilities needed to visualize the models become a part of the modeling framework, and do not have to be reimplemented.
- Finally, flexible conceptual and software frameworks facilitate interactive experimentation with the models [46, Appendix A].

Our framework is intended both for purpose of image synthesis and as a research and visualization tool for model studies in plant morphogenesis and ecology. These goals are addressed at the levels of the simulation system and the modeling language design. The underlying paradigm of plant-environment interaction is described in Section 2. The resulting design of the simulation software is outlined in Section 3. The language for specifying plant models is presented in Section 4. It extends the concept of environmentally-sensitive L-systems [45] with constructs for bi-directional communication with the environment. The following sections illustrate the proposed framework with concrete models of plants interacting with their environment. The examples include: the development of planar branching systems controlled by the crowding of apices (Section 5), the development of clonal plants controlled by both the crowding of ramets and the quality of terrain (Section 6), the development of roots controlled by the concentration of water transported in the soil (Section 7), and the development of tree crowns affected by the local distribution of light (Section 8) The paper concludes with an evaluation of the results and a list of open problems (Section 9).

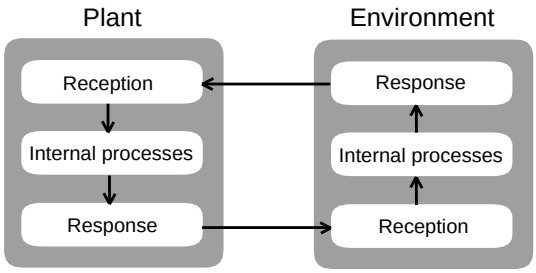


Figure 1: Conceptual model of plant and environment treated as communicating concurrent processes

2 CONCEPTUAL MODEL

As described by Hart [30], every environmentally controlled phenomenon can be considered as a chain of causally linked events. After a stimulus is perceived by the plant, information in some form is transported through the plant body (unless the site of stimulus perception coincides with the site of response), and the plant reacts. This reaction reciprocally affects the environment, causing its modification that in turn affects the plant. For example, roots growing in the soil can absorb or extract water (depending on the water concentration in their vicinity). This initiates a flow of water in the soil towards the depleted areas, which in turn affects further growth of the roots [12, 24].

According to this description, the interaction of a plant with the environment can be conceptualized as two concurrent processes that communicate with each other, thus forming a feedback loop of information flow (Figure 1). The plant process performs the following functions:

- reception of information about the environment in the form of scalar or vector values representing the stimuli perceived by specific organs;
- transport and processing of information inside the plant;
- generation of the response in the form of growth changes (*e.g.* development of new branches) and direct output of information to the environment (*e.g.* uptake and excretion of substances by a root tip).

Similarly, the environmental process includes mechanisms for the:

- perception of the plant's actions;
- simulation of internal processes in the environment (*e.g.* the diffusion of substances or propagation of light);
- presentation of the modified environment in a form perceivable by the plant.

The design of a simulation system based on this conceptual model is presented next.

3 SYSTEM DESIGN

The goal is to create a framework, in which a wide range of plant structures and environments can be easily created, modified, and

used for experimentation. This requirement led us to the following design decisions:

- The plant and the environment should be modeled by separate programs and run as two communicating processes. This design is:
  - compatible with the assumed conceptual model of plant-environment interaction (Figure 1);
  - consistent with the principles of structured design (modules with clearly specified functions jointly contribute to the solution of a problem by communicating through a well defined interface; information local to each module is hidden from other modules);
  - appropriate for interactive experimentation with the models; in particular, changes in the plant program can be implemented without affecting the environmental program, and *vice versa*;
  - extensible to distributed computing environments, where different components of a large ecosystem may be simulated using separate computers.
- The user should have control over the type and amount of information exchanged between the processes representing the plant and the environment, so that all the needed but no superfluous information is transferred.
- Plant models should be specified in a language based on L-systems, equipped with constructs for bi-directional communication between the plant and the environment. This decision has the following rationale:
  - A succinct description of the models in an interpreted language facilitates experimentation involving modifications to the models;
  - L-systems capture two fundamental mechanisms that control development, namely flow of information from a mother module to its offspring (cellular descent) and flow of information between coexisting modules (endogenous interaction) [38]. The latter mechanism plays an essential role in transmitting information from the site of stimulus perception to the site of the response. Moreover, L-systems have been extended to allow for input of information from the environment (see Section 4);
  - Modeling of plants using L-systems has reached a relatively advanced state, manifested by models ranging from algae to herbaceous plants and trees [43, 46].
- Given the variety of processes that may take place in the environment, they should be modeled using special-purpose programs.
- Generic aspects of modeling, not specific to particular models, should be supported by the modeling system. This includes:
  - an L-system-based plant modeling program, which interprets L-systems supplied as its input and visualizes the results, and
  - the support for communication and synchronization of processes simulating the modeled plant and the environment.

A system architecture stemming from this design is shown in Figure 2. We will describe it from the perspective of extensions to the formalism of L-systems.

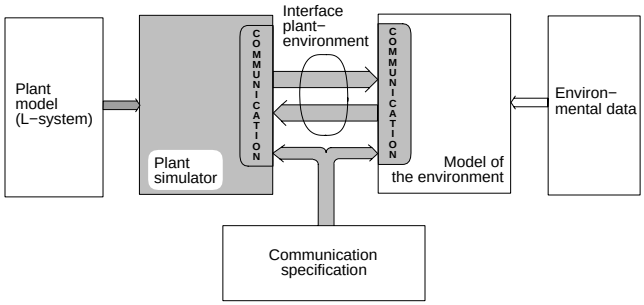


Figure 2: Organization of the software for modeling plants interacting with their environment. Shaded rectangles indicate components of the modeling framework, clear rectangles indicate programs and data that must be created by a user specifying a new model of a plant or environment. Shaded arrows indicate information exchanged in a standardized format.

4 OPEN L-SYSTEMS

Historically, L-systems were conceived as closed cybernetic systems, incapable of simulating any form of communication between the modeled plant and its environment. In the first step towards the inclusion of environmental factors, Rozenberg defined *table L-systems*, which allow for a change in the set of developmental rules (the production set of the L-system) in response to a change in the environment [31, 49]. Table L-systems were applied, for example, to capture the switch from the production of leaves to the production of flowers by the apex of a plant due to a change in day length [23]. *Parametric L-systems* [29, 46], introduced later, made it possible to implement a variant of this technique, with the environment affecting the model in a quantitative rather than qualitative manner. In a case study illustrating this possibility, weather data containing daily minimum and maximum temperatures were used to control the rate of growth in a bean model [28]. *Environmentally-sensitive L-systems* [45] represented the next step in the inclusion of environmental factors, in which local rather than global properties of the environment affected the model. The new concept was the introduction of query symbols, returning current position or orientation of the turtle in the underlying coordinate system. These parameters could be passed as arguments to user-defined functions, returning local properties of the environment at the queried location. Environmentally-sensitive L-systems were illustrated by models of topiary scenes. The environmental functions defined geometric shapes, to which trees were pruned.

Open L-systems, introduced in this paper, augment the functionality of environmentally-sensitive L-systems using a reserved symbol for bilateral communication with the environment. In short, parameters associated with an occurrence of the communication symbol can be set by the environment and transferred to the plant model, or set by the plant model and transferred to the environment. The environment is no longer represented by a simple function, but becomes an active process that may react to the information from the plant. Thus, plants are modeled as open cybernetic systems, sending information to and receiving information from the environment.

In order to describe open L-systems in more detail, we need to recall the rudiments of L-systems with turtle interpretation. Our presentation is reproduced from [45].

An L-system is a parallel rewriting system operating on branching structures represented as *bracketed strings* of symbols with associated numerical parameters, called *modules*. Matching pairs of square brackets enclose branches. Simulation begins with an initial string called the *axiom*, and proceeds in a sequence of discrete *derivation steps*. In each step, *rewriting rules* or *productions* replace all modules in the predecessor string by successor modules. The applicability of a production depends on a predecessor's context (in context-sensitive L-systems), values of parameters (in productions guarded by conditions), and on random factors (in stochastic L-systems). Typically, a production has the format:

$$id : lc < pred > rc : cond \rightarrow succ : prob$$

where id is the production identifier (label), lc, $pred$, and rc are the left context, the strict predecessor, and the right context, $cond$ is the condition, $succ$ is the successor, and $prob$ is the probability of production application. The strict predecessor and the successor are the only mandatory fields. For example, the L-system given below consists of axiom ω and three productions p_1, p_2, and p_3.

ω: A(1)B(3)A(5)
p_1: A(x) $\rightarrow$A(x+1) : 0.4
p_2: A(x) $\rightarrow$B(x–1) : 0.6
p_3: A(x) < B(y) > A(z) : y < 4 $\rightarrow$B(x+z)[A(y)]

The stochastic productions p_1 and p_2 replace module $A(x)$ by either $A(x+1)$ or $B(x-1)$, with probabilities equal to 0.4 and 0.6, respectively. The context-sensitive production p_3 replaces a module $B(y)$ with left context $A(x)$ and right context $A(z)$ by module $B(x+z)$ supporting branch $A(y)$. The application of this production is guarded by condition $y < 4$. Consequently, the first derivation step may have the form:

A(1)B(3)A(5) $\Rightarrow$ A(2)B(6)[A(3)]B(4)

It was assumed that, as a result of random choice, production p_1 was applied to the module $A(1)$, and production p_2 to the module $A(5)$. Production p_3 was applied to the module $B(3)$, because it occurred with the required left and right context, and the condition $3 < 4$ was true.

In the L-systems presented as examples we also use several additional constructs (*cf.* [29, 44]):

- Productions may include statements assigning values to local variables. These statements are enclosed in curly braces and separated by semicolons.
- The L-systems may also include arrays. References to array elements follow the syntax of C; for example, $MaxLen[order]$.
- The list of productions is ordered. In the deterministic case, the first matching production applies. In the stochastic case, the set of all matching productions is established, and one of them is chosen according to the specified probabilities.

For details of the L-system syntax see [29, 43, 46].

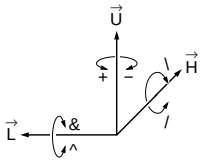


Figure 3: Controlling the turtle in three dimensions

In contrast to the parallel application of productions in each derivation step, the interpretation of the resulting strings proceeds sequentially, with reserved modules acting as commands to a LOGO-style turtle [46]. At any point of the string, the *turtle state* is characterized by a position vector $\vec{P}$ and three mutually perpendicular orientation vectors $\vec{H}$, $\vec{U}$, and $\vec{L}$, indicating the turtle's heading, the up direction, and the direction to the left (Figure 3). Module F causes the turtle to draw a line in the current direction. Modules $+$, $-$, &, $\wedge$, / and \ rotate the turtle around one of the vectors $\vec{H}, \vec{U}$, or $\vec{L}$, as shown in Figure 3. The length of the line and the magnitude of the rotation angle can be given globally or specified as parameters of individual modules. During the interpretation of branches, the opening square bracket pushes the current position and orientation of the turtle on a stack, and the closing bracket restores the turtle to the position and orientation popped from the stack. A special interpretation is reserved for the module %, which cuts a branch by erasing all symbols in the string from the point of its occurrence to the end of the branch [29]. The meaning of many symbols depends on the context in which they occur; for example, $+$ and $-$ denote arithmetic operators as well as modules that rotate the turtle.

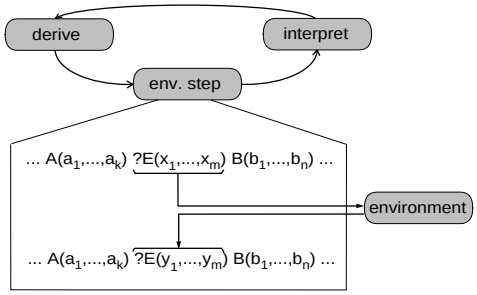


Figure 4: Information flow during the simulation of a plant interacting with the environment, implemented using an open L-system

The turtle interpretation of L-systems described above was designed to visualize models in a postprocessing step, with no effect on the L-system operation. Position and orientation of the turtle are important, however, while considering environmental phenomena, such as collisions with obstacles and exposure to light. The environmentally-sensitive extension of L-systems makes these attributes accessible during the rewriting process [45]. The generated string is interpreted after each derivation step, and turtle attributes found during the interpretation are returned as parameters to reserved *query modules*. Syntactically, the query modules have the form $?X(x, y, z)$, where $X = P, H, U,$ or L. Depending on the actual symbol X, the values of parameters x, y, and z represent a position or an orientation vector.

Open L-systems are a generalization of this concept. *Communication modules* of the form $?E(x_1, \ldots, x_m)$ are used both to send and receive environmental information represented by the values of parameters $x_1, \ldots, x_m$ (Figure 4). To this end, the string resulting from a derivation step is scanned from left to right to determine the state of the turtle associated with each symbol. This phase is similar to the graphical interpretation of the string, except that the results need not be visualized. Upon encountering a communication symbol, the plant process creates and sends a message to the environment including all or a part of the following information:

- the address (position in the string) of the communication module (mandatory field needed to identify this module when a reply comes from the environment),
- values of parameters x_i,
- the state of the turtle (coordinates of the position and orientation

vector, as well as some other attributes, such as current line width),

- the type and parameters of the module following the communication module in the string (not used in the examples discussed in this paper).

The exact message format is defined in a *communication specification file*, shared between the programs modeling the plant and the environment (Figure 2). Consequently, it is possible to include only the information needed in a particular model in the messages sent to the environment. Transfer of the last message corresponding to the current scan of the string is signaled by a reserved end-of-transmission message, which may be used by the environmental process to start its operation.

The messages output by the plant modeling program are transferred to the process that simulates the environment using an interprocess communication mechanism provided by the underlying operating system (a pair of UNIX pipes or shared memory with access synchronized using semaphores, for example). The environment processes that information and returns the results to the plant model using messages in the following format:

- the address of the target communication module,
- values of parameters y_i carrying the output from the environment.

The plant process uses the received information to set parameter values in the communication modules (Figure 4). The use of addresses makes it possible to send replies only to selected communication modules. Details of the mapping of messages received by the plant process to the parameters of the communication modules are defined in the communication specification file.

After all replies generated by the environment have been received (a fact indicated by an end-of-transmission message sent by the environment), the resulting string may be interpreted and visualized, and the next derivation step may be performed, initiating another cycle of the simulation.

Note that, by preceding every symbol in the string with a communication module it is possible to pass complete information about the model to the environment. Usually, however, only partial information about the state of a plant is needed as input to the environment. Proper placement of communication modules in the model, combined with careful selection of the information to be exchanged, provide a means for keeping the amount of transferred information at a manageable level.

We will illustrate the operation of open L-systems within the context of complete models of plant-environment interactions, using examples motivated by actual biological problems.

5 A MODEL OF BRANCH TIERS

Background. Apical meristems, located at the endpoints of branches, are engines of plant development. The apices grow, contributing to the elongation of branch segments, and from time to time divide, spawning the development of new branches. If all apices divided periodically, the number of apices and branch segments would increase exponentially. Observations of real branching structures show, however, that the increase in the number of segments is less than exponential [8]. Honda and his collaborators modeled several hypothetical mechanisms that may control the extent of branching in order to prevent overcrowding [7, 33] (see also [4]). One of the models [33], supported by measurements and earlier simulations of the tropical tree *Terminalia catappa* [19], assumes an exogenous interaction mechanism. *Terminalia* branches form horizontal tiers, and the model is limited to a single tier, treated as a two-dimensional structure. In this case, the competition for light effectively amounts to collision detection between the apices and leaf clusters. We reproduce this model as the simplest example illustrating the methodology proposed in this paper.

Communication specification. The plant communicates with the environment using communication modules of the form $?E(x)$. Messages sent to the environment include the turtle position and the value of parameter x, interpreted as the vigor of the corresponding apex. On this basis, the environmental process determines the fate of each apex. A parameter value of $x = 0$ returned to the plant indicates that the development of the corresponding branch will be terminated. A value of $x = 1$ allows for further branching.

The model of the environment. The environmental process considers each apex or non-terminal node of the developing tier as the center of a circular leaf cluster, and maintains a list of all clusters present. New clusters are added in response to messages received from the plant. All clusters have the same radius ρ, specified in the environmental data file (*cf.* Figure 2). In order to determine the fate of the apices, the environment compares apex positions with leaf cluster positions, and authorizes an apex to grow if it does not fall into an existing leaf cluster, or if it falls into a cluster surrounding an apex with a smaller vigor value.

The plant model. The plant model is expressed as an open L-system. The values of constants are taken from [33].

```
#define r1 0.94      /* contraction ratio and vigor 1 */
#define r2 0.87      /* contraction ratio and vigor 2 */
#define α1 24.4      /* branching angle 1 */
#define α2 36.9      /* branching angle 2 */
#define φ 138.5      /* divergence angle */
ω: –(90)[F(1)?E(1)A(1)]+(φ)[F(1)/?E(1)A(1)]
   +(φ)[F(1)?E(1)A(1)]+(φ)[F(1)/?E(1)A(1)]
   +(φ)[F(1)?E(1)A(1)]

p1: ?E(x) < A(v) : x == 1 →
        [+(α2)F(v*r2)?E(r2)A(v*r2)] –(α1)F(v*r1)/?E(r1)A(v*r1)
p2: ?E(x) → ε
```

The axiom ω specifies the initial structure as a whorl of five branch segments F. The divergence angle φ between consecutive segments is equal to 138.5°. Each segment is terminated by a communication symbol $?E$ followed by an apex A. In addition, two branches include module $/$, which changes the directions at which subsequent branches will be issued (left *vs.* right) by rotating the apex 180° around the segment axis.

Production p_1 describes the operation of the apices. If the value of parameter x returned by a communication module $?E$ is not 1, the associated apex will remain inactive (do nothing). Otherwise the apex will produce a pair of new branch segments at angles α_1 and α_2 with respect to the mother segment. Constants r_1 and r_2 determine the lengths of the daughter segments as fractions of the length of their mother segment. The values r_1 and r_2 are also passed to the process simulating the environment using communication modules $?E$. Communication modules created in the previous derivation step are no longer needed and are removed by production p_2.

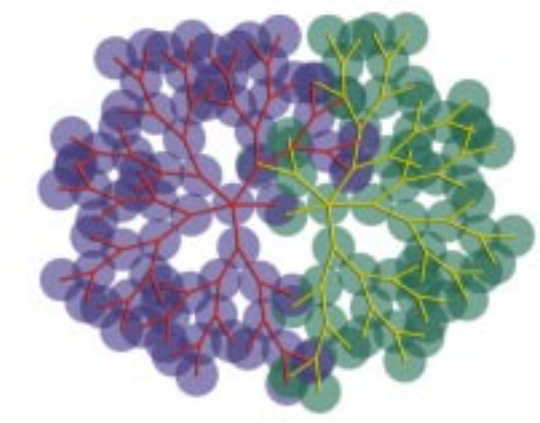


Figure 5: Competition for space between two tiers of branches simulated using the Honda model

Simulation. Figure 5 illustrates the competition for space between two tiers developing next to each other. The extent of branching in each tier is limited by collisions between its apices and its own or the neighbor's leaf clusters. The limited growth of each structure in the direction of its neighbor illustrates the phenomenon of *morphological plasticity*, or adaptation of the form of plants to their environment [17].

6 A MODEL OF FORAGING IN CLONAL PLANTS

Background. Foraging (propagation) patterns in clonal plants provide another excellent example of response to crowding. A clonal plant spreads by means of horizontal stem segments (*spacers*), which form a branching structure that grows along the ground and connects individual plants (*ramets*) [3]. Each ramet consists of a leaf supported by an upright stem and one or more buds, which may give rise to further spacers and ramets. Their gradual death, after a certain amount of time, causes gradual separation of the whole structure (the *clone*) into independent parts.

Following the surface of the soil, clonal plants can be captured using models operating in two dimensions [5], and in that respect resemble *Terminalia* tiers. We propose a model of a hypothetical plant that responds to favorable environmental conditions (high local intensity of light) by more extensive branching and reduced size of leaves (allowing for more dense packing of ramets). It has been inspired by a computer model of clover outlined by Bell [4], the analysis of responses of clonal plants to the environment presented by Dong [17], and the computer models and descriptions of vegetative multiplication of plants involving the death of intervening connections by Room [47].

Communication specification. The plant sends messages to the environment that include turtle position and two parameters associated with the communications symbol, $?E(type, x)$. The first parameter is equal to 0, 1, or 2, and determines the type of exchanged information as follows:

- The message $?E(0, x)$ represents a request for the light intensity (irradiance [14]) at the position of the communication module. The environment responds by setting x to the intensity of incoming light, ranging from 0 (no light) to 1 (full light).
- The message $?E(1, x)$ notifies the environment about the creation of a ramet with a leaf of radius x at the position of the communication module. No output is generated by the environment in response to this message.
- The message $?E(2, x)$ notifies the environment about the death of a ramet with a leaf of radius x at the position of the communication module. Again, no output is generated by the environment.

The model of the environment. The purpose of the environment process is to determine light intensity at the locations requested by the plant. The ground is divided into patches (specified as a raster image using a paint program), with different light intensities assigned to each patch. In the absence of shading, these intensities are returned by the environmental process in response to messages of type 0. To consider shading, the environment keeps track of the set of ramets, adding new ramets in response to a messages of type 1, and deleting dead ramets in response to messages of type 2. If a sampling point falls in an area occupied by a ramet, the returned light intensity is equal to 0 (leaves are assumed to be opaque, and located above the sampling points).

The plant model. The essential features of the plant model are specified by the following open L-system.

```
#define α 45              /* branching angle */
#define MinLight 0.1      /* light intensity threshold */
#define MaxAge 20         /* lifetime of ramets and spacers */
#define Len 2.0           /* length of spacers */
#define Prob_B(x)         (0.12+x*0.42)
#define Prob_R(x)         (0.03+x*0.54)
#define Radius(x)         (sqrt(15–x*5)/π)
ω:  A(1)?E(0,0)

p1: A(dir) > ?E(0,x) : x >= MinLight
                  → R(x)B(x,dir)F(Len,0)A(–dir)?E(0,0)
p2: A(dir) > ?E(0,x) : x < MinLight  → ε

p3: B(x,dir) → [+(α*dir)F(Len,0)A(–dir)?E(0,0)] : Prob_B(x)
p4: B(x,dir) → ε : 1–Prob_B(x)

p5: R(x) → [@o(Radius(x),0)?E(1,Radius(x))] : Prob_R(x)
p6: R(x) → ε : 1–Prob_R(x)

p7: @o(radius,age): age < MaxAge  → @o(radius,age+1)
p8: @o(radius,age): age == MaxAge  → ?E(2,radius)

p9: F(len,age): age < MaxAge  → F(len,age+1)
p10: F(len,age): age == MaxAge  → f(len)

p11: ?E(type,x) → ε
```

The initial structure specified by the axiom ω consists of an apex A followed by the communication module $?E$. If the intensity of light x reaching an apex is insufficient (below the threshold $MinLight$), the apex dies (production p_2). Otherwise, the apex creates a ramet initial R (*i.e.*, a module that will yield a ramet), a branch initial B, a spacer F, and a new apex A terminated by communication module $?E$ (production p_1). The parameter dir, valued either 1 or -1, controls the direction of branching. Parameters of the spacer module specify its length and age.

A branch initial B may create a lateral branch with its own apex A and communication module $?E$ (production p_3), or it may die and disappear from the system (production p_4). The probability of survival is an increasing linear function $Prob_B$ of the light intensity x that has reached the mother apex A in the previous derivation step. A similar stochastic mechanism describes the production of a ramet by the ramet initial R (productions p_5 and p_6), with the probability of ramet formation controlled by an increasing linear function $Prob_R$. The ramet is represented as a circle $@o$; its radius is a decreasing function $Radius$ of the light intensity x. As in the case of spacers, the second parameter of a ramet indicates its age, initially set to 0. The environment is notified about the creation of the ramet using a communication module $?E$.

The subsequent productions describe the aging of spacers (p_7) and ramets (p_9). Upon reaching the maximum age $MaxAge$, a ramet is removed from the system and a message notifying the environment about this fact is sent by the plant (p_8). The death of the spacers is simulated by replacing spacer modules F with invisible line segments f of the same length. This replacement maintains the relative position of the remaining elements of the structure. Finally, production p_{11} removes communication modules after they have performed their tasks.

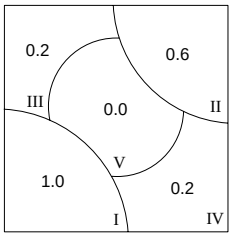


Figure 6: Division of the ground into patches

Simulations. Division of the ground into patches used in the simulations is shown in Figure 6. Arabic numerals indicate the intensity of incoming light, and Roman numerals identify each patch. The development of a clonal plant assuming this division is illustrated in Figure 7. As an extension of the basic model discussed above, the length of the spacers and the magnitude of the branching angle have been varied using random functions with a normal distribution. Ramets have been represented as trifoliate leaves.

The development begins with a single ramet located in relatively good (light intensity 0.6) patch II at the top right corner of the growth area (Figure 7, step 9 of the simulation). The plant propagates through the unfavorable patch III without producing many branches and leaves (step 26), and reaches the best patch I at the bottom left corner (step 39). After quickly spreading over this patch (step 51), the plant searches for further favorable areas (step 62). The first attempt to reach patch II fails (step 82). The plant tries again, and this time succeeds (steps 101 and 116). Light conditions in patch II are not sufficient, however, to sustain the continuous presence of the plant (step 134). The colony disappears (step 153) until the patch is reached again by a new wave of propagation (steps 161 and 182).

The sustained occupation of patch I and the repetitive invasion of patch II represent an emerging behavior of the model, difficult to predict without running simulations. Variants of this model, including other branching architectures, responses to the environment, and layouts of patches in the environment, would make it possible to analyze different foraging strategies of clonal plants. A further extension could replace the empirical assumptions regarding plant responses with a more detailed simulation of plant physiology (for example, including production of photosynthates and their transport and partition between ramets). Such physiological models could provide insight into the extent to which the foraging patterns optimize plants' access to resources [17].

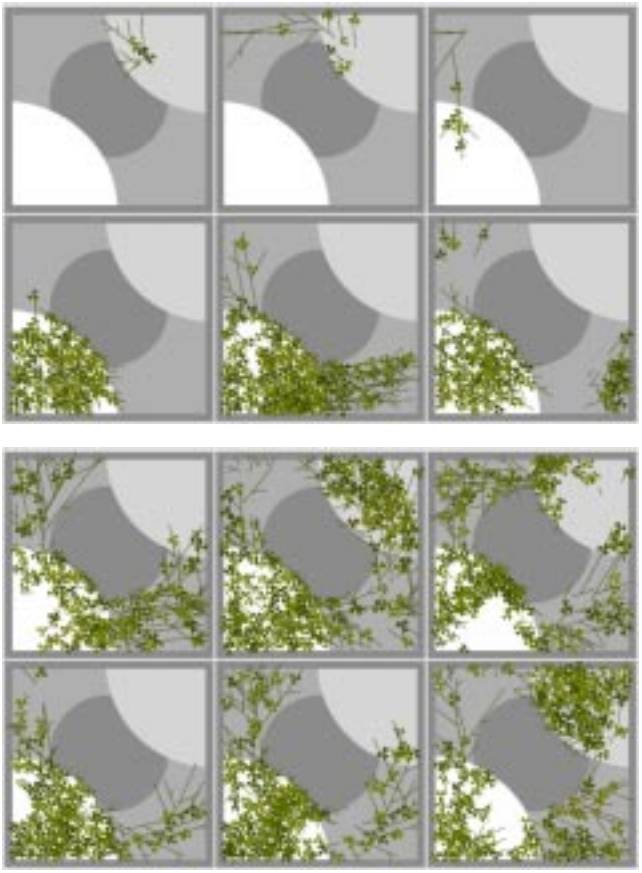


Figure 7: Development of a hypothetical clonal plant simulated using an extension of L-system 3. The individual images represent structures generated in 9, 26, 39, 51, 62, and 82 derivation steps (top), followed by structures generated in 101, 116, 134, 153, 161, and 182 steps (bottom).

7 A MODEL OF ROOT DEVELOPMENT

Background. The development of roots provides many examples of complex interactions with the environment, which involve mechanical properties, chemical reactions, and transport mechanisms in the soil. In particular, the main root and the rootlets absorb water from the soil, locally changing its concentration (volume of water per unit volume of soil) and causing water motion from water-rich to depleted regions [24]. The tips of the roots, in turn, follow the gradient of water concentration [12], thus adapting to the environment modified by their own activities.

Below we present a simplified implementation of the model of root development originally proposed by Clausnitzer and Hopmans [12]. We assume a more rudimentary mechanism of water transport, namely diffusion in a uniform medium, as suggested by Liddell and Hansen [37]. The underlying model of root architecture is similar to that proposed by Diggle [16]. For simplicity, we focus on model operation in two-dimensions.

Communication specification. The plant interacts with the environment using communication modules $?E(c, \theta)$ located at the apices of the root system. A message sent to the environment includes the turtle position $\vec{P}$, the heading vector $\vec{H}$, the value of

parameter c representing the requested (optimal) water uptake, and the value of parameter θ representing the tendency of the apex to follow the gradient of water concentration. A message returned to the plant specifies the amount of water actually received by the apex as the value of parameter c, and the angle biasing direction of further growth as the value of θ.

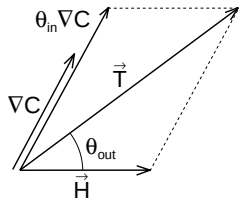


Figure 8: Definition of the biasing angle θ_{out}

The model of the environment. The environment maintains a field C of water concentrations, represented as an array of the amounts of water in square sampling areas. Water is transported by diffusion, simulated numerically using finite differencing [41]. The environment responds to a request for water from an apex located in an area (i, j) by granting the lesser of the values requested and available at that location. The amount of water in the sampled area is then decreased by the amount received by the apex. The environment also calculates a linear combination $\vec{T}$ of the turtle heading vector $\vec{H}$ and the gradient of water concentration ∇C (estimated numerically from the water concentrations in the sampled area and its neighbors), and returns an angle θ between the vectors $\vec{T}$ and $\vec{H}$ (Figure 8). This angle is used by the plant model to bias turtle heading in the direction of high water concentration.

The root model. The open L-system representing the root model makes use of arrays that specify parameters for each branching order (main axis, its daughter axes, *etc.*). The parameter values are loosely based on those reported by Clausnitzer and Hopmans [12].

```
#define N 3                          /* max. branching order + 1 */
Define: { array
Req[N] = {0.1, 0.4, 0.05},           /* requested nutrient intake */
MinReq[N] = {0.01, 0.06, 0.01},      /* minimum nutrient intake */
ElRate[N] = {0.55, 0.25, 0.55},      /* maximum elongation rate */
MaxLen[N] = {200, 5, 0.8},           /* maximum branch length */
Sens[N] = {10, 0, 0},                /* sensitivity to gradient */
Dev[N] = {30, 75, 75},               /* deviation in heading */
Del[N–1] = {30, 60},                 /* delay in branch growth */
BrAngle[N–1] = {90, 90},             /* branching angle */
BrSpace[N–1] = {1, 0.5}              /* distance between branches */
}
ω: A(0,0,0)?E(Req[0],Sens[0])

p1: A(n,s,b) > ?E(c,θ) : (s > MaxLen[n]) || (c < MinReq[n]) → ε
p2: A(n,s,b) > ?E(c,θ) :
        (n >= N–1) || (b < BrSpace[n]) {h=c/Req[n]*ElRate[n];}
        → +(nran(θ,Dev[n]))F(h) A(n,s+h,b+h)?E(Req[n],Sens[n])
p3: A(n,s,b) > ?E(c,θ) :
        (n < N–1) && (b >= BrSpace[n]) {h=c/Req[n]*ElRate[n];}
        → +(nran(θ,Dev[n]))B(n,0)F(h)
          /(180)A(n,s+h,h)?E(Req[n],Sens[n])
p4: B(n,t) : t < Del[n]  → B(n,t+1)
p5: B(n,t) : t >= Del[n]
        → [+(BrAngle[n])A(n+1,0,0)?E(Req[n+1],Sens[n+1])]
p6: ?E(c,θ)  → ε
```

The development starts with an apex A followed by a communication module $?E$. The parameters of the apex represent the branch order (0 for the main axis, 1 for its daughter axes, *etc.*), current axis length, and distance (along the axis) to the nearest branching point.

Figure 9: A two-dimensional model of a root interacting with water in soil. Background colors represent concentrations of water diffusing in soil (blue: high, black: low). The initial and boundary values have been set using a paint program.

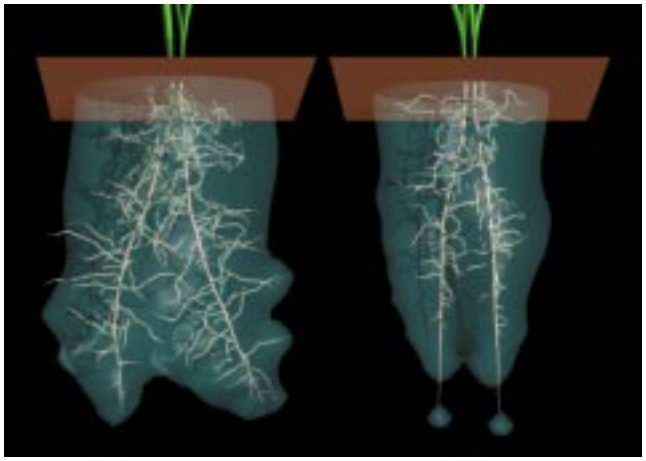


Figure 10: A three-dimensional extension of the root model. Water concentration is visualized by semi-transparent iso-surfaces [55] surrounding the roots. As a result of competition for water, the roots grow away from each other. The divergence between their main axes depends on the spread of the rootlets, which grow faster on the left then on the right.

Productions p_1 to p_3 describe possible fates of the apex as described below.

If the length s of a branch axis exceeds a predefined maximum value $MaxLen[n]$ characteristic to the branch order n, or the amount of water c received by the apex is below the required minimum $MinReq[n]$, the apex dies, terminating the growth of the axis (production p_1).

If the branch order n is equal to the maximum value assumed in the model ($N - 1$), or the distance b to the closest branching point on the axis is less than the threshold value $BrSpace[n]$, the apex adds a new segment F to the axis (production p_2). The length h of F is the product of the nominal growth increment $ElRate[n]$ and the ratio of the amount of water received by the apex c to the amount requested $Req[n]$. The new segment is rotated with respect to its predecessor by an angle $nran(\theta, Dev[n])$, where $nran$ is a random function with a normal distribution. The mean value θ, returned by the environment, biases the direction of growth towards regions of

higher water concentration. The standard deviation $Dev[n]$ characterizes the tendency of the root apex to change direction due to various factors not included explicitly in the model.

If the branch order n is less than the maximum value assumed in the model ($N-1$), and the distance b to the closest branching point on the axis is equal to or exceeds the threshold value $BrSpace[n]$, the apex creates a new branch initial B (production p_3). Other aspects of apex behavior are the same as those described by production p_2.

After the delay of $Del[n]$ steps (production p_4), the branch initial B is transformed into an apex A followed by the communication module $?E$ (production p_5), giving rise to a new root branch. Production p_6 removes communication modules that are no longer needed.

Simulations. A sample two-dimensional structure obtained using the described model is shown in Figure 9. The apex of the main axis follows the gradient of water concentration, with small deviations due to random factors. The apices of higher-order axes are not sensitive to the gradient and change direction at random, with a larger standard deviation. The absorption of water by the root and the rootlets decreases water concentration in their neighborhood; an effect that is not fully compensated by water diffusion from the water-rich areas. Low water concentration stops the development of some rootlets before they have reached their potential full length.

Figure 10 presents a three-dimensional extension of the previous model. As a result of competition for water, the main axes of the roots diverge from each other (left). If their rootlets grow more slowly, the area of influence of each root system is smaller and the main axes are closer to each other (right). This behavior is an emergent property of interactions between the root modules, mediated by the environment.

8 MODELS OF TREES CONTROLLED BY LIGHT

Background. Light is one of the most important factors affecting the development of plants. In the essentially two-dimensional structures discussed in Section 5, competition for light could be considered in a manner similar to collision detection between leaves and apices. In contrast, competition for light in three-dimensional structures must be viewed as long-range interaction. Specifically, shadows cast by one branch may affect other branches at significant distances.

The first simulations of plant development that take the local light environment into account are due to Greene [25]. He considered the entire sky hemisphere as a source of illumination and computed the amount of light reaching specific points of the structure by casting rays towards a number of points on the hemisphere. Another approach was implemented by Kanamaru *et al.* [35], who computed the amount of light reaching a given sampling point by considering it a center of projection, from which all leaf clusters in a tree were projected on a surrounding hemisphere. The degree to which the hemisphere was covered by the projected clusters indicated the amount of light received by the sampling point. In both cases, the models of plants responded to the amount and the direction of light by simulating heliotropism, which biased the direction of growth towards the vector of the highest intensity of incoming light. Subsequently, Chiba *et al.* extended the models by Kanamaru *et al.* using more involved tree models that included a mechanism simulating the flow of hypothetical endogenous information within the tree [10, 11]. A biologically better justified model, formulated in terms of production and use of photosynthates by a tree, was proposed by Takenaka [52]. The amount of light reaching leaf clusters was calculated by sampling a sky hemisphere, as in the work by Greene. Below we reproduce the main features of the Takenaka's model using the formalism of open L-systems. Depending on the underlying tree architecture, it can be applied to synthesize images of deciduous and coniferous trees. We focus on a deciduous tree, which requires a slightly smaller number of productions.

Communication specification. The plant interacts with the environment using communication modules $?E(r)$. A message sent by the plant includes turtle position $\vec{P}$, which represents the center of a spherical leaf cluster, and the value of parameter r, which represents the cluster's radius. The environment responds by setting r to the flux [14] of light from the sky hemisphere, reaching the cluster.

The model of the environment. Once all messages describing the current distribution of leaves on a tree have been received, the environmental process computes the extent of the tree in the x, y, and z directions, encompasses the tree in a tight grid ($32 \times 32 \times 32$ voxels in our simulations), and allocates leaf clusters to voxels to speed up further computations. The environmental process then estimates the light flux Φ from the sky hemisphere reaching each cluster (shadows cast by the branches are ignored). To this end, the hemisphere is represented by a set of directional light sources S (9 in the simulations). The flux densities (radiosities) B of the sources approximate the non-uniform distribution of light from the sky (*cf.* [52]). For each leaf cluster L_i and each light source S, the environment determines the set of leaf clusters L_j that may shade L_i. This is achieved by casting a ray from the center of L_i in the direction of S and testing for intersections with other clusters (the grid accelerates this process). In order not to miss any clusters that may partially occlude L_i, the radius of each cluster L_j is increased by the maximum value of cluster radius r_{max}.

To calculate the flux reaching cluster L_i, this cluster and all clusters L_j that may shade it according to the described tests are projected on a plane P perpendicular to the direction of light from the source S. The impact of a cluster L_j on the flux Φ reaching cluster L_i is then computed according to the formula:

$$\Phi = (A_i - A_{ij})B + A_{ij}\tau B$$

where A_i is the area of the projection of L_i on P, A_{ij} is the area of the intersection between projections of L_i and L_j, and τ is the light transmittance through leaf cluster L_j (equal to 0.25 in the simulations). If several clusters L_j shade L_i, their influences are multiplied. The total flux reaching cluster L_i is calculated as the sum of the fluxes received from each light source S.

The plant model. In addition to the communication module $?E$, the plant model includes the following types of modules:

- Apex $A(vig, del)$. Parameter vig represents vigor, which determines the length of branch segments (internodes) and the diameter of leaf clusters produced by the apex. Parameter del is used to introduce a delay, needed for propagating products of photosynthesis through the tree structure between consecutive stages of development (years).

- Leaf $L(vig, p, age, del)$. Parameters denote the leaf radius vig, the amount of photosynthates produced in unit time according to the leaf's exposure to light p, the number of years for which a leaf has appeared at a given location age, and the delay del, which plays the same role as in the apices.

- Internode $F(vig)$. Consistent with the turtle interpretation, the parameter vig indicates the internode length.

- Branch width symbol $!(w, p, n)$, also used to carry the endogenous information flow. The parameters determine: the width of the following internode w, the amount of photosynthates reaching the symbol's location p, and the number of terminal branch segments above this location n.

The corresponding L-system is given below.

```
#define φ  137.5      /* divergence angle */
#define α0  5         /* direction change - no branching */
#define α1  20        /* branching angle - main axis */
#define α2  32        /* branching angle - lateral axis */
#define W  0.02       /* initial branch width */
#define VD  0.95      /* apex vigor decrement */
#define Del  30       /* delay */
#define LS  5         /* how long a leaf stays */
#define LP  8         /* full photosynthate production */
#define LM  2         /* leaf maintenance */
#define PB  0.8       /* photosynthates needed for branching */
#define PG  0.4       /* photosynthates needed for growth */
#define BM  0.32      /* branch maintenance coefficient */
#define BE  1.5       /* branch maintenance exponent */
#define N_min  25     /* threshold for shedding */
Consider:  ?E[]!L     /* for context matching */
ω:   !(W,1,1)F(2)L(1,LP,0,0)A(1,0)[!(0,0,0)]!(W,0,1)

p1:  A(vig,del) : del<Del  → A(vig,del+1)
p2:  L(vig,p,age,del) : (age<LS)&&(del<Del–1)  → L(vig,p,age,del+1)
p3:  L(vig,p,age,del) : (age<LS)&&(del==Del–1)
                  → L(vig,p,age,del+1)?E(vig*0.5)
p4:  L(vig,p,age,del) > ?E(r) : (age<LS) && (r*LP>=LM)
                  && (del == Del) → L(vig,LP*r–LM,age+1,0)
p5:  L(vig,p,age,del) > ?E(r) : ((age == LS)||(r*LP<=LM))
                  && (del == Del) → L(0,0,LS,0)
p6:  ?E(r) < A(vig,del) : r*LP–LM>PB {vig=vig*VD;}
                  → /(φ)[+(α2)!(W,–PB,1)F(vig)L(vig,LP,0,0)A(vig,0)
                        [!(0,0,0)]!(W,0,1)]
                    –(α1)!(W,0,1)F(vig)L(vig,LP,0,0)/A(vig,0)
p7:  ?E(r) < A(vig,del) : r*LP–LM > PG {vig=vig*VD;}
                  → /(φ)–(α0)[!(0,0,0)]
                    !(W,–PG,1)F(vig)L(vig,LP,0,0)A(vig,0)
p8:  ?E(r) < A(vig,del) : r*LP–LM <= PG  → A(vig,0)
p9:  ?E(r)  → ε
p10: !(w0,p0,n0) > L(vig,pL,age,del) [!(w1,p1,n1)]!(w2,p2,n2) :
                  {w=(w1^2+w2^2)^0.5; p=p1+p2+pL–BM*(w/W)^BE;}
                  (p>0) || (n1+n2 >=N_min)  → !(w,p,n1+n2)
p11: !(w0,p0,n0) > L(vig,pL,age,del) [!(w1,p1,n1)]!(w2,p2,n2)
                  → !(w0,0,0)L%
```

The simulation starts with a structure consisting of a branch segment F, supporting a leaf L and an apex A (axiom ω). The first branch width symbol ! defines the segment width. Two additional symbols ! following the apex create "virtual branches," needed to provide proper context for productions p_{10} and p_{11}. The tree grows in stages, with the delay of $Del + 1$ derivation steps between consecutive stages introduced by production p_1 for the apices and p_2 for the leaves. Immediately before each new growth stage, communication symbols are introduced to inform the environment about the location and size of the leaf clusters (p_3). If the flux r returned by the environment results in the production of photosynthates $r * LP$ exceeding the amount LM needed to maintain a cluster, it remains in the structure (p_4). Otherwise it becomes a liability to the tree and dies (p_5). Another condition to production p_5 prevents a leaf from occupying the same location for more than LS years.

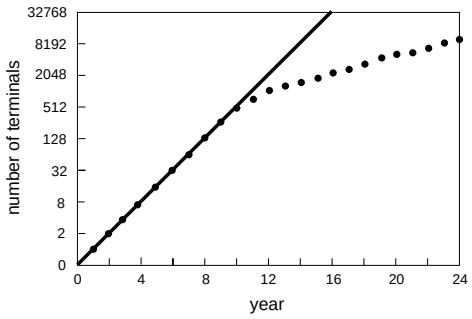


Figure 11: The number of terminal branch segments resulting from unrestricted bifurcation of apices (continuous line), compared to the number of segments generated in a simulation (isolated points)

The flux r also determines the fate of the apex, captured by productions p_6 to p_8. If the amount of photosynthates $r * LP - LM$ transported from the nearby leaf exceeds a threshold value PB, the apex produces two new branches (p_6). The second parameter in the first branch symbol ! is set to $-PB$, to subtract the amount of photosynthates used for branching from the amount that will be transported further down. The length of branch segments vig is reduced with respect to the mother segment by a predefined factor VD, reflecting a gradual decrease in the vigor of apices with age. The branch width modules ! following the first apex A are introduced to provide context required by productions p_{10} and p_{11}, as in the axiom.

If the amount of photosynthates $r * LP - LM$ transported from the leaf is insufficient to produce new branches, but above the threshold PG, the apex adds a new segment F to the current branch axis without creating a lateral branch (p_7). Again, a virtual branch containing the branch width symbol ! is being added to provide context for productions p_{10} and p_{11}.

If the amount of photosynthates is below PG, the apex remains dormant (p_8). Communication modules no longer needed are removed from the structure (p_9).

Production p_{10} captures the endogenous information flow from leaves and terminal branch segments to the base of the tree. First, it determines the radius w of the mother branch segment as a function of the radii w_1 and w_2 of the supported branches:

$$w = \sqrt{w_1^2 + w_2^2}.$$

Thus, a cross section of the mother segment has an area equal to the sum of cross sections of the supported segments, as postulated in the literature [40, 46]. Next, production p_{10} calculates the flow p of photosynthates into the mother segment. It is defined as the sum of the flows p_L, p_1 and p_2 received from the associated leaf L and from both daughter branches, decreased by the amount $BM * (w/W)^{BE}$ representing the cost of maintaining the mother segment. Finally, production p_10 calculates the number of terminal branch segments n supported by the mother segment as the sum of the numbers of terminal segments supported by the daughter branches, n_1 and n_2.

Production p_{10} takes effect if the flow p is positive (the branch is not a liability to the tree), or if the number n of supported terminals is above a threshold N_{min}. If these conditions are not satisfied,

Figure 12: A tree model with branches competing for access to light, shown without the leaves

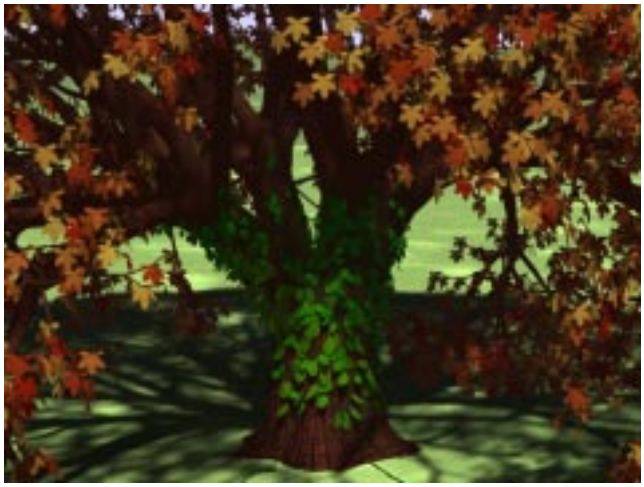


Figure 13: A climbing plant growing on the tree from the previous figure

production p_{11} removes (sheds) the branch from the tree using the cut symbol %.

Simulations. The competition for light between tree branches is manifested by two phenomena: reduced branching or dormancy of apices in unfavorable local light conditions, and shedding of

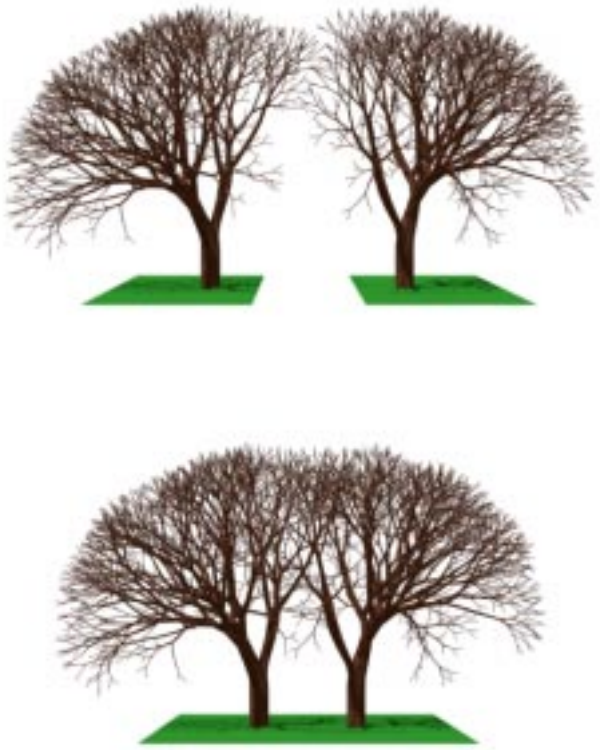


Figure 14: A model of deciduous trees competing for light. The trees are shown in the position of growth (top) and moved apart (bottom) to reveal the adaptation of crown geometry to the presence of the neighbor tree.

branches which do not receive enough light to contribute to the whole tree. Both phenomena limit the extent of branching, thus controlling the density of the crown. This property of the model is supported by the simulation results shown in Figure 11. If the growth was unlimited (production p_6 was always chosen over p_7 and p_8), the number of terminal branch segments would double every year. Due to the competition for light, however, the number of terminal segments observed in an actual simulation increases more slowly. For related statistics using a different tree architecture see [52].

A tree image synthesized using an extension of the presented model is shown in Figure 12. The key additional feature is a gradual reduction of the branching angle of a young branch whose sister branch has been shed. As the result, the remaining branch assumes the role of the leading shoot, following the general growth direction of its supporting segment. Branch segments are represented as texture-mapped generalized cylinders, smoothly connected at the branching points (*cf.* [6]). The bark texture was created using a paint program.

As an illustration of the flexibility of the modeling framework presented in this paper, Figure 13 shows the effect of seeding a hypothetical climbing plant near the same tree. The plant follows the surface of the tree trunk and branches, and avoids excessively dense colonization of any particular area. Thus, the model integrates sev-

Figure 15: A model of coniferous trees competing for light. The trees are shown in the position of growth (top) and moved apart (bottom).

eral environmentally-controlled phenomena: the competition of tree branches for light, the following of surfaces by a climbing plant, and the prevention of crowding as discussed in Section 6. Leaves were modeled using cubic patches (*cf.* [46]).

In the simulations shown in Figure 14 two trees described by the same set of rules (younger specimens of the tree from Figure 12) compete for light from the sky hemisphere. Moving the trees apart after they have grown reveals the adaptation of their crowns to the presence of the neighbor tree. This simulation illustrates both the necessity and the possibility of incorporating the adaptive behavior into tree models used for landscape design purposes.

The same phenomenon applies to coniferous trees, as illustrated in Figure 15. The tree model is similar to the original model by Takenaka [52] and can be viewed as consisting of approximately horizontal tiers (as discussed in Section 5) produced in sequence by the apex of the tree stem. The lower tiers are created first and therefore potentially can spread more widely then the younger tiers higher up (the *phase effect* [46]). This pattern of development is affected by the presence of the neighboring tree: the competition for light prevents the crowns from growing into each other.

The trees in Figure 15 retain branches that do not receive enough light. In contrast, the trees in the stand presented in Figure 16 shed branches that do not contribute photosynthates to the entire tree,

Figure 16: Relationship between tree form and its position in a stand.

using the same mechanism as described for the deciduous trees. The resulting simulation reveals essential differences between the shape of the tree crown in the middle of a stand, at the edge, or at the corner. In particular, the tree in the middle retains only the upper part of its crown. In lumber industry, the loss of lower branches is usually a desirable phenomenon, as it reduces knots in the wood and the amount of cleaning that trees require before transport. Simulations may assist in choosing an optimal distance for planting trees, where self-pruning is maximized, yet there is sufficient space between trees too allow for unimpeded growth of trunks in height and diameter.

9 CONCLUSIONS

In this paper, we introduced a framework for the modeling and visualization of plants interacting with their environment. The essential elements of this framework are:

- a system design, in which the plant and the environment are treated as two separate processes, communicating using a standard interface, and
- the language of open L-systems, used to specify plant models that can exchange information with the environment.

We demonstrated the operation of this framework by implementing models that capture collisions between branches, the propagation of clonal plants, the development of roots in soil, and the development of tree crowns competing for light. We found that the proposed framework makes it possible to easily create and modify models spanning a wide range of plant structures and environmental processes. Simulations of the presented phenomena were fast enough to allow interactive experimentation with the models (Table 1).

There are many research topics that may be addressed using the simulation and visualization capabilities of the proposed framework. They include, for instance:

- Fundamental analysis of the role of different forms of information flow in plant morphogenesis (in particular, the relationship between endogenous and exogenous flow). This is a continuation

Fig.	Number of		Derivation		Time[a]	
	branch segments	leaf clusters	steps	yrs	sim.	render.
5	138	140	5	5	1 s	1 s
7	786	229	182	NA	50 s	2 s
9	4194	34[b]	186	NA	67 s	3 s
10	37228	448[b]	301	NA	15 min	70 s
12	22462	19195	744	24	22 min	13 s[c]
15	13502	3448	194	15	4 min	8 s[d]

[a] Simulation and rendering using OpenGL on a 200MHz/64MB Indigo^2 Extreme
[b] active apices
[c] without generalized cylinders and texture mapping
[d] branching structure without needles

Table 1: Numbers of primitives and simulation/rendering times for generating and visualizing selected models

of the research pioneered by Bell [4] and Honda *et al.* [7, 33].

- Development of a comprehensive plant model describing the cycling of nutrients from the soil through the roots and branches to the leaves, then back to the soil in the form of substances released by fallen leaves.
- Development of models of specific plants for research, crop and forest management, and for landscape design purposes. The models may include environmental phenomena not discussed in this paper, such as the global distribution of radiative energy in the tree crowns, which affects the amount of light reaching the leaves and the local temperature of plant organs.

The presented framework itself is also open to further research. To begin, the precise functional specification of the environment, implied by the design of the modeling framework, is suitable for a formal analysis of algorithms that capture various environmental processes. This analysis may highlight tradeoffs between time, memory, and communication complexity, and lead to programs matching the needs of the model to available system resources in an optimal manner.

A deeper understanding of the spectrum of processes taking place in the environment may lead to the design of a mini-language for environment specification. Analogous to the language of L-systems for plant specification, this mini-language would simplify the modeling of various environments, relieving the modeler from the burden of low-level programming in a general-purpose language. Fleischer and Barr's work on the specification of environments supporting collisions and reaction-diffusion processes [20] is an inspiring step in this direction.

Complexity issues are not limited to the environment, but also arise in plant models. They become particularly relevant as the scope of modeling increases from individual plants to groups of plants and, eventually, entire plant communities. This raises the problem of selecting the proper level of abstraction for designing plant models, including careful selection of physiological processes incorporated into the model and the spatial resolution of the resulting structures.

The complexity of the modeling task can be also addressed at the level of system design, by assigning various components of the model (individual plants and aspects of the environment) to different components of a distributed computing system. The communication structure should then be redesigned to accommodate information transfers between numerous processes within the system.

In summary, we believe that the proposed modeling methodology and its extensions will prove useful in many applications of plant modeling, from research in plant development and ecology to landscape design and realistic image synthesis.

Acknowledgements

We would like to thank Johannes Battjes, Campbell Davidson, Art Diggle, Heinjo During, Michael Guzy, Naoyoshi Kanamaru, Bruno Moulia, Zbigniew Prusinkiewicz, Bill Remphrey, David Reid, and Peter Room for discussions and pointers to the literature relevant to this paper. We would also like to thank Bruno Andrieu, Mark Hammel, Jim Hanan, Lynn Mercer, Chris Prusinkiewicz, Peter Room, and the anonymous referees for helpful comments on the manuscript. Most images were rendered using the ray tracer `rayshade` by Craig Kolb. This research was sponsored by grants from the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] AGRAWAL, P. The cell programming language. *Artificial Life 2*, 1 (1995), 37–77.

[2] ARVO, J., AND KIRK, D. Modeling plants with environment-sensitive automata. In *Proceedings of Ausgraph'88* (1988), pp. 27 – 33.

[3] BELL, A. *Plant form: An illustrated guide to flowering plants.* Oxford University Press, Oxford, 1991.

[4] BELL, A. D. The simulation of branching patterns in modular organisms. *Philos. Trans. Royal Society London, Ser. B 313* (1986), 143–169.

[5] BELL, A. D., ROBERTS, D., AND SMITH, A. Branching patterns: the simulation of plant architecture. *Journal of Theoretical Biology 81* (1979), 351–375.

[6] BLOOMENTHAL, J. 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.

[7] BORCHERT, R., AND HONDA, H. Control of development in the bifurcating branch system of *Tabebuia rosea*: A computer simulation. *Botanical Gazette 145*, 2 (1984), 184–195.

[8] BORCHERT, R., AND SLADE, N. Bifurcation ratios and the adaptive geometry of trees. *Botanical Gazette 142*, 3 (1981), 394–401.

[9] CHEN, S. G., CEULEMANS, R., AND IMPENS, I. A fractal based *Populus* canopy structure model for the calculation of light interception. *Forest Ecology and Management* (1993).

[10] CHIBA, N., OHKAWA, S., MURAOKA, K., AND MIURA, M. Visual simulation of botanical trees based on virtual heliotropism and dormancy break. *The Journal of Visualization and Computer Animation 5* (1994), 3–15.

[11] CHIBA, N., OHSHIDA, K., MURAOKA, K., MIURA, M., AND SAITO, N. A growth model having the abilities of growth-regulations for simulating visual nature of botanical trees. *Computers and Graphics 18*, 4 (1994), 469–479.

[12] CLAUSNITZER, V., AND HOPMANS, J. Simultaneous modeling of transient three-dimensional root growth and soil water flow. *Plant and Soil 164* (1994), 299–314.

[13] COHEN, D. Computer simulation of biological pattern generation processes. *Nature 216* (October 1967), 246–248.

[14] COHEN, M., AND WALLACE, J. *Radiosity and realistic image synthesis*. Academic Press Professional, Boston, 1993. With a chapter by P. Hanrahan and a foreword by D. Greenberg.

[15] DE REFFYE, P., HOULLIER, F., BLAISE, F., BARTHELEMY, D., DAUZAT, J., AND AUCLAIR, D. A model simulating above- and below-ground tree architecture with agroforestry applications. *Agroforestry Systems 30* (1995), 175–197.

[16] DIGGLE, A. J. ROOTMAP - a model in three-dimensional coordinates of the structure and growth of fibrous root systems. *Plant and Soil 105* (1988), 169–178.

[17] DONG, M. *Foraging through morphological response in clonal herbs*. PhD thesis, Univeristy of Utrecht, October 1994.

[18] FISHER, J. B. How predictive are computer simulations of tree architecture. *International Journal of Plant Sciences 153 (Suppl.)* (1992), 137–146.

[19] FISHER, J. B., AND HONDA, H. Computer simulation of branching pattern and geometry in *Terminalia* (Combretaceae), a tropical tree. *Botanical Gazette 138*, 4 (1977), 377–384.

[20] FLEISCHER, K. W., AND BARR, A. H. A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In *Artificial Life III*, C. G. Langton, Ed. Addison-Wesley, Redwood City, 1994, pp. 389–416.

[21] FORD, E. D., AVERY, A., AND FORD, R. 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* (1990), 15–36.

[22] FORD, H. Investigating the ecological and evolutionary significance of plant growth form using stochastic simulation. *Annals of Botany 59* (1987), 487–494.

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

[24] GARDNER, W. R. Dynamic aspects of water availability to plants. *Soil Science 89*, 2 (1960), 63–73.

[25] GREENE, N. 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.

[26] GREENE, N. Detailing tree skeletons with voxel automata. SIGGRAPH '91 Course Notes on Photorealistic Volume Modeling and Rendering Techniques, 1991.

[27] GUZY, M. R. A morphological-mechanistic plant model formalized in an object-oriented parametric L-system. Manuscript, USDA-ARS Salinity Laboratory, Riverside, 1995.

[28] HANAN, J. Virtual plants — Integrating architectural and physiological plant models. In *Proceedings of ModSim 95* (Perth, 1995), P. Binning, H. Bridgman, and B. Williams, Eds., vol. 1, The Modelling and Simulation Society of Australia, pp. 44–50.

[29] HANAN, J. S. *Parametric L-systems and their application to the modelling and visualization of plants*. PhD thesis, University of Regina, June 1992.

[30] HART, J. W. *Plant tropisms and other growth movements*. Unwin Hyman, London, 1990.

[31] HERMAN, G. T., AND ROZENBERG, G. *Developmental systems and languages*. North-Holland, Amsterdam, 1975.

[32] HONDA, H. 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* (1971), 331–338.

[33] HONDA, H., TOMLINSON, P. B., AND FISHER, J. B. Computer simulation of branch interaction and regulation by unequal flow rates in botanical trees. *American Journal of Botany 68* (1981), 569–585.

[34] KAANDORP, J. *Fractal modelling: Growth and form in biology*. Springer-Verlag, Berlin, 1994.

[35] KANAMARU, N., CHIBA, N., TAKAHASHI, K., AND SAITO, N. CG simulation of natural shapes of botanical trees based on heliotropism. *The Transactions of the Institute of Electronics, Information, and Communication Engineers J75-D-II*, 1 (1992), 76–85. In Japanese.

[36] KURTH, W. *Growth grammar interpreter GROGRA 2.4: A software tool for the 3-dimensional interpretation of stochastic, sensitive growth grammars in the context of plant modeling. Introduction and reference manual*. Forschungszentrum Waldökosysteme der Universität Göttingen, Göttingen, 1994.

[37] LIDDELL, C. M., AND HANSEN, D. Visualizing complex biological interactions in the soil ecosystem. *The Journal of Visualization and Computer Animation 4* (1993), 3–12.

[38] LINDENMAYER, A. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology 18* (1968), 280–315.

[39] LINDENMAYER, A. Developmental systems without cellular interaction, their languages and grammars. *Journal of Theoretical Biology 30* (1971), 455–484.

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

[41] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical recipes in C: The art of scientific computing. Second edition*. Cambridge University Press, Cambridge, 1992.

[42] PRUSINKIEWICZ, P. Visual models of morphogenesis. *Artificial Life 1*, 1/2 (1994), 61–74.

[43] PRUSINKIEWICZ, P., HAMMEL, M., HANAN, J., AND MĚCH, R. Visual models of plant development. In *Handbook of formal languages*, G. Rozenberg and A. Salomaa, Eds. Springer-Verlag, Berlin, 1996. To appear.

[44] PRUSINKIEWICZ, P., AND HANAN, J. L-systems: From formalism to programming languages. In *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, G. Rozenberg and A. Salomaa, Eds. Springer-Verlag, Berlin, 1992, pp. 193–211.

[45] PRUSINKIEWICZ, P., JAMES, M., AND MĚCH, R. Synthetic topiary. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994), pages 351–358, ACM SIGGRAPH, New York, 1994.

[46] PRUSINKIEWICZ, P., AND LINDENMAYER, A. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[47] ROOM, P. M. 'Falling apart' as a lifestyle: the rhizome architecture and population growth of *Salvinia molesta*. *Journal of Ecology 71* (1983), 349–365.

[48] ROOM, P. M., MAILLETTE, L., AND HANAN, J. Module and metamer dynamics and virtual plants. *Advances in Ecological Research 25* (1994), 105–157.

[49] ROZENBERG, G. T0L systems and languages. *Information and Control 23* (1973), 357–381.

[50] SACHS, T., AND NOVOPLANSKY, A. Tree from: Architectural models do not suffice. *Israel Journal of Plant Sciences 43* (1995), 203–212.

[51] SIPPER, M. Studying artificial life using a simple, general cellular model. *Artificial Life 2*, 1 (1995), 1–35.

[52] TAKENAKA, A. A simulation model of tree architecture development based on growth response to local light environment. *Journal of Plant Research 107* (1994), 321–330.

[53] ULAM, S. On some mathematical properties connected with patterns of growth of figures. In *Proceedings of Symposia on Applied Mathematics* (1962), vol. 14, American Mathematical Society, pp. 215–224.

[54] WEBER, J., AND PENN, J. Creation and rendering of realistic trees. Proceedings of SIGGRAPH '95 (Los Angeles, California, August 6–11, 1995), pages 119–128, ACM SIGGRAPH, New York, 1995.

[55] WYVILL, G., MCPHEETERS, C., AND WYVILL, B. Data structure for soft objects. *The Visual Computer 2*, 4 (February 1986), 227–234.

The use of positional information in the modeling of plants

Przemyslaw Prusinkiewicz, Lars Mündermann, Radoslaw Karwowski, Brendan Lane

Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4
pwp|lars|radekk|laneb@cpsc.ucalgary.ca

Abstract

We integrate into plant models three elements of plant representation identified as important by artists: posture (manifested in curved stems and elongated leaves), gradual variation of features, and the progression of the drawing process from overall silhouette to local details. The resulting algorithms increase the visual realism of plant models by offering an intuitive control over plant form and supporting an interactive modeling process. The algorithms are united by the concept of expressing local attributes of plant architecture as functions of their location along the stems.

CR categories: F.4.2 **[Mathematical Logic and Formal Languages]**: Grammars and Other Rewriting Systems, I.3.5 **[Computer Graphics]**: Computational Geometry and Object Modeling, J.3 **[Life and Medical Sciences]**: Biology.

Keywords: realistic image synthesis, interactive procedural modeling, plant, positional information, phyllotaxis, Chomsky grammar, L-system, differential turtle geometry, generalized cylinder.

1 Introduction

Forward simulation of development is a well established paradigm for modeling plants. It underlies, for example, the AMAP simulation software [9] and modeling methods based on L-systems [28]. In both cases, a plant is modeled using a set of rules that describe the emergence and growth of individual plant components. The simulation program traces their fate over time, and integrates them into the structure of the whole plant.

Over the years, the simulation paradigm has been extended to include a wide range of interactions between plants and their environments [15, 21]. The resulting models have gained acceptance as a research tool in biology and have led to increasingly convincing visualizations. In image synthesis applications, however, the simulation-based approach has several drawbacks:

- Visual realism of the models is linked to the biological and physical accuracy of simulations. This requires the modeler to have a good understanding of the underlying processes, makes comprehensive models complicated, and results in long simulation times.

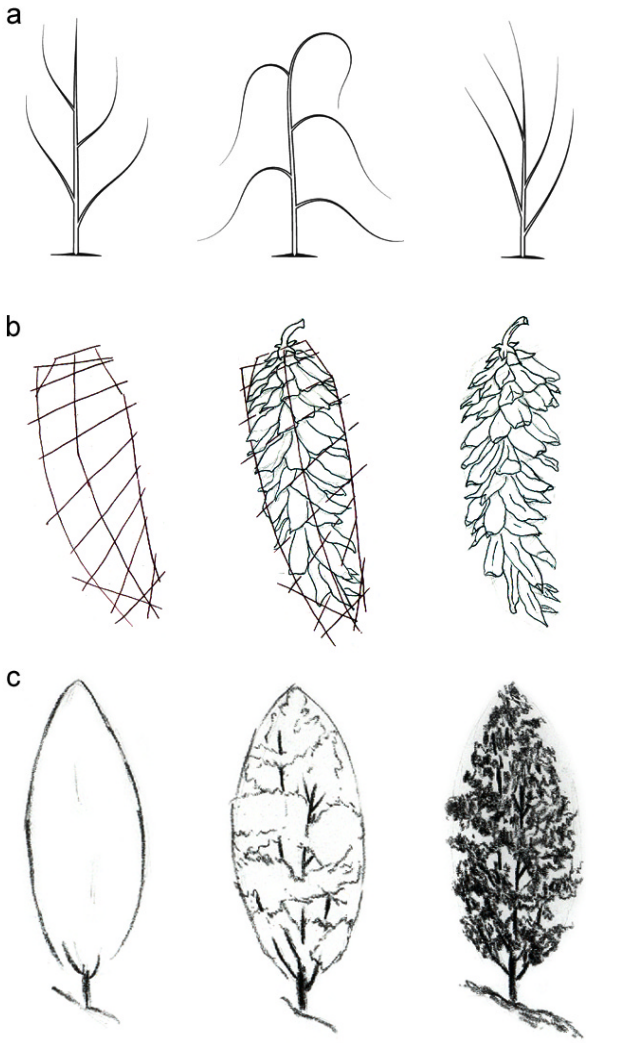


Figure 1: Selected elements of the artistic representation of plants: (a) posture, (b) regular arrangement and gradual variation of organs along an axis, and (c) progression from silhouette to detail in the drawing process. Figure (a) is based on [37, page 41], (b) is redrawn from [38, page 68], and (c) is redrawn from [24, page 13].

- Global characteristics of plant appearance, such as the curvature of plant axes, the density of organ distribution, and the overall silhouette of the plant, are emergent properties of the models and therefore are difficult to control.

Methods for creating visually realistic representations of plants

ACM SIGGRAPH 2001 (Los Angeles, California, August12-17, 2001), pp. 289 - 300.

have long been understood by artists (Figure 1). Important elements include plant *posture*, defined by the angles of insertion and curvature of organs, and the *arrangement* and gradual *variation* of organs on their supporting stems. The drawing process progresses in a *global-to-local* fashion, from silhouette to detail. Inspired by the quality of botanical illustrations, we have developed a plant modeling method that supports similar elements and processes. The proposed method *inverts* the local-to-global operation of simulation-based models by progressing from global plant characteristics specified by the user to algorithmically generated details. The algorithms are united by their use of *positional information*, which we define as the position of plant components along the *axes* of their supporting stems or branches. User-defined functions map this information to *morphogenetic gradients* [2], which describe the distributions of features along the axes.

The notions of positional information and morphogenetic gradients unify and generalize several plant-modeling concepts that have already appeared in botanical and computer graphics literature. Following their review (Section 2), we outline our modeling software environment, focusing on the language that we use to formally describe the algorithms and models (Section 3). We then develop the mathematical foundations of plant modeling based on positional information: the modeling and framing of individual axes (Section 4), and their partitioning into internodes (Section 5). In Section 6, we present the resulting modeling method from the modeler's perspective, and illustrate its applications using plants and plant structures organized along a single axis. In Section 7, we address the important special case of organ arrangement in closely packed spiral phyllotactic patterns. Finally, in Section 8, we extend the proposed modeling method to plants with higher-order branches, including trees. We conclude the paper with a discussion of the results, applications, and problems open for further research (Section 9). Proofs of selected mathematical results pertinent to the use of positional information in the modeling of plants are presented in the Appendices.

2 Previous work

Applications of positional information have their origins in early descriptive plant models created by biologists: the poplar model by Burk *et al.* [7] and the larch sapling model by Remphrey and Powell [30]. In both cases, the length of lateral branches was expressed as a function of their position on the main stem. The models were visualized as two-dimensional line drawings.

In computer graphics, a related concept was first applied to model trees by Reeves and Blau [29], who expressed the length of first-order branches as the distance from the branching point to a user-specified surface defining the silhouette of the tree. Higher-order branches were generated algorithmically, with "many parameters inherited from the parent."

A more elaborated model was introduced by Weber and Penn [36]. They characterized a tree using several positional functions, and pointed to an advantage of this technique: "Since our parameters can address the character of an entire stem and not just its segment-to-segment nature, we allow users to make changes on a level they can more easily understand and visualize."

Lintermann and Deussen incorporated positional information into their interactive plant modeling program `xfrog` [18, 19]. The position of a sample point along an axis may affect the length of an internode, the length of a branch, the magnitude of a branching angle, and other attributes. Functions that map positions to attribute values

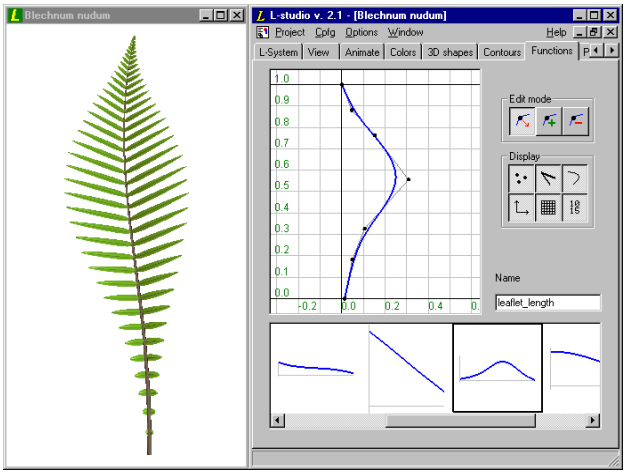


Figure 2: A snapshot of the L-studio/cpfg screen. The model can be manipulated using textual and graphical editors displayed on the right side of the screen. In this example, the outline of the fishbone water fern leaf (*Blechnum nudum*) is being defined using a graphical function editor. A *gallery* under the editor's window provides access to various functions used in this model. The second row of tabs near the top of the screen makes it possible to select other editors, such as the textual editor of the L-system that has been used to specify the algorithmic structure of this model.

can be specified graphically, by editing function plots, or textually, by editing algebraic expressions. The authors did not describe in detail the algorithms underlying their software, but experience with `xfrog` was an inspiration for our work. From the user interface perspective, the editing of function plots is an extension of the interactive manipulation of plant parameters using sliders [23, 28].

3 The modeling environment

We have adapted the L-system-based modeling software `L-studio/cpfg` [27] to the needs of modeling using positional information. A screenshot of the system in operation is shown in Figure 2.

An L-studio model consists of two basic components: a description of a generative algorithm in the `cpfg` modeling language [25], and a set of graphically defined entities. These entities can be defined and manipulated using the L-studio *function*, *curve*, *surface* and *material* editors [27], or imported from external sources.

The fundamental constructs of the `cpfg` language are *rewriting rules*, or *productions*. The program supports both parallel application of productions, characteristic of L-systems [28], and sequential application of productions, characteristic of Chomsky grammars [8]. In the context of plant modeling, these formalisms compare as follows.

L-system productions capture the *development* of plant components over time. For example, the division of a mother cell A into two daughter cells B and C can be described by the production $A \rightarrow BC$. In the case of multicellular organisms, L-system productions are applied in *parallel* to advance time consistently in all cells. The simulation is completed when the organism reaches a predefined *terminal age*, corresponding to a given number of derivation steps.

Chomsky grammars, in contrast, characterize the *structure* of plants, that is, the distribution of their features and components in space. The fact that organism A consists of parts B and C can again be expressed by a production, for example $A \rightsquigarrow BC$, but such a *decomposition rule* has a different meaning and functions in a different way than its L-system counterpart. Since non-overlapping substructures can be partitioned independently from each other, the decomposition rules may be applied sequentially. Furthermore, the appropriate condition for terminating a decomposition process is the reaching of *terminal symbols*, which represent components that cannot be divided further.

Our intended use of positional information is to capture the distribution of plant features and components in space. Consequently, the meaning and formal properties of productions used in this paper correspond with the definition of Chomsky grammars. In the big picture of a complete plant modeling software design, the switch from L-systems to Chomsky grammars amounts to a relatively minor modification of the code. Consequently, our modeling language, outlined below, expands, rather than replaces, features of the earlier purely L-system-based implementations of the `cpfg` modeling language [13, 28].

As in the case of L-systems, a branching structure is represented by a *bracketed string* of *modules* (symbols with associated parameters). Matching pairs of brackets enclose branches. Derivation begins with an initial string identified by the keyword `axiom`. *Context-free* productions are specified using the syntax

$$pred : \{block_1\} \;\; cond \;\; \{block_2\} \rightsquigarrow succ, \qquad (1)$$

where $pred$ is the predecessor (a single module), and $succ$ is the successor (a bracketed string of modules) [25]. The optional field $cond$ is the condition (logical expression) that guards production application. Fields $block_1$ and $block_2$ are sequences of C statements. The first block is executed before the evaluation of the condition. If the condition is true, the second block is also evaluated and the production is applied. For example, the rule

$$A(x) : \{y = x + 2;\} \;\; y \geq 5 \;\; \{z = y/3;\} \rightsquigarrow B(z)C(z+1) \quad (2)$$

can be applied to module A(4), subdividing it into modules $B(2)C(3)$.

The `cpfg` language also supports *context-sensitive* productions, in which the strict predecessor (module being replaced) $pred$ may be preceded by one or more modules constituting the *left context*, and/or followed by modules constituting the *right context*. These contexts are separated from the strict predecessor by symbols $<$ and $>$ respectively. For example, production

$$A(x) < B(y) > C(z) : x + z > 0 \rightsquigarrow M(y/2)N(y/2) \qquad (3)$$

decomposes module B into a pair of modules M and N, provided that module B appears in the context of modules A and C, and the sum of their parameters is greater then 0. In the scope of this paper, context is limited to query symbols, discussed later on.

In order to conveniently specify morphogenetic gradients inherent in the use of positional information, we have extended the `cpfg` modeling language with function calls of the form $\mathrm{func}(id, x)$. The integer number id is the identifier of a planar B-spline curve and the real number x is the function argument. Function plots are manipulated using the *interactive function editor* (Figure 2). It constrains the motion of the control points that define the function plots to guarantee that they assign a unique value y to each argument x.

The modeling language also supports function calls of the form $\mathrm{curveX}(id, s)$, $\mathrm{curveY}(id, s)$, $\mathrm{curveZ}(id, s)$, and $\mathrm{tanX}(id, s)$, $\mathrm{tanY}(id, s)$, $\mathrm{tanZ}(id, s)$, where id is the identifier of an arbitrary B-spline curve. These calls return coordinates of a point on the curve id and of the tangent vector at this point, given the arc-length distance s from the curve origin. The call $\mathrm{curveLen}(id)$ returns the total length of the curve.

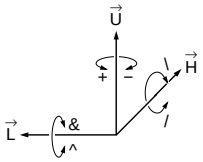


Figure 3: Controlling the turtle in three dimensions.

To create a graphical model, the derived string is scanned sequentially and reserved modules are interpreted as commands to a LOGO-style turtle [28]. At any point within the string, the *turtle state* is characterized by a position vector $\vec{P}$ and three mutually perpendicular orientation vectors $\vec{H}$, $\vec{L}$, and $\vec{U}$ that indicate the turtle's *heading*, the direction to the *left*, and the *up* direction (Figure 3). The coordinates of these vectors can be accessed using *query* modules of the form $?X(x, y, z)$, where X is the vector to be accessed, one of P, H, L, or U [26]. Module F causes the turtle to draw a line in the current direction, while modules f causes the turtle to move without drawing a line. Modules $+$, $-$, $\&$, $\wedge$, $/$, and $\backslash$ rotate the turtle around one of the vectors $\vec{H}$, $\vec{L}$, or $\vec{U}$, as shown in Figure 3. Many symbols are overloaded; for example, $+$ and $-$ denote the modules that rotate the turtle as well as the usual arithmetic operations. The length of the line and the magnitude of the rotation angle can be given globally or specified as parameters of individual modules. Branches are created using a stack mechanism: the opening square bracket pushes the current state of the turtle on the stack, and the closing bracket restores it to the last saved state. Other interpreted symbols will be introduced with the sample models.

4 Modeling curved limbs

The shape of curved limbs, such as stems and elongated leaves, is "vital in capturing the character of a species" [37]. In computer graphics, this was first recognized by Bloomenthal [5], who applied *generalized cylinders* to model tree branches. A generalized cylinder is obtained by sweeping a planar *generating curve*, which determines the organ's *cross section*, along a *carrier curve* [16] that defines the organ's *axis*. The generating curve may be closed, as is typically the case for stems, or open, as for thin leaves, and it may change size and shape while being swept [33]. It also must be properly oriented with respect to the carrier curve. The *Frenet frame* [34], which is frequently used for this purpose, creates well known problems along straight sections of the carrier curve and at inflection points, where it is not defined. It also twists 180° in the proximity of the inflection points [6]. To avoid these problems, we propose an alternative solution based on the use of turtle geometry. This solution subsumes the Frenet frame, as well as the twist-minimizing *parallel transport frame* [3, 6, 14], as special cases. The turtle frame was previously used by Jirasek *et al.* [15] in the context of biomechanical modeling of plant branches.

The carrier curve is defined as a sequence of infinitesimal turtle movements. Let s denote the arc-length distance of the turtle from the origin of this curve. To define a smooth curve, we specify functions $\omega_H(s)$, $\omega_L(s)$ and $\omega_U(s)$ that characterize the *rates* of turtle's rotations around the axes $\vec{H}\vec{L}\vec{U}$ as the turtle moves (we use the term "rate of rotation" although s is a spatial coordinate and not time). The infinitesimal rotations $d\Omega_H$, $d\Omega_L$ and $d\Omega_U$ between

curve points $\vec{P}(s)$ and $\vec{P}(s+ds)$ are then given by the equations:

$$d\Omega_H = \omega_H(s)ds, \quad d\Omega_L = \omega_L(s)ds, \quad d\Omega_U = \omega_U(s)ds. \qquad (4)$$

This specification yields a uniquely defined curve and moving reference frame (Appendix A.1). After replacing the infinitesimal increments ds by finite increments Δs, we obtain the following straightforward algorithm for modeling elongated plant organs:

Algorithm 1

```
1  #define ℓ            1.0   /* total axis length */
2  #define 𝒢            7     /* cross section ID */
3  #define Δs           0.02  /* turtle step */
4
5  #define ω_L(s)       func(1,s)
6  #define ω_U(s)       func(2,s)
7  #define ω_H(s)       func(3,s)
8  #define φ(s)         func(4,s)
9  #define width(s)     func(5,s)
10
11 Axiom: @#(𝒢) A(0,0)
12
13 A(s,φ): s ≤ ℓ
14       { ΔΩ_L = ω_L(s)Δs;
15         ΔΩ_H = ω_H(s)Δs;
16         ΔΩ_U = ω_U(s)Δs;
17         φ    = φ + φ(s)Δs; } ⇝
18              +(ΔΩ_L) &(ΔΩ_U) /(ΔΩ_H)
19              /(φ) #(width(s)) F(Δs) \(φ)
20              A(s + Δs,φ)
21
22 A(s,Φ): s > ℓ ⇝ ϵ
```

Following the implementation of generalized cylinders in the `cpfg` program [20], the generating curve is selected by expression $@\#(\mathcal{G})$ in the axiom (line 11). The generalized cylinder is created recursively by the first production (lines 13-20) as a sequence of slices of length Δs. The cross section size is defined by module # with the parameter $\mathrm{width}(s)$ (line 19), and is linearly interpolated between points s and $s+\Delta s$. The angles of turtle rotation are calculated according to Equation 4 in lines 14–16, and applied to the turtle in line 18. The order of rotations represented by the symbols $+$, $\&$ and $/$ in line 18 is arbitrary, since infinitesimal rotations commute. Function $\phi(s)$ (line 17) rotates the generating curve around the cylinder axis without affecting the shape of the axis. This is convenient when defining twisted organs. The second production (line 22) removes the apex A at the end of cylinder generation, by replacing it with the empty symbol ϵ. Figure 4 shows sample leaves and stems generated by this Algorithm, with all functions specified using the interactive function editor (Section 3).

From the user's perspective, functions ω_L, ω_U, ω_H, ϕ and `width`, control bending, twist, and tapering of a generalized cylinder. Our experience confirms Barr's observation that such deformations are intuitive operations for modeling three-dimensional objects [1]. On the other hand, the user may prefer to specify the shape of an axis directly, for example as a spline curve. If this is the case, we *frame* it (*i.e.*, compute turtle's rotations $d\Omega_U$, $d\Omega_L$ and $d\Omega_H$) as follows.

Let $\vec{P}(s), s \in [0,\ell]$, be a given smooth curve. Assume that it has been framed by a moving turtle; the turtle's heading vector $\vec{H}$ thus coincides with the tangent vector $\vec{T}$ to the curve for all $s \in [0,\ell]$. Denote by $\vec{H}\vec{L}\vec{U}$ the turtle orientation at point $\vec{P}(s)$ of this curve

Figure 4: Leaves and stems of a herb lily (left) and tulip (right), modeled using Algorithm 1. The models are based on drawings in [38, pp. 56 and 58].

and by $\vec{H}' = \vec{H} + d\vec{H}$ the direction of the heading vector at point $\vec{P}(s+ds)$. Following [12], the infinitesimal rotation $d\vec{\Omega}$ that changes vector $\vec{H}$ to $\vec{H}'$ satisfies the equation $d\vec{H} = d\vec{\Omega} \times \vec{H}$, hence:

$$d\vec{H} = d\vec{\Omega} \times \vec{H} = (\vec{U}d\Omega_U + \vec{L}d\Omega_L + \vec{H}d\Omega_H) \times \vec{H} \qquad (5)$$

$$= (\vec{U} \times \vec{H})d\Omega_U + (\vec{L} \times \vec{H})d\Omega_L + (\vec{H} \times \vec{H})d\Omega_H \qquad (6)$$

$$= \vec{L}d\Omega_U - \vec{U}d\Omega_L + 0d\Omega_H. \qquad (7)$$

By taking dot products of the first and last expression with vectors $\vec{L}$ and $\vec{U}$, we obtain:

$$d\vec{H} \cdot \vec{L} = (\vec{H}' - \vec{H}) \cdot \vec{L} = \vec{H}' \cdot \vec{L} = d\Omega_U, \qquad (8)$$

$$d\vec{H} \cdot \vec{U} = (\vec{H}' - \vec{H}) \cdot \vec{U} = \vec{H}' \cdot \vec{U} = -d\Omega_L. \qquad (9)$$

By substituting $\vec{T}'$ for $\vec{H}'$ to emphasize that $\vec{T}'$ is a given tangent vector to the curve being framed, we obtain finally:

$$d\Omega_U = \vec{T}' \cdot \vec{L} \quad \text{and} \quad d\Omega_L = -\vec{T}' \cdot \vec{U}. \qquad (10)$$

Equations 10 constrain two rotational degrees of freedom. The third angle $d\Omega_H$ remains unconstrained, because it is multiplied by 0 in Equation 7. This implies that a moving turtle frame can be assigned to a given curve in different ways. In particular, if we set $\omega_H(s)$ in such a way that vector $\vec{L}$ (or $\vec{U}$) always lies in the osculating plane, we obtain the Frenet frame, and if $\omega_H(s) \equiv 0$, we obtain the parallel transport frame. We commonly use the latter, because it minimizes rotations of the reference frame around the axis of the generalized cylinder. The resulting algorithm for approximating and framing a given curve $\vec{P}(s)$ using a sequence of turtle motions is given below.

Algorithm 2

```
1  #define 𝒫   1       /* curve ID */
2  #define K   57.29   /* radians to degrees */
3
4  Axiom: A(0) ?U(0,0,0) ?L(0,0,0)
5
6  A(s) > ?U(u_x,u_y,u_z) ?L(l_x,l_y,l_z) : { s' = s + Δs } s' ≤ ℓ
7        { t'_x = tanX(𝒫,s'); t'_y = tanY(𝒫,s'); t'_z = tanZ(𝒫,s');
8          ΔΩ_L = K * (t'_x l_x + t'_y l_y + t'_z l_z);
9          ΔΩ_U = −K * (t'_x u_x + t'_y u_y + t'_z u_z); } ⇝
10               +(ΔΩ_U) &(ΔΩ_L) F(Δs) A(s')
```

The initial structure consists of apex A followed by query modules $?U$ and $?L$ (line 4). The parameter of the apex represents the current position of the turtle, measured as its arc-length distance from

the origin of curve $\mathcal{P}$. The production (lines 6 to 10) creates an organ axis as a sequence of generalized cylinder slices of length Δs, as in Algorithm 1 (functions controlling the orientation and size of the generating curve have been omitted here for simplicity). Specifically, rotations $\Delta\Omega_U$ and $\Delta\Omega_L$ are calculated by multiplying (dot product) the vectors $\vec{U}$ and $\vec{L}$ (lines 8 and 9) returned by the query modules $?U$ and $?L$ (line 6) with the tangent vector to the curve $\mathcal{P}$ returned by the tanX, tanY and tanZ function calls (line 7). The values $\Delta\Omega_U$ and $\Delta\Omega_L$ orient the next segment of the curve, represented by module F(Δs) in line 10. House-keeping productions that erase modules A, $?U$ and $?L$ at the end of the derivation have been omitted from this listing.

Figure 5: *Allum vineale* (field garlic), modeled using Algorithm 2 after the photograph in [4].

A sample application of Algorithm 2 is shown in Figure 5. Stems of a dry garlic plant have been modeled interactively, then framed using Algorithm 2 to orient the generating curve. Although the generating curve is circular in this case, its orientation is important for proper polygonization of the resulting generalized cylinders.

The turtle frame also plays an important role in orienting the organs and branches that are attached to an axis. Before discussing this in detail, we will consider the spacing of organs along an axis.

5 Organ spacing

We call points at which organs are attached to an axis the *nodes*, and the axis segments delimited by them the *internodes*. Let $\{s_i\}, i = 0, 1, \ldots$, be a sequence of node positions on an axis, and $\{l_i = s_{i+1} - s_i\}$ be the associated sequence of internode lengths (Figure 6a). It is straightforward to define the internode lengths using a function λ of the position of one of its incident nodes, for instance using the formula $l_i = s_{i+1} - s_i = \lambda(s_i)$. Unfortunately, with this definition function λ does not provide a robust control over the node distribution, because a small change in the position of the initial node s_0 may result in a totally different sequence of the nodes that follow. For example, if $s_0 = 0$, the function λ shown in Figure 6b will yield the sequence of node positions $\{s_i\} = 0, 1, 2, 3, \ldots$ (internode length equal to 1), but if $s_0' = 0.25$, the sequence of node positions will be $\{s_i'\} = 0.25, 0.75, 1.25, 1.75, \ldots$ (internode length 0.5).

To achieve a more stable behavior, we observe that $1/\lambda(s)$ can be interpreted as the local *density* of nodes, in the sense that the integer part of the integral

$$N(s_o, s) = \int_{s_0}^{s} \frac{ds}{\lambda(s)} \qquad (11)$$

represents the number of internodes between node s_0 and point s on the axis. Thus, given the initial node s_0, positions of the subsequent nodes correspond to the integer increments of the value of function N, that is, $N(s_o, s_{i+1}) = N(s_o, s_i) + 1$ (Figure 6c). The sequence of nodes $\{s_i\}$ defined this way is no longer critically sensitive to the initial node position s_0. Specifically, in Appendix A.2 we prove

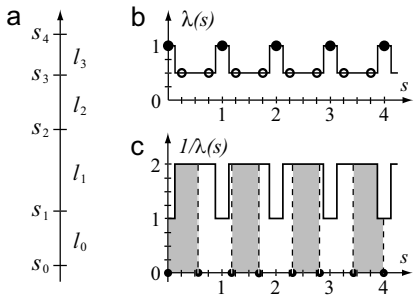


Figure 6: Partitioning an axis into segments. (a) The labeling of nodes and internodes. (b) Positional information represents the internode length. The same function $\lambda(s)$ generates very different node sequences (filled and empty circles), depending on the position of the initial node. (c) Positional information represents node density. Nodes are placed at the locations corresponding to the unit areas under the curve $1/\lambda(s)$. This definition leads to a more stable node spacing than (b).

that for any two node sequences $\{s_i\}$, $\{s_i'\}$ such that $s_0 < s_0' < s_1$, the elements of both sequences interleave: $s_i < s_i' < s_{i+1}$ for all $i = 0, 1, 2, \ldots$.

Specification of node spacing based on Equation 11 also has other useful properties. First, if $\lambda(s)$ has a constant value l between nodes s_i and s_{i+1}, then l is equal to the internode length:

$$\int_{s_i}^{s_{i+1}} \frac{ds}{l} = 1 \;\text{ implies }\; s_{i+1} - s_i = l. \qquad (12)$$

Second, if $\lambda(s)$ is a linear function, $\lambda(s) = as + b$, the length of consecutive internodes changes in a geometric sequence, $l_{i+1} = e^a l_i$ (proof in Appendix A.3). The ease of defining geometric sequences is important, because their approximations are often observed in nature (according to Niklas, they form the "null hypothesis" [22]).

The algorithm for placing nodes according to a given function $\lambda(s)$ is presented below.

Algorithm 3

```
1   Axiom: A(0,0)
2
3   A(s,a) : { s' = s + Δs }  s' ≤ ℓ
4         { a' = a + Δs/λ(s) ;
5           if (a' < 1) { flag = 0; }
6           else { a' = a' − 1; flag = 1; } } ⇝
7                 F(Δs) B(flag) A(s',a')
8
9   B(flag) : flag == 0 ⇝ ϵ
10  B(flag) : flag == 1 ⇝ @o
```

The initial structure consists of apex A (line 1). The first parameter represents the distance of the current point on the axis from the axis base, as in Algorithms 1 and 2. The second parameter represents the fractional part of the integral $N(0, s)$ given by Equation 11. The production in lines 3 to 7 creates the axis as a sequence of segments F of length Δs, separated by markers of potential node locations B. If the $flag$ is zero, module B is subsequently erased (line 9). When a exceeds 1, the $flag$ is set (line 6) to produce a node marked by symbols $@o$ (line 10).

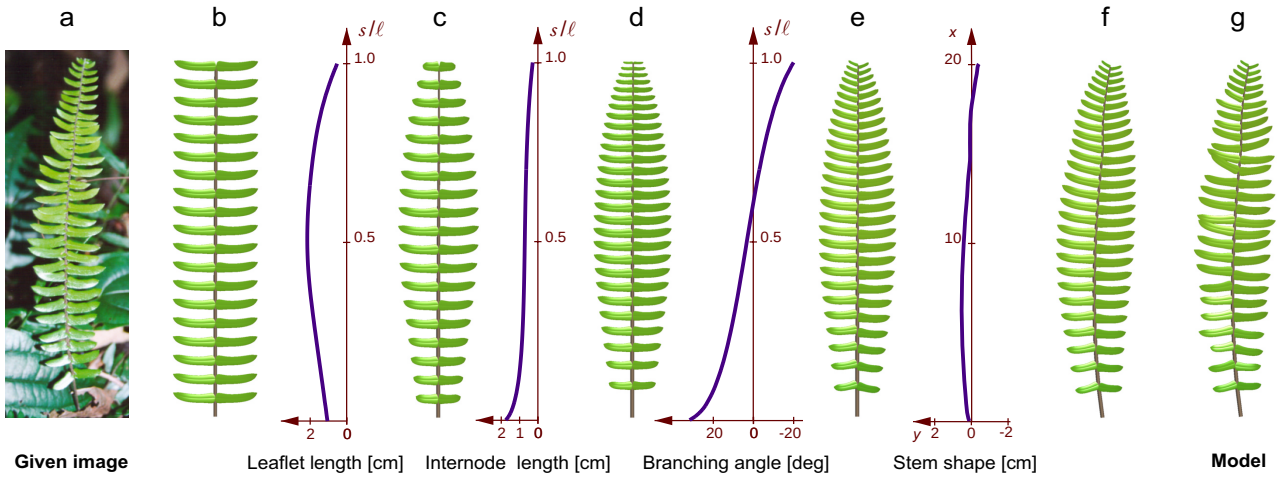


Figure 7: Using positional information to model a *Pellaea falcata* (sickle fern) leaf.

6 Modeling single-compound plant structures

We have combined the methods for framing and partitioning an axis into the following algorithm, which makes it possible to model a variety of *single-compound* structures (sequences of organs supported by a single stem). Definitions of graphical functions and constants used in previous algorithms have not been included. Secondary features, such as the randomization of values returned by functions, have also been omitted.

Algorithm 4

```
1   #define Φ   0          /* phyllotactic angle */
2
3   Axiom: A(0,0,0) ?U(0,0,0) ?L(0,0,0)
4
5   A(s,a,φ) > ?U(u_x,u_y,u_z) ?L(l_x,l_y,l_z) :
6              { s' = s + Δs } s' ≤ ℓ
7       { t'_x = tanX(P,s'); t'_y = tanY(P,s'); t'_z = tanZ(P,s');
8         ΔΩ_L = K * (t'_x l_x + t'_y l_y + t'_z l_z);
9         ΔΩ_U = −K * (t'_x u_x + t'_y u_y + t'_z u_z);
10          a   = a + Δs/λ(s)
11        if (a < 1) { flag = 0; }
12        else { a = a − 1; flag = 1; φ = φ + Φ; } } ⇝
13             +(ΔΩ_U) &(ΔΩ_L) #(stem_width(s))
14             F(Δs)B(s,φ,flag) A(s',a,φ)
15
16  B(s,φ,flag) : flag == 0 ⇝ ε
17  B(s,φ,flag) : flag == 1
18      { l = length(s); w = width(s); } ⇝
19          [ /(φ) [ +(brangle(s)) ~L(l,w) ]
20                 [ −(brangle(s)) ~L(l,w) ] ]
```

The key new element is the third production (lines 17 to 20), which inserts a pair of organs at the node. The organs are defined as instances of a predefined surface L, with the length, width and angle of insertion determined by functions of position s.

In order to present the operation of Algorithm 4 from a user's perspective, let us consider the process of modeling a *Pellaea falcata* (sickle fern) leaf. The photograph of the target structure is shown in Figure 7a. Construction begins with a generic single-compound (pinnate) leaf (b), which is generated when all graphically defined functions are set to their default constant values. The length of the leaflets is then modified as a function of their position on the stem (c). Since the leaf silhouette is determined by the extent of its component leaflets, this function controls the overall leaf shape. The next two functions define the lengths of the internodes (d) and the values of the branching angles between the stem and the leaflets (e). The stem shape is then established by manipulating a parametric curve (f). Finally, the branching angles and the leaflet lengths are randomized to capture the unorganized variation present in the original leaf (g). The model also makes use of functions that have not been shown in Figure 7, which define the taper of the stem and the width of the leaflets.

In the above example, the individual leaflets have been modeled as predefined surfaces L, scaled in length and width using functions of their position on the stem (lines 18 to 20 in Algorithm 4). Leaves, petals and similar organs can also be modeled as generalized cylinders with Algorithm 1. We use this technique in most

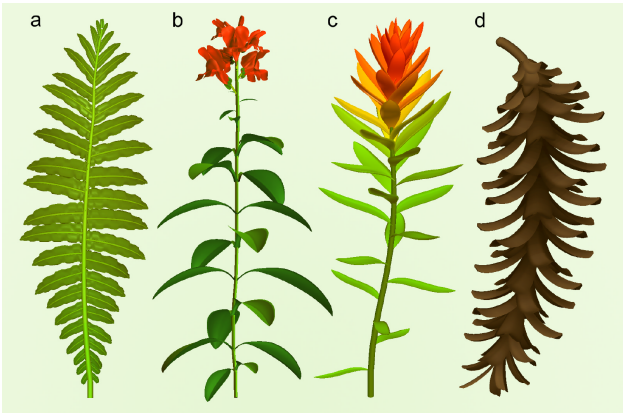


Figure 8: Plants and plant organs with different phyllotactic patterns: (a) *Blechnum gibbum* leaf with the distichous arrangement of leaflets, (b) *Antirrhinum majus* (snapdragon) plant with a decussate arrangement of leaves, (c,d) *Casilleja coccinea* (Indian paintbrush) plant and *Pinus strobus* (white pine) cone with spiral arrangements of leaves and scales.

models, because it allows us to define and manipulate organ shapes more easily. For example, the rippled surface of the *Blechnum gibbum* leaflets (Figure 8a) was obtained by randomly changing the shape, size and orientation of the generating curve.

Constant Φ in Algorithm 4 controls *phyllotaxis*, or the arrangement of organs around the stem [28]. If $\Phi = 0$, organs are arranged in a planar *distichous* pattern, as in Figures 7 and 8a. If $\Phi = 90^\circ$, consecutive pairs of organs are issued in mutually perpendicular planes, forming a *decussate* pattern (Figure 8b). Finally, if $\Phi = 137.5^\circ$ (the golden angle), and only one organ is attached to each node (line 20 of Algorithm 4 is removed), a *spiral* phyllotactic pattern results (Figure 8c and d). Thus, a change in a single constant extends Algorithm 4 to three dimensions.

Figure 9: *Helichrysum bracteatum* (strawflower).

A distinctive feature of *Helichrysum bracteatum* (a strawflower, Figure 9) is the posture of petals (ray florets), which are more curved near the center of the flower head than on the outside. To capture this gradient, the position of the petals on the main axis of the flower head was used to interpolate between two curves that describe the extreme postures of the petals. A similar technique made it possible to control the shape of leaves and petals in the beargrass model (Figure 10). Photographs of the inflorescences that we used as a reference to construct this model are shown in Figure 11.

7 Compact phyllotactic patterns

In spiral phyllotactic patterns, the individual organs, *e.g.* petals, florets, or scales, are often densely packed on their supporting surface (the *receptacle*), as illustrated by the model of beargrass. Modeling such patterns using Algorithm 4 requires a coordinated manipulation of the radius of the receptacle, the size of the organs being placed, and their vertical displacement (corresponding to the internode length). In this section we facilitate the modeling process by relating the vertical displacement to the radius of the supporting surface and the size of organs. Both the radius and the organ size can be defined as functions of organ position on the receptacle, making it possible to capture a wide range of forms and patterns. The proposed model has the same generative power as the *collision-based* model of phyllotaxis introduced by Fowler *et al.* [11], but operates faster because it avoids the explicit detection of collisions between organs.

Vogel [35] provided the first mathematical description of phyllotactic patterns used for computer graphics purposes [28]. His model places equally sized organs on the surface of a flat disk, stating that the n-th organ will have polar coordinates:

$$\phi = n \cdot 137.5^\circ, \quad r = c\sqrt{n}, n = 1, 2, \ldots \tag{13}$$

where c is a constant. The angular displacement of 137.5° between consecutive organs is treated as empirical data, reproduced but not explained by the model. The formula for the radial displacement r is justified by two observations: (a) since organs are placed from the disk center outwards, the ordering number n of the organ placed at a distance r from the center is equal to the total number of organs

Figure 10: Model of *Xerophyllum tenax* (beargrass).

Figure 11: Photographs of *Xerophyllum tenax* inflorescences.

that occupy a disk of radius r, and (b) if all organs occupy the same area, the total number n of organs in a disk of radius r will be proportional to r^2, hence $r = c\sqrt{n}$.

Vogel's model abstracts from the shape of organs and places them in a disk according to the area they occupy. Lintermann and Deussen proposed a similar approximation to derive a formula for placing organs on the surface of a sphere [19]. Both approaches are subsumed by the model of Ridley [31], which operates on arbitrary surfaces of revolution. Our algorithm is based on Ridley's analysis.

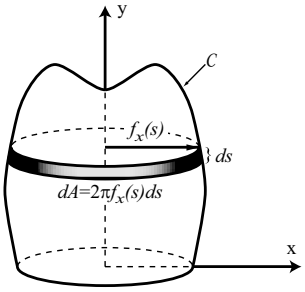


Figure 12: A receptacle.

Let $(f_x(s), f_y(s)), \quad s \in [0, L]$ be a parametric definition of a planar curve $\mathcal{C}$ that generates the receptacle when rotated around the y axis of the coordinate system (Figure 12). We assume natural parameterization of the curve $\mathcal{C}$, which means that parameter s is the arc-length distance of point $(f_x(s), f_y(s))$ from the origin of this curve. The area dA of the infinitesimal slice of the receptacle generated by the arc $[s, s + ds]$ is then equal to $2\pi f_x(s)ds$ (Figure 12). We denote by $\pi\rho^2(s)$ the area occupied by an organ placed on the receptacle at a distance s from the origin of the generating curve $\mathcal{C}$. As in the case of partitioning an axis into internodes (Section 5), we can interpret $1/\pi\rho^2(s)$ as the organ density at s. The integer part of the integral

$$N(0, s) = \int_0^s \frac{2\pi f_x(s)}{\pi\rho^2(s)} ds = \int_0^s \frac{2 f_x(s)}{\rho^2(s)} ds \qquad (14)$$

is then equal to the total number of organs placed in the portion $[0, s]$ of the receptacle. Consecutive organs are placed at locations that increment $N(0, s)$ by one. This leads to the following algorithm:

Algorithm 5

```
1   #define C      1              /* generating curve ID */
2   #define ℓ      curveLen(C)    /* length of curve C */
3   #define ρ(s)   func(2,s)      /* density function */
4   #define Δs     0.001          /* integration step */
5
6   Axiom: A(0,0)
7
8   A(s,a) : s < ℓ
9       { while( a < 1 && s < ℓ)
10        { x = curveX(C, s);
11          a = a + (2x/ρ²(s))Δs;
12          s = s + Δs;
13        }
14      a = a − 1; y = curveY(C, s);
15    }
16    ⇝ [f(y)-(90)f(x)~O(ρ(s))] \(137.5) A(s,a)
```

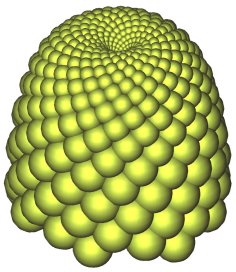


Figure 13: Example of a compact phyllotactic pattern generated using Algorithm 5.

The first parameter of module A represents the arc-length distance s of the current point from the base of the receptacle. The second parameter is the fractional part a of the integral $N(0, s)$ (Equation 14). The integration is performed incrementally by the *while* loop inside the production (lines 9 to 13). When the integral reaches 1, an organ O of radius $\rho(s)$ is placed at height y and distance x from the receptacle axis y (line 16). Consecutive organs are rotated with respect to each other by the golden angle 137.5° measured around this axis. A sample pattern generated by Algorithm 5 is shown in Figure 13.

Figure 14: Inflorescences of *Kniphofia* sp. (red-hot poker plant) generated using Algorithm 5: models of two developmental stages (top) and the photographs used as a reference (bottom).

Figure 15: A *Pinus banksiana* (Jack pine) cone.

In realistic models, we replace spheres O by models of plant organs, as in [11]. For example, Figure 14 shows two developmental stages of the inflorescence of *Kniphofia* sp. (red-hot poker plant), in which florets have been modeled using generalized cylinders. In Figure 15 the algorithm has been additionally modified to allow for a curved cone axis. This modification is equivalent to the deformation of a straight cone, performed as a post-processing step.

8 Modeling multiple-compound structures

Algorithms 4 and 5, introduced in the previous sections, have been illustrated using examples of single-compound monopodial structures, each consisting of a sequence of organs placed along an axis or on a receptacle. The same algorithms can also be used, how-

Figure 16: A photograph and a model of a *Spiraea* sp. twig. The arrangement of shoots on the twig and the arrangement of leaves and flowers in each shoot follow the spiral phyllotactic pattern. The approximately vertical posture of all shoots reflects strong orthotropism, which has been simulated by biasing the turtle's heading vector in the vertical direction as described in [28, page 58].

ever, to generate structures in which the main axis supports entire substructures. For example, the *Spiraea* sp. twig shown in Figure 16 was constructed using Algorithm 4 twice: first to place the flower-bearing shoots along the main stem, then to place the leaves and the flowers within each shoot. In this case, all shoots have been assumed equal, except for the different shoot axis shapes caused by their *orthotropism* (tendency to grow vertically). In general, however, the supported structures may vary in a systematic manner, reflecting a morphogenetic gradient along the main stem.

To capture this gradient, we assume that, given two branches of the same order, the shorter branch is identical (up to the effects of tropisms and random variation) to the *top* portion of the longer branch. This concept of *branch mapping* is supported by both biological arguments and simulation results.

Biologically, it is related to the fact that apical meristems, the main engines of plant development, are located at the distal ends of

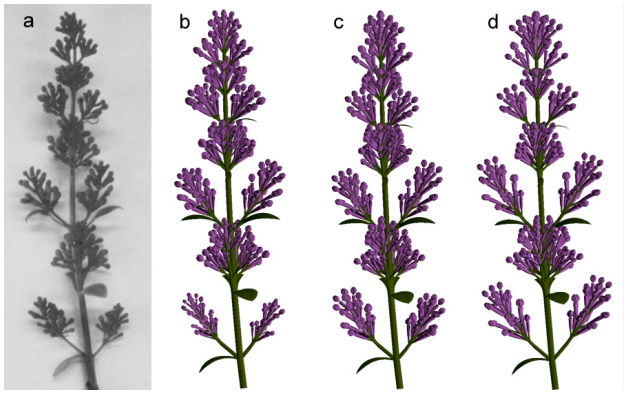


Figure 17: The effect of branch mapping. (a) An inflorescence of common lilac *Syringa vulgaris*. (b) Reconstruction of this inflorescence based on the measurements of all branches and flowers. (c) The same structure, all flowers assumed to be identical. (d) An approximate reconstruction based on branch mapping.

branches. Thus, if branch $\mathcal{B}$ develops over a shorter time or at a slower rate than an otherwise equivalent branch $\mathcal{A}$, branch $\mathcal{B}$ will resemble the *top* portion of $\mathcal{A}$.

A modeling example supporting the use of branch mapping is shown in Figure 17. An inflorescence of common lilac *Syringa vulgaris* (a) has been measured and reconstructed at three levels of accuracy: with all architectural information present (b), using the assumption that all flowers are identical (c), and using the assumption that shorter branches are identical to the top portions of the longer branches of the same order (d). Although reconstruction (d) is visually the least accurate, it still matches the real structure well.

Branch mapping makes it possible to define all branches of the same order using one set of functions. This concept is captured by the following algorithm.

Algorithm 6

```
1   Axiom: A(0,0)
2
3   A(o,s) : o < MAX && s < max_len[o]
4            { rel = s/max_len[o]; } ~>
5        #(int_width(o,rel)) F(int_len(o,rel))
6        [+(branch_ang(o,rel))
7             A(o + 1,max_len[o + 1] - branch_len(o,rel)) ]
8        [-(branch_ang(o,rel))
9             A(o + 1,max_len[o + 1] - branch_len(o,rel)) ]
10       /(90) A(o,s+int_len(o, rel))
11
12  A(o,s) : s >= max_len[o] ~> ~K
```

Algorithm 6 can be viewed as a recursive version of Algorithm 4, with the mechanism for creating curved axes removed for simplicity, and the internode length determined using point-sampled positional information as in Figure 6b for the same reason. Parameters o and s of the apices A represent the axis order and position along this axis, respectively. The array max_len$[o]$ specifies the length ℓ_{max} of the longest axis of each order $o <$ MAX. This value is used to represent positional information in relative terms, as a fraction rel of ℓ_{max} (line 4). This facilitates the specification of all functions, since they have fixed domain $[0, 1]$. Functions int_width(o, rel), int_len(o, rel), branch_ang(o, rel) and branch_len(o, rel) characterize morphogenetic gradients: the width and length of internodes, the branching angles at which the child branches are inserted, and the length of these child branches. All axes of the same order share the same set of functions. Within an axis of length ℓ, parameter s ranges from the initial value of $\ell_{max} - \ell$ (assigned to the newly created apices A in lines 7 and 9) to the maximum value of ℓ_{max} (condition in line 3). Thus, morphogenetic gradients along shorter axes are aligned with the distal portion of the longest axis of the same order, as required for branch mapping. Predefined flowers K are placed at the ends of the branches (line 12).

Examples of lilac inflorescences generated by Algorithm 6 are shown in Figure 18. Lilac inflorescences have decussate phyllotaxis. As was the case for Algorithm 4, a small modification of Algorithm 6 makes it possible to generate structures with spiral phyllotaxis. An example of the resulting structure — the inflorescence of an *Astilbe plant* — is shown in Figure 19.

Algorithm 6 can also be applied to approximate trees with clearly delineated branch axes (many young trees satisfy this criterion). If the axes of first-order branches are approximately straight and higher-order branches are relatively short, the outline of the tree crown is determined by the extent of the first-order branches and

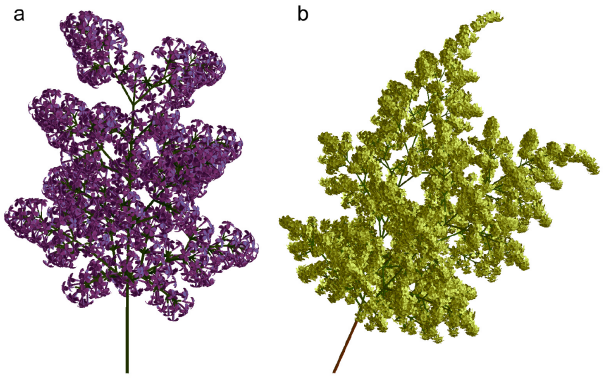


Figure 18: Inflorescences of two lilac species modeled using Algorithm 6: (a) *Syringa chinensis* CV. Rubra and (b) *Syringa reticulata*.

Figure 19: A photograph and a model of an *Astilbe x arendsii* CV. Diamant plant.

can easily be controlled by function branch_len$(0, rel)$ (Figure 20). In this sense, the use of positional information addresses the problem of progressing from silhouette to detail in the modeling process, exemplified by Figure 1c.

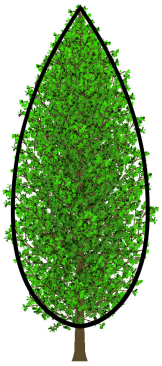


Figure 20: A generic tree model and its silhouette specifciation.

The problem of generating trees given their silhouettes occurs in several applications. One of them is the modeling and rendering of plant ecosystems. According to the approach proposed by Deussen *et al.* [10], the complexity of ecosystem modeling can be addressed by performing an individual-based simulation of the whole ecosystem, then replacing the coarse plant models used in this simulation with their detailed counterparts. The modeling method described in the present paper provides a means of creating plant models that match silhouettes determined at the ecosystem level (Figure 21).

9 Conclusions

We have explored the idea of plant modeling with functions that relate features of a plant to their positions along plant axes. Our experience confirms previous observations that this use of positional information is intuitive and well suited to the interactive modeling of plants. Visually important aspects of plant appearance — posture, the arrangement of components, and the overall silhouette — can easily be captured and controlled, while the procedural approach removes the tedium of specifying and placing each plant component individually. The algorithms are sufficiently fast to support interactive plant modeling on current personal computers.

We demonstrated the power of the modeling with positional information by recreating the form of several plants found in nature, presented on photographs, or depicted in drawings. The modeled structures range from individual leaves to compound herbaceous plants and trees.

The use of positional information is not limited to interactive modeling applications. We showed this by incorporating detailed tree models into a plant ecosystem model that only provided coarse characteristics of tree silhouettes. A related potential application is the automatic generation of plant models that match silhouettes of real trees, given their photographs [32].

At the technical level, our paper contributes: (a) a conceptual distinction between L-systems and Chomsky grammars as formal bases of developmental and structural plant models; (b) a generalized method for framing plant axes, free of the artifacts of the Frenet frame; (c) a robust method for spacing organs along plant axes; (d) an analytic method for generating phyllotactic patterns on arbitrary surfaces of revolution, based on Ridley's model; (e) the notion of branch mapping and its application to the modeling of compound plant structures; and (f) an example of the modeling system that integrates all of these concepts.

One open research problem is the use of constraints. In Algorithm 5 we introduced a relation between organ size and available space to constrain organ position in phyllotactic patterns. Many other relations have also been identified by biologists and can be applied to plant modeling [17]. By incorporating them into the algorithms we may further facilitate the modeling process. Specifically, constraints may reduce the number of parameters and functions that must be specified explicitly, while enforcing biological plausibility of the resulting structures.

Another interesting problem falls in the domain of interactive modeling techniques. In the present implementation, the user manipulates function plots, curves, and surfaces that are displayed separately from the model. A direct manipulation interface, in which the user would interact with the modeled structure itself, may lead to an even more intuitive modeling process.

A Appendices

A.1 Fundamental theorem of differential turtle geometry

The method for modeling curved limbs presented in Section 4 is based on the following extension of the fundamental theorem of differential geometry for three-dimensional curves [34, page 61] to the turtle reference frame.

Theorem. Let $\vec{H}(s)\vec{L}(s)\vec{U}(s)$ denote a moving reference frame defined on an interval $[0, \ell]$. Furthermore, let $\vec{H}(0)\vec{L}(0)\vec{U}(0)$ be the

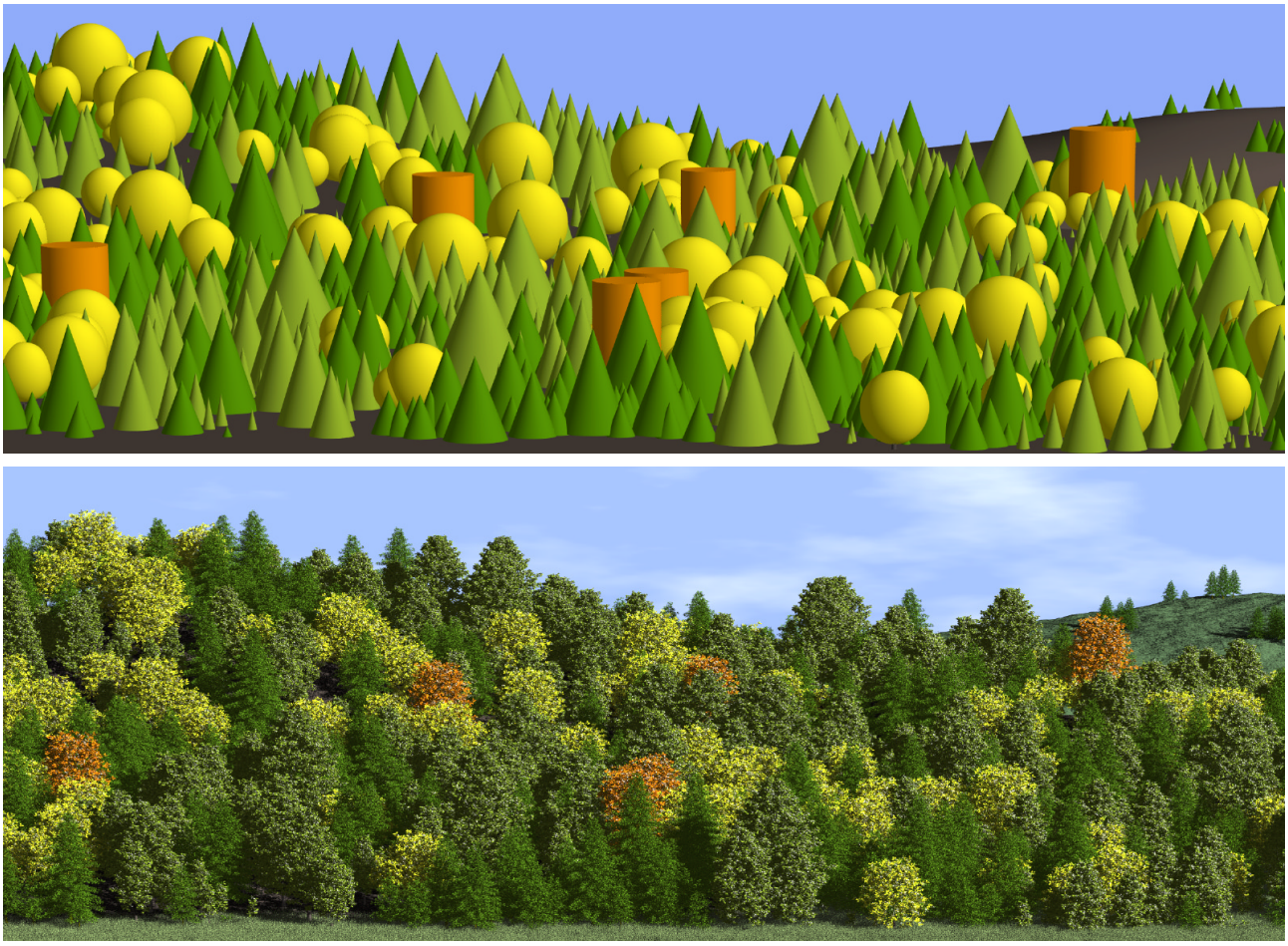


Figure 21: Visualization of an ecosystem simulation. Top: direct visualization. Bottom: realistic visualization. Tree silhouettes match the shapes coarsely defined at the ecosystem level.

initial orientation of this frame, and differentiable functions $\omega_H(s)$, $\omega_L(s)$ and $\omega_U(s)$ be its rates of rotation around the axes $\vec{H}(s)$, $\vec{L}(s)$ and $\vec{U}(s)$. The orientation of this frame is then uniquely defined for all $s \in [0, \ell]$. Moreover, given the initial frame position $\vec{P}(0)$, there is a unique differentiable curve $\vec{P}(s)$ for which s is the natural (arc-length) parameter, such that $\vec{H}(s)$ is tangent to $\vec{P}(s)$ for all $s \in [0, \ell]$.

Proof. Following [12], an infinitesimal rotation vector $d\vec{\Omega}$ acting on an arbitrary vector $\vec{A}$ changes it by $d\vec{A} = d\vec{\Omega} \times \vec{A}$. Thus, changes of the $\vec{H}\vec{L}\vec{U}$ reference frame due to the rotation rate vector $\vec{\omega} = \omega_H\vec{H} + \omega_L\vec{L} + \omega_U\vec{U}$ satisfy the system of equations:

$$\frac{d\vec{H}}{ds} = \vec{\omega} \times \vec{H}, \quad \frac{d\vec{L}}{ds} = \vec{\omega} \times \vec{L}, \quad \frac{d\vec{U}}{ds} = \vec{\omega} \times \vec{U}. \qquad (15)$$

Given the initial frame orientation $\vec{H}(0)\vec{L}(0)\vec{U}(0)$, vectors $\vec{H}(s)$, $\vec{L}(s)$ and $\vec{U}(s)$ are thus the unique solution to the initial value problem for the system of differential equations (15) in the interval $[0, \ell]$. Moreover, curve $\vec{P}(s)$ is given by the integral:

$$\vec{P}(s) = \vec{P}(0) + \int_0^s \vec{H}(s)ds. \quad \square \qquad (16)$$

A.2 Stability of node distribution

The fact that the distribution of nodes defined by integer values of Equation 11 does not depend critically on the choice of the initial node can be formally stated as follows.

Theorem. Consider a function λ such that $\lambda(s) > 0$ for all $s > 0$, and let $s_o, s'_0 > 0$ be two numbers. Using function N specified by Equation 11, define sequences $\{s_i\}$ and $\{s'_i\}$ such that $s_{i+1} = N(s_0, s_i) + 1$ and $s'_{i+1} = N(s'_0, s'_i) + 1$ for all $i = 0, 1, 2, \ldots$. If $s_0 < s'_0 < s_1$ then $s_i < s'_i < s_{i+1}$ for all $i = 0, 1, 2, \ldots$.

Proof by induction on i. The assumption $\lambda(s) > 0$ implies that $F(s) \equiv N(s_0, s)$ is an increasing function of the argument s. Thus, $s_i < s'_i < s_{i+1}$ implies $F(s_i) < F(s'_i) < F(s_{i+1})$, and therefore $F(s_i)+1 < F(s'_i)+1 < F(s_{i+1})+1$. By substituting $F(s_i)+1 = F(s_{i+1})$, $F(s'_i) + 1 = F(s'_{i+1})$, and $F(s_{i+1}) + 1 = F(s_{i+2})$, we obtain $F(s_{i+1}) < F(s'_{i+1}) < F(s_{i+2})$, hence $s_{i+1} < s'_{i+1} < s_{i+2}$. $\square$

A.3 Distribution of nodes defined by a linear function λ.

Theorem. Consider the sequence of nodes s_i defined by integer values of Equation 11, and let $\lambda(s) = as + b$. The length of consecutive internodes $l_i = s_{i+1} - s_i$ satisfies the equation $l_{i+1} = e^a l_i$ for $i = 0, 1, 2, \ldots$.

Proof. From Equation 11 we obtain:

$$1 = N(s_0, s_{i+1}) - N(s_0, s_i) \quad (17)$$

$$= \int_{s_i}^{s_{i+1}} \frac{ds}{as+b} = \frac{1}{a} \ln \frac{as_{i+1}+b}{as_i+b} \; . \quad (18)$$

Thus, $as_{i+1} + b = e^a(as_i + b)$ and, similarly, $as_{i+2} + b = e^a(as_{i+1}+b)$. By subtracting these equations sidewise and dividing by a we obtain $s_{i+2} - s_{i+1} = e^a(s_{i+1} - s_i)$, or $l_{i+1} = e^a l_i$. □.

Acknowledgments

We would like to thank: Lynn Mercer for contributing Figures 1b and c, Josh Barron for Figure 8a, Laura Marik for Figure 15, Enrico Coen for joint work on the snapdragon model (Figure 8b), Campbell Davidson for joint work on the lilac models (Figure 18), Christophe Godin for joint work on the decomposition rules, Bernd Lintermann and Oliver Deussen for a detailed demo of `xfrog`, and the referees for their insightful comments. The support of the Natural Sciences and Engineering Research Council of Canada, the International Council for Canadian Studies, the Alberta MACI project and the University of Calgary is gratefully acknowledged.

References

[1] A. H. Barr. Global and Local Deformations of Solid Primitives. Proceedings of SIGGRAPH 84, in *Computer Graphics*, 18, 3, July 1984, pages 21–30.

[2] D. Barthélémy, Y. Caraglio, and E. Costes. Architecture, Gradients Morphogénétiques et Age Physiologique ches les Végétaux. In J. Bouchon, Ph. De Reffye, and D. Barthélémy, editors, *Modélisation et Simulation de l'Architecture des Végétaux*, pages 89–136. INRA Editions, Paris, 1997.

[3] R. L. Bishop. There Is More Than One Way to Frame a Curve. *Amer. Math. Monthly*, 82(3):246–251, March 1975.

[4] H. Bjornson. *Weeds*. Chronicle Books, San Francisco, 2000.

[5] J. Bloomenthal. Modeling the Mighty Maple. Proceedings of SIGGRAPH 85, in *Computer Graphics*, 19, 3, July 1985, pages 305–311.

[6] J. Bloomenthal. Calculation of Reference Frames Along a Space Curve. In A. Glassner, editor, *Graphics Gems*, pages 567–571. Academic Press, Boston, 1990.

[7] T. E. Burk, N. D. Nelson, and J. G. Isebrands. Crown Architecture of Short-rotation, Intensively Cultured *Populus*. III. A Model of First-order Branch Architecture. *Canadian Journal of Forestry Research*, 13:1107–1116, 1983.

[8] N. Chomsky. Three Models for the Description of Language. *IRE Trans. on Information Theory*, 2(3):113–124, 1956.

[9] P. de Reffye, C. Edelin, J. Françon, M. Jaeger, and C. Puech. Plant Models Faithful to Botanical Structure and Development. Proceedings of SIGGRAPH 88, in *Computer Graphics* 22, 4, August 1988, pages 151–158.

[10] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. Proceedings of SIGGRAPH 98, Annual Conference Series, July, 1998, pages 275–286.

[11] D. R. Fowler, P. Prusinkiewicz, and J. Battjes. A Collision-based Model of Spiral Phyllotaxis. Proceedings of SIGGRAPH 92, in *Computer Graphics*, 26, 2, July 1992, pages 361–368.

[12] H. Goldstein. *Classical Mechanics*. Addison-Wesley, Reading, 1980.

[13] J. S. Hanan. *Parametric L-systems and Their Application to the Modelling and Visualization of Plants*. PhD thesis, University of Regina, June 1992.

[14] A. J. Hanson. Quaternion Gauss Maps and Optimal Framings of Curves and Surfaces. Technical Report 518, Computer Science Department, Indiana University, Bloomington, IN, 1998.

[15] C. Jirasek, P. Prusinkiewicz, and B. Moulia. Integrating Biomechanics into Developmental Plant Models Expressed Using L-systems. In H.-Ch. Spatz and T. Speck, editors, *Plant Biomechanics 2000*, pages 615–624. Georg Thieme Verlag, Stuttgart, 2000.

[16] J. J. Koenderink. *Solid Shape*. MIT Press, Cambridge, 1993.

[17] P. Kruszewski and S. Whitesides. A General Random Combinatorial Model of Botanical Trees. *Journal of Theoretical Biology*, 191(2):221–236, 1998.

[18] B. Lintermann and O. Deussen. XFROG 2.0. www.greenworks.de, December 1998.

[19] B. Lintermann and O. Deussen. Interactive Modeling of Plants. *IEEE Computer Graphics and Applications*, 19(1):56–65, 1999.

[20] R. Měch. *Modeling and Simulation of the Interactions of Plants with the Environment using L-systems and their Extensions*. PhD thesis, University of Calgary, October 1997.

[21] R. Měch and P. Prusinkiewicz. Visual Models of Plants Interacting with their Environment. Proceedings of SIGGRAPH 96, Annual Conference Series, August, 1996, pages 397–410.

[22] K. J. Niklas. *Plant Allometry: The Scaling of Form and Process*. The University of Chicago Press, Chicago, 1994.

[23] P. Oppenheimer. Real Time Design and Animation of Fractal Plants and Trees. Proceedings of SIGGRAPH 86, in *Computer Graphics*, 20, 4, August 1986, pages 151–158.

[24] W. F. Powell. *Drawing Trees*. Walter Foster Publishing, Inc., Laguna Hills, CA, 1998.

[25] P. Prusinkiewicz, J. Hanan, and R. Měch. An L-system-based Plant Modeling Language. Lecture Notes in Computer Science 1779, pages 395–410. Springer-Verlag, Berlin, 2000.

[26] P. Prusinkiewicz, M. James, and R. Měch. Synthetic Topiary. Proceedings of SIGGRAPH 94, Annual Conference Series, July, 1994, pages 351–358.

[27] P. Prusinkiewicz, R. Karwowski, R. Měch, and J. Hanan. L-studio/cpfg: A Software System for Modeling Plants, 2000. Lecture Notes in Computer Science 1779, pages 457–464. Springer-Verlag, Berlin, 2000.

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

[29] W. T. Reeves and R. Blau. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. Proceedings of SIGGRAPH 85, in *Computer Graphics*, 19, 3, July 1985, pages 313–322.

[30] 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.

[31] J. N. Ridley. Ideal Phyllotaxis on General Surfaces of Revolution. *Mathematical Biosciences*, 79:1–24, 1986.

[32] T. Sakaguchi. Botanical Tree Structure Modeling Based on Real Image Set. SIGGRAPH 98 Conference Abstracts and Applications, 1998.

[33] J. M. Snyder and J. T. Kajiya. Generative Modeling: A Symbolic System for Geometric Modeling. Proceedings of SIGGRAPH 92, in *Computer Graphics*, 26, 2, July 1992, pages 369–378.

[34] I. Vaisman. *A First Course in Differential Geometry*. Marcel Dekker, New York, 1984.

[35] H. Vogel. A Better Way to Construct the Sunflower Head. *Mathematical Biosciences*, 44:179–189, 1979.

[36] J. Weber and J. Penn. Creation and Rendering of Realistic Trees. Proceedings of SIGGRAPH 95, Annual Conference Series, August, 1995, pages 119–128.

[37] K. West. *How to Draw Plants. The Techniques of Botanical Illustration*. Timber Press, Portland, OR, 1997.

[38] E. Wunderlich. *Botanical Illustration in Watercolor*. Watson–Guptill, New York, 1991.

L-systems and partial differential equations*

Mark Hammel and Przemyslaw Prusinkiewicz
Department of Computer Science
University of Calgary

1 Introduction

Interesting applications of parametric context-sensitive L-systems stem from their capability of expressing numerical solutions to initial value problems for partial differential equations. This capability was originally explored in the context of simulations performed using CELIA, the first software implementation of L-systems [2, 3, 9, 12], with the most general observations made in [10]. In this note, we present an approach to solving the initial value problem for PDEs with L-systems, using a parabolic (diffusion) equation as an example. We then apply this approach to solve a system of reaction-diffusion equations operating in a one-dimensional medium of constant size, as well as in an expanding medium. These solutions represent the evolution of the spatial distribution of the dependent variable(s) over time, and therefore lend themselves in a natural way to visualizations using extruded objects in space-time. In the examples considered, the visualizations lead to a realistic image of the shell of *Nautilus pompilius* with a pigmentation pattern, and to a graphical representation of the development of a filamentous bacteria *Anabaena catenula*.

2 Diffusion and decay

Let us consider the following equation:

$$\frac{\partial u}{\partial t} = -\nu u + D\frac{\partial^2 u}{\partial x^2}. \tag{1}$$

*Adapted from: M. Hammel and P. Prusinkiewicz: Visualization of developmental processes by extrusion in space-time, Proceedings of Graphics Interface '96, pp. 246–258.

If u is interpreted as the concentration of a substance C, this equation represents the decay of C with time constant ν and the diffusion of C along axis x with the diffusion coefficient D (for example, see [6]). Suppose that we want to solve this equation in the interval $[a, b]$ for $t \geq 0$, assuming the boundary conditions $u(a, t) = u_a, u(b, t) = u_b$, and the initial conditions

$$u(x, 0) = u_a + (u_b - u_a)\frac{x - a}{b - a}. \tag{2}$$

Following the finite-difference method [19, Chapter 19], we approximate the derivatives in Equation (1) using values taken at equally spaced sampling points along both the x and t axes:

$$\begin{array}{rcl} x_i & = & x_0 + i\Delta x, \quad \text{where } i = 0, 1, \ldots, m, \\ t_j & = & t_0 + j\Delta t, \quad \text{where } j = 0, 1, 2, \ldots. \end{array} \tag{3}$$

Using notation $u_i^j = u(x_i, t_j)$, we obtain:

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} = -\nu u_i^j + D\frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{(\Delta x)^2}, \tag{4}$$

which leads to

$$u_i^{j+1} = u_i^j + \left(-\nu u_i^j + D\frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{(\Delta x)^2}\right)\Delta t. \tag{5}$$

For any values of indices i and j, Equation (5) can be regarded as assigning a new value u_i^{j+1} to the variable u_i^j, taking into account the values u_{i+1}^j and u_{i-1}^j at the neighboring sampling points. Any sampling point along the axis x (except for the boundary points) is subject to a similar assignment, thus Equation (5) can be rewritten as the following context-sensitive L-system production:

$$M(u_l) < M(u) > M(u_r) \rightarrow M\left(u + (-\nu u + D\tfrac{u_l - 2u + u_r}{(\Delta x)^2})\Delta t\right). \tag{6}$$

Notice that the L-system notation eliminates the need for index arithmetic. The subscripts in the formal parameter names u_l, u, and u_r are not numbers, but mnemonic descriptors of the left and right neighbors. Similarly, indices are not needed to distinguish between the "old" and "new" values of variable u at any point in space, because the progress of time is implicit in the notion of a derivation step in an L-system.

To provide a framework for finite differencing expressed by production (6) a complete L-system solving Equation (1) must also:

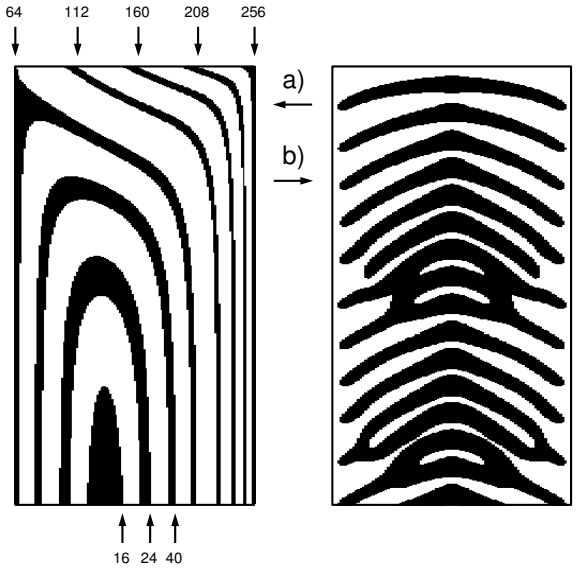


Figure 1: a) Visual representation of a solution to the PDE (1) obtained using an L-system based on production (6). Boundaries separating black and white regions indicate selected values of variable u. b) Visual representation of a solution to the PDE (7) obtained using an L-system based on production (10). White areas represent concentrations $a < 0.15$, and black areas represent concentrations $a \geq 0.15$. In both figures, time progresses from the top down.

- create a string of m modules M from the axiom,
- set the initial value of variable u in each module,
- maintain the boundary values of u in the first and the last modules M during the derivation process.

In addition, a graphical output must be associated with each module M if a visual representation of the solution is needed.

Figure 1a shows an extruded representation of the solution to PDE (1) obtained using an L-system in which each module M is shown as a line segment of unit length, with the color dependent on the value of variable u. The values of constants used in this simulation were: $\nu = 0.01, D = 5, a = 0, u_a = 64, b = 128, u_b = 256, \Delta t = 1$, and $m = 128$, yielding $\Delta x = \frac{b-a}{m} = 1$.

3 Reaction-diffusion

The described approach to solving partial differential equations using L-systems can easily be extended to systems of equations. In this case, a module M will have several parameters, each representing a different dependent variable. We will illustrate this technique by referring to reaction-diffusion models of the formation of pigmentation patterns in sea shells [7, 14, 15, 16, 17][1]. The models recreate pattern formation in nature, which is characterized by Meinhardt as follows [15, p. vii]:

> A mollusc can enlarge its shell only at the shell margin. In most cases, only at this margin are new elements of the pigmentation pattern added. Therefore, the shell pattern preserves a record in time of a process that took place in a narrow zone at the growing edge. A certain point on the shell represents a certain moment in its history. Like a time machine one can go into the past or the future just by turning the shell back and forth.

According to this description, a pigmentation pattern can be captured by simulating processes taking place at the growing edge and extruding this edge along an axis representing time. For example, the following system of differential equations was proposed by Meinhardt to model the formation of the pigmentation pattern on the shell of *Nautilus pompilius* [16] (see also [15, page 61]):

$$\begin{aligned} \frac{\partial a}{\partial t} &= a' - \mu a + D_a \frac{\partial^2 a}{\partial x^2}, \\ \frac{\partial s}{\partial t} &= \sigma(x) - a' - \nu s + D_s \frac{\partial^2 s}{\partial x^2}, \end{aligned} \tag{7}$$

where

$$a' = \rho s \frac{a^2}{1 + \kappa a^2} + \rho_0, \tag{8}$$

and

$$\sigma(x) = \sigma_{min} + (\sigma_{max} - \sigma_{min}) \frac{2 \min\{x - x_{min}, x_{max} - x\}}{x_{max} - x_{min}}. \tag{9}$$

[1] The idea of modeling shell patterns using L-systems is not entirely new. Specifically, Baker and Herman generated pigmentation patterns similar to those found in *Oliva porphyria* [3] (see also [10, Chapter 18]) by applying L-systems to express a cellular automaton model proposed by Waddington and Cowe [21]. This approach preceded the formulation of the reaction-diffusion models of pigmentation, first reported in [14], and therefore did not expose the general possibility of expressing reaction-diffusion models using L-systems.

The variables a and b in Equation (7) describe concentrations of two chemical substances, called the *activator* and the *substrate*, which diffuse along the growing edge and react with each other. Equation (9) characterizes the production of the substrate $\sigma(x)$ as a triangle-shaped function of the position of the sampling point x along the edge $[x_{min}, x_{max}]$.

To solve Equation (7) using an L-system, we discretize the growing edge and represent it as a string of modules M. The production that implements the finite difference method is:

$$\begin{aligned} M(a_l, s_l, \sigma_l) < M(a, s, \sigma) > M(a_r, s_r, \sigma_r) \\ \rightarrow M\Big(a + (a' - \mu a + D_a \tfrac{a_l - 2a + a_r}{(\Delta x)^2})\Delta t, \\ s + (\sigma - a' - \nu s + D_s \tfrac{s_l - 2s + s_r}{(\Delta x)^2})\Delta t, \sigma\Big). \end{aligned} \tag{10}$$

where a' is defined by Equation (8). As in the diffusion-decay example discussed in Section 2, the complete L-system for solving Equations (7) must also create the string of modules M and assign the initial and boundary values to the variables. This includes, in particular, the values of substrate production σ, which depend on the module position in the string (Equation 9).

A solution to Equation (7) in the interval

$$[x_{min}, x_{max}] = [0, 100] \tag{11}$$

with the initial conditions $a(x, 0) = s(x, 0) = 0$ and boundary conditions $a(0, t) = a(100, t) = s(0, t) = s(100, t) = 0$, is visualized in Figure 1b. The following constants were used: $\rho = 0.5, \kappa = 1, \rho_0 = 0.004, \mu = 0.1, D_a = 0.1, \nu = 0, D_s = 0.1, \sigma_{min} = 0.012, \sigma_{max} = 0.038, \Delta x = 1$, and $\Delta t = 1$.

A realistic model of the *Nautilus* shell can be obtained assuming that the shell opening has the shape of a circle, growing exponentially from one derivation step to another, and that the axis of extrusion is coiled into a logarithmic spiral (see [7, 15] for details regarding the modeling of shell shape). Both phenomena can be easily expressed using an L-system, resulting in the model shown in Figure 2.

4 Reaction-diffusion in an expanding medium

The model of *Nautilus pompilius* extends the range of applications of L-system models to sea shells with pigmentation patterns. More generally,

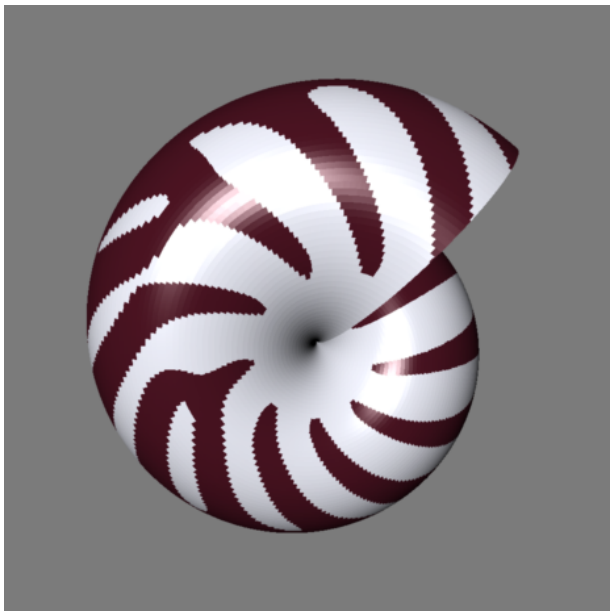


Figure 2: Model of a *Nautilus pompilius* shell

it demonstrates that reaction-diffusion processes can be expressed using L-systems. However, the integration of reaction-diffusion processes and L-systems also leads to a wider class of models of morphogenesis, characterized by reaction-diffusion taking place in expanding media.

From a historical perspective, reaction-diffusion models were originally formulated under the simplifying assumption that the medium in which diffusion takes place does not grow [20]. This assumption dominated subsequent applications of the reaction-diffusion model. Exceptions include the consideration of edge growth in models of the pigmentation pattern of selected sea shells [15, 16], a model of stripe rearrangement during growth on the skin of the fish *Pomacanthus semicirculatus* [11], and a generic model of a growing filament that maintains a constant spacing between dividing and non-dividing cells [4]. In this section we present a related model of the development of the bacteria *Anabaena catenula*.

As described by Mitchison and Wilcox [18], the cells of *Anabaena* are organized into filaments which consist of sequences of *vegetative cells* separated by *heterocysts*. The vegetative cells divide into two cells of unequal length and, in some cases, differentiate into heterocysts which do not further

divide. Due to this differentiation, the organism maintains an approximately constant spacing between heterocysts: whenever the distance between two heterocysts becomes too large due to the division and elongation of vegetative cells, a new heterocyst emerges.

What mechanisms is responsible for the differentiation of heterocysts and the maintenance of constant spacing between them? Baker and Herman [2, 3] (see also [5, 9, 12] proposed the following model. The heterocysts fix atmospheric nitrogen and transform it into nitrogenous compounds. These compounds diffuse along the filament and are used by the vegetative cells. When the level of nitrogenous compounds drops below a threshold value, the cells that detect this reduced level differentiate into heterocysts.

Although the model of Baker and Herman is capable of reproducing the observed pattern of heterocyst spacing, it is very sensitive to parameter values. Small changes in these values easily result in filaments with pairs of heterocysts appearing almost simultaneously, close to each other. This is not surprising, considering the operation of the model. The gradient of the concentration of nitrogenous compounds may be too small near the middle of a sequence of vegetative cells to precisely define the point in which a new heterocyst should differentiate. Consequently, the threshold value may be reached almost simultaneously by several neighboring cells, resulting in the differentiation of two or more heterocysts close to each other.

The described model can be improved assuming that the prospective heterocysts compete until one "wins" and suppresses the differentiation of its neighbors. This "interactive" model was originally proposed by Wilcox *et al* [22]. We formalize it using the framework of the *activator-inhibitor* class of reaction-diffusion models [13]. In addition to the nitrogenous compounds that inhibit the differentiation, the cells are assumed to carry a hypothetical substance referred to as the activator. The concentration of the activator is the criterion that distinguishes the vegetative cells (low concentration) from the heterocysts (high concentration). The activator and inhibitor are antagonistic substances: the production of the activator is suppressed by the inhibitor unless the concentration of the inhibitor is low. In that case, production of the activator drastically increases through an autocatalytic process (an increased concentration of the activator promotes its own further production). High concentration of the activator also promotes the production of the inhibitor, which diffuses to the neighboring cells. This establishes a ground for competition in which activator-producing cells attempt to suppress production of the activator in the neighboring cells. For proper values of parameters that control this process, only individual, widely

spaced cells are able to maintain the high-activation state.

An L-system implementation of these mechanisms (a variant of the L-system from [8]) is given below:

$$
\begin{array}{ll}
\omega: & M(0.5,1,200,\text{right})M(0.5,1,100,\text{right})M(0.5,1,100,\text{right}) \\
p_1: & M(s_l,a_l,h_l,p_l) < M(s,a,h,p) > M(s_r,a_r,h_r,p_r): \\
 & \quad s < s_{max}\ \&\ a < a_{th} \rightarrow M(s',a',h',p) \\
p_2: & M(s_l,a_l,h_l,p_l) < M(s,a,h,p) > M(s_r,a_r,h_r,p_r): \\
 & \quad s \geq s_{max}\ \&\ a < a_{th}\ \&\ p = \text{left} \rightarrow \\
 & \quad M(ks',a',h',\text{left})M((1-k)s',a',h',\text{right}) \\
p_3: & M(s_l,a_l,h_l,p_l) < M(s,a,h,p) > M(s_r,a_r,h_r,p_r): \\
 & \quad s \geq s_{max}\ \&\ a < a_{th}\ \&\ p = \text{right} \rightarrow \\
 & \quad M((1-k)s',a',h',\text{left})M(ks',a',h',\text{right}) \\
p_4: & M(s_l,a_l,h_l,p_l) < M(s,a,h,p) > M(s_r,a_r,h_r,p_r): \\
 & \quad a \geq a_{th} \rightarrow M(s,a',h',p)
\end{array}
\tag{12}
$$

where:

$$
\begin{array}{rcl}
s' & = & s(1+r\Delta t), \\
a' & = & a + \left(\frac{\rho}{h}(\frac{a^2}{1+\kappa a^2} + a_0) - \mu a\right)\Delta t, \\
h' & = & h + \left(\rho(\frac{a^2}{1+\kappa a^2} + h_0) - \nu h + D_h\frac{h_l+h_r-h}{sw}\right)\Delta t.
\end{array}
\tag{13}
$$

The cells are specified as modules M, where parameter s stands for cell length, a is the concentration of the activator, h is the concentration of the inhibitor, and p denotes polarity, which plays a role during cell division. All productions are context-sensitive to capture diffusion of the activator and inhibitor. It is assumed that the main barrier for the diffusion are cell walls of width w. Production p_1 characterizes growth of vegetative cells ($a < a_{th}$), controlled by the growth rate r. A cell that reaches the maximum length of s_{max} divides into two unequal daughter cells, with the lengths controlled by constant $k < 0.5$. The respective positions of the longer and shorter cells depends on the polarity p of the mother cell, as described by productions p_2 and p_3. Increase of the concentration of the activator a to or above the threshold value a_{th} indicates the emergence of a heterocyst. According to production p_4, a heterocyst does not further elongate or divide. The equations for s', a', and h' govern the exponential elongation of the cells and the activator-inhibitor interactions [13].

The operation of the model is illustrated in Figure 3. The vertical lines

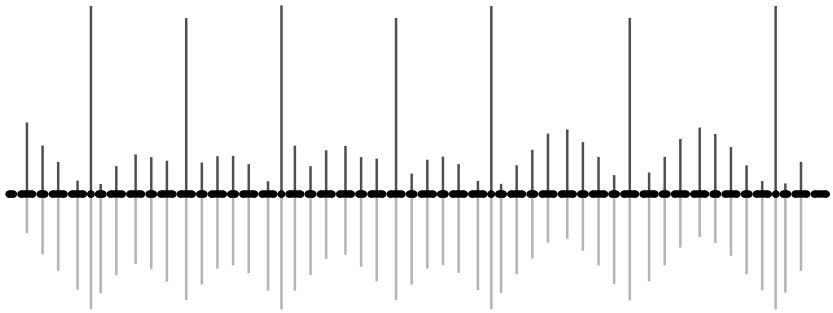


Figure 3: Fragment of a simulated filament of *Anabaena*. Vertical lines indicate the concentrations of the activator and inhibitor (above and below the cells, respectively). Notice the sharp peaks of the activator concentration that define the heterocysts, and high levels of the inhibitor concentration in the neighboring vegetative, which prevent their differentiation. The parameters used in the simulation were: $\rho = 3$, $\kappa = 0.001$, $a_0 = 0.01$, $\mu = 0.1$, $h_0 = 0.001$, $\nu = 0.45$, $D_h = 0.004$, $a_{th} = 1$, $k = 0.38196$, $s_{max} = 1$, $r = 0.002$, and $w = 0.001$.

indicate the concentrations of the activator (above the filament) and inhibitor (below the filament) associated with each cell.

It is interesting from the historical perspective that the interactive model of Wilcox *et al.* [22] and its subsequent L-system implementation [8] predicted the essential structure of the gene regulation network that controls the development of *Anabaena* filaments in nature [1]. The activator corresponds to the protein HetR, which plays a key role in the maintenance of the heterocyst state, whereas the inhibitor corresponds to the protein PatS (or a fragment of it), which diffuses across the filament and maintains the spacing between the heterocysts. The character of interactions captured by the simulation model is consistent with the postulated structure of the gene regulation network, in which HetR upregulates its own production as well as the production of PatS, whereas PatS downregulates production of HetR.

References

[1] D. G. Adams. Heterocyst formation in cyanobacteria. *Current Opinoin in Microbiology*, 3:618–624, 2000.

[2] R. Baker and G. T. Herman. CELIA — a cellular linear iterative array simulator. In *Proceedings of the Fourth Conference on Applications of Simulation* (9–11 December 1970), pages 64–73, 1970.

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

[4] J.-P. Boon and A. Noullez. Development, growth, and form in living systems. In H. E. Stanley and N. Ostrowsky, editors, *On growth and form*, pages 174–183. Martinus Nijhoff Publ., Boston, 1986.

[5] 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.

[6] L. Edelstein-Keshet. *Mathematical models in biology.* Random House, New York, 1988.

[7] D. R. Fowler, H. Meinhardt, and P. Prusinkiewicz. Modeling seashells. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26–31, 1992), in *Computer Graphics*, 26, 2 (July 1992), pages 379–387, ACM SIGGRAPH, New York, 1992.

[8] M. Hammel and P. Prusinkiewicz. Visualization of developmental processes by extrusion in space-time. In *Proceedings of Graphics Interface '96*, pages 246–258, 1996.

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

[10] 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.

[11] S. Kondo and R. Asai. A reaction-diffusion wave on the skin of the marine angelfish *Pomacanthus*. *Nature*, 376:765–768, 31 August 1995.

[12] 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.

[13] H. Meinhardt. *Models of biological pattern formation.* Academic Press, London, 1982.

[14] H. Meinhardt. Models for positional signalling, the threefold subdivision of segments and the pigmentations pattern of molluscs. *Journal of Embryology and Experimental Morphology*, 83:289–311, 1984.

[15] H. Meinhardt. *The algorithmic beauty of sea shells.* Springer-Verlag, Berlin, 1995.

[16] H. Meinhardt and M. Klinger. A model for pattern formation on the shells of molluscs. *Journal of Theoretical Biology*, 126:63–89, 1987.

[17] H. Meinhardt and M. Klinger. Pattern formation by coupled oscillations: The pigmentation patterns on the shells of molluscs. In *Lecture Notes in Biomathematics*, volume 71, pages 184–198. Springer-Verlag, Berlin, 1987.

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

[19] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C: The art of scientific computing. Second edition.* Cambridge University Press, Cambridge, 1992.

[20] A. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London B*, 237:37–72, 1952.

[21] C. H. Waddington and J. Cowe. Computer simulations of a molluscan pigmentation pattern. *Journal of Theoretical Biology*, 25:219–225, 1969.

[22] M. Wilcox, G. J. Mitchison, and R. J. Smith. Pattern formation in the blue-green alga, *Anabaena*. I. Basic mechanisms. *Journal of Cell Science*, 12:707–723, 1973.

Solving Linear Algebraic and Differential Equations with L-Systems.

Pavol Federl
University of Calgary

Przemyslaw Prusinkiewicz
University of Calgary

1. Introduction

In the previous note it was shown how L-Systems can be used to numerically solve systems of partial differential equations, for a constant or growing medium, and the method was applied to computer graphics purposes. The L-System from the previous section employed a forward Euler method for finite differencing. Although simple to implement, the forward Euler method is in many case inadequate, for example when the equations are stiff. In this note we show how an implicit method for solving differential equations can be implemented within the framework of L-Systems. At the heart of this method lies a technique for solving systems of banded linear equations. To present this method, we use analogy between the processes involved in diffusion and the behavior of electric circuits.

We begin the discussion by describing the electric circuit in Section 2, for which we wish to find the voltages and currents. Then we proceed to develop a set of continuous equations describing the relationship of currents and voltages in the circuits. The continuous time in these equations is then discretized through a Crank-Nicholson implicit finite differencing scheme, described in Section 3. The resulting discretized equations form a set of linear equations, describing the changing state of voltages and currents during a time step. Such a set of equations needs to be then solved at each time step. The set of linear equations can be represented by a 5-diagonal coefficient matrix. In Section 4 we show how such a banded system of linear equations can be effectively solved using L-Systems. In Section 5 we combine the material presented in Sections 3 and 4 into a complete L-System, and in Section 6 we discuss the results.

2. The circuit

The circuit we wish to analyze consists of n circuit segments (Figure 2) connected in series (Figure 1). Each segment contains 3 resistors (horizontal, vertical and parallel) and a capacitor. The resistances for the k-th segment are labeled R_k^H, R_k^V and R_k^P, and the capacitance is labeled C_k. The voltage $v_k(t)$ on the capacitor, or in short v_k, is a function of time. The currents passing through the horizontal, vertical and parallel resistors are $i_k(t)$, $i_k^V(t)$ and $i_k^P(t)$, respectively, while the current passing through the capacitor is $i_k^C(t)$. All 4 currents are also functions of time, and will be referred to using a short-hand notation as i_k, i_k^V, i_k^H and i_k^C, respectively.

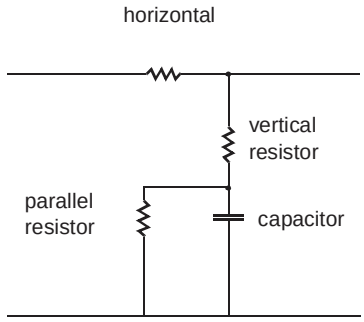


Figure 1: A circuit segment.

At the right end of the circuit, the n-th segment is left open. On the left end, the initial segment is connected to a constant voltage source (with voltage V_S) and a resistor (with resistance R_S).

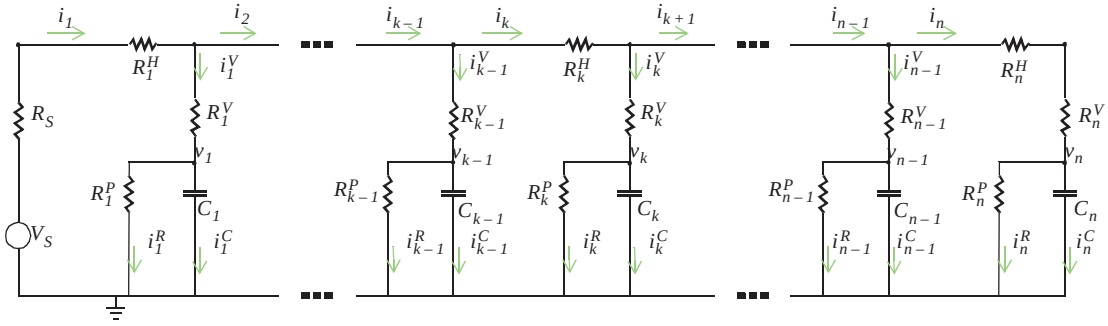


Figure 2: The overall circuit assembled from n segments connected in series.

2.1. Continuous equations

In order to analyze the relationship between the currents and voltages in the whole circuit, we consider the currents and voltages from the perspective of a single segment. There are 3 cases to consider: the general case and 2 boundary conditions. In the general case a segment is connect to other segments both on its left side and on its right side. In the boundary cases, a segment is only connected to one other segment. Since the voltage source and the initial resistor are present only on the left side of the circuit, the equations for the left and right boundary cases are derived separately. The initial conditions for the circuit are trivial, i.e. $i_k(0) = v_k(0) = 0$ for all k.

The general case:

The general case applies to a segment k when $k \in [2, n-1]$. In such cases the segment k is connected to a segment $k-1$ on the left, and to segment $k+1$ on the right. We can express the voltage v_k and current i_k in terms of currents i_{k-1} and i_{k+1}, and the voltage v_{k-1} as follows. The relationship between voltages v_{k-1} and v_k is established using Kirchoff's voltage law (KVL), i.e.:

Eq.1
$$v_k = v_{k-1} + R_{k-1}^V i_{k-1}^V - R_k^H i_k - R_k^V i_k^V .$$

From Kirchoff's current law (KCL) we know that $i_{k-1}^V = i_{k-1} - i_k$ and $i_k^V = i_k - i_{k+1}$. Substituting these into Eq. 1 results in:

$$v_k = v_{k-1} + R_{k-1}^V (i_{k-1} - i_k) - R_k^H i_k - R_k^V (i_k - i_{k+1}) \quad , \text{or}$$

$$v_k = v_{k-1} + R_{k-1}^V i_{k-1} - R_{k-1}^V i_k - R_k^H i_k - R_k^V i_k + R_k^V i_{k+1} , \text{or}$$

Eq.2
$$R_{k-1}^V i_{k-1} + v_{k-1} - (R_{k-1}^V + R_k^H + R_k^V) i_k - v_k + R_k^V i_{k+1} = 0 \quad .$$

The current passing through the capacitor is defined as a time derivative of the voltage on the capacitor, i.e.:

Eq.3
$$C_k \frac{dv_k}{dt} = i_k^C .$$

From KVL we know that $v_k - R_k^P i_k^R = 0$ and from KCL we know that $i_k^R = i_k^V - i_k^C$ and therefore $i_k^R = i_k - i_{k+1} - i_k^C$. Substituting these into Eq. 3 yields:

Eq.4
$$R_k^P C_k \frac{dv_k}{dt} = R_k^P (i_k - i_{k+1}) - v_k \quad .$$

Equations 2 and 4 provide an implicit relationship between the voltages and currents of a segment k, and the voltages and currents of its immediate neighbors: segments $k-1$ and $k+1$.

The right boundary case:

The equations for the right boundary case can be derived by considering the current i_n and voltage v_n for segment n. Since the circuit to the right of segment n is open, the equations for the right boundary case can be derived directly from the general case by setting $k = n$ and $i_{n+1} = 0$. Eq. 2 then becomes:

Eq.5 $$-R^V_{n-1} i_{n-1} - v_{n-1} + (R^V_{n-1} + R^H_n + R^V_n) i_n + v_n = 0,$$

and Eq. 4 becomes:

Eq.6 $$R^P_n C_n \frac{dv_n}{dt} = R^P_n i_n - v_n.$$

The left boundary case:

The voltage v_1 can be derived again from KVL:

$$v_1 = V_S - (R_S + R^H_1) i_1 - R^V_1 i^V_1, \text{ or}$$

$$v_1 = V_S - (R_S + R^H_1 + R^V_1) i_1 + R^V_1 i_2, \text{ or}$$

Eq.7 $$(R_S + R^H_1 + R^V_1) i_1 + v_1 - R^V_1 i_2 = V_S.$$

By following the same process for deriving Eq. 4, we arrive to the second equation for the first circuit segment:

Eq.8 $$R^P_1 C_1 \frac{dv_k}{dt} = R^P_1 (i_1 - i_2) - v_1.$$

3. Discretization of time

For a circuit with n segments, $2n$ coupled equations describe the relationships between currents and voltages. Of the $2n$ equations, 2 correspond to the left boundary case (Equations 7 and 8), $2(n-2)$ equations come from the general case (Equations 2 and 4), and 2 equations correspond to the right boundary case (Equations 5 and 6). One half of these equations is a set of regular algebraic equations, while the other half is a set of first order differential equations. An analytical solution to such a system of equations is unfeasible even for moderate values of n, and thus numerical solution becomes a necessity.

We find a numerical solution to these equations through finite differencing. To this end, we discretize the time domain into discrete time steps, each of size Δt. The solution of current $i_k(t)$ is approximated by I^m_k, where the integer m represents the discretized time t, i.e. $t = m\Delta t$. Similarly, the solution for voltage $v_k(t)$ is approximated by V^m_k. For example, the initial condition corresponding to $i_k(0) = v_k(0) = 0$ is expressed by setting

Eq.9 $$I^0_k = V^0_k = 0 \text{ for all } k.$$

The n algebraic equations (7, 2 and 5) provide a relationship between the currents and voltages at a specific time t. These equations are therefore discretized by applying the following substitutions:

Eq.10 $$i_p \to I^m_p \text{ and } v_p \to V^m_p.$$

The set of n first order differential equations are discretized using the Crank-Nicholson finite differencing scheme, using the following substitutions:

Eq.11 $$i_p \to \frac{I^m_p + I^{m-1}_p}{2}, \; v_p \to \frac{V^m_p + V^{m-1}_p}{2} \text{ and } \frac{dv_p}{dt} \to \frac{V^m_p - V^{m-1}_p}{\Delta t}.$$

The discretized equations 7, 8, 2, 4, 5 and 6 take on the following form:

Eq.12 $$(R_S + R_1^H + R_1^V)I_1^m + V_1^m - R_1^V I_2^m = V_S,$$

Eq.13 $$-\Delta t R_1^P I_1^m + (2R_1^P C_1 + \Delta t)V_1^m + \Delta t R_1^P I_2^m = \Delta t R_1^P I_1^{m-1} - \Delta t R_1^P I_2^{m-1} + (2R_1^P C_1 - \Delta t)V_1^{m-1},$$

Eq.14 $$R_{k-1}^V I_{k-1}^m + V_{k-1}^m - (R_{k-1}^V + R_k^H + R_k^V)I_k^m - V_k^m + R_k^V I_{k+1}^m = 0,$$

Eq.15 $$-\Delta t R_k^P I_k^m + (2R_k^P C_k + \Delta t)V_k^m + \Delta t R_k^P I_{k+1}^m = (2R_k^P C_k - \Delta t)V_k^{m-1} + \Delta t R_k^P I_k^{m-1} - \Delta t R_k^P I_{k+1}^{m-1},$$

Eq.16 $$R_{n-1}^V I_{n-1}^m + V_{n-1}^m - (R_{n-1}^V + R_n^H + R_n^V)I_n^m - V_n^m = 0, \text{ and}$$

Eq.17 $$-\Delta t R_n^P I_n^m + (2R_n^P C_n + \Delta t)V_n^m = (2R_n^P C_n - \Delta t)V_n^{m-1} + \Delta t R_n^P I_n^{m-1}.$$

The above equations form a set of linear equations, where the unknown values are the approximations of currents and voltages for a given time, i.e. I_1^m, V_1^m, I_2^m, V_2^m ... I_n^m, V_n^m. These unknown values are implicitly defined from the old values I_1^{m-1}, V_1^{m-1}, I_2^{m-1}, V_2^{m-1} ... I_n^{m-1}, V_n^{-1}, and from additional constants. The process of finding a numerical solution to the circuit problem consists of starting with the initial condition (a set of known values for time $m = 0$), and then iteratively finding the unknowns for the next time step.

3.1. 5-diagonal system of linear equations

The system of linear Equations 12-17 can be written in the matrix form as:

Eq.18 $$Ax = b,$$

where x is the column vector representing the solution:

Eq.19 $$x^T = \left[I_1^m\ V_1^m\ I_2^m\ V_2^m\ I_3^m\ V_3^m\ \ldots\ I_n^m\ V_n^m\right],$$

b is a column vector containing the right hand sides of the equations:

Eq.20 $$b^T = \left[B_1\ B_2\ \ldots\ B_{2n-1}\ B_{2n}\right]$$

and A is a $2n$ by $2n$ matrix containing the coefficients of the unknowns:

Eq.21 $$A = \begin{bmatrix} A_{1,1} & A_{1,2} & & A_{1,2n} \\ A_{2,1} & A_{2,2} & \ldots & A_{2,2n} \\ & \ldots & & \ldots \\ A_{2n,1} & A_{2n,2} & \ldots & A_{2n,n} \end{bmatrix}.$$

For even values of j, the entries $A_{i,j}$ represents the coefficient for the unknowns $I_{i/2}^m$, while for odd values of j, the entries $A_{i,j}$ correspond to the coefficients of $V_{\lfloor i/2 \rfloor}^m$. A quick inspection of Equations 12-17 reveals that the coefficient matrix A is 5-diagonal, i.e. that for a given row i, only entries $A_{i,i-2}$, $A_{i,i-1}$, $A_{i,i}$, $A_{i,i+1}$ and $A_{i,i+2}$ can be non-zero. The fact that matrix is 5-diagonal is important in that it allows for efficient solution to the system of linear equations. Specifically, such a system can be solved in linear time.

The techniques for solving a set of LEs can be divided into two main categories: direct and iterative methods. Direct methods solve the equations by algebraic manipulations, while iterative methods solve the equations by improving an existing solution in successive iterations. Gaussian elimination falls into the category of direct solution techniques and runs in average time of $O(n^3)$. It solves the system of LEs by successively simplifying the original system, which is achieved in two phases. In the first phase, the LEs are adjusted so that all non-zero coefficients below the diagonal are eliminated. In the second phase, the entries above the diagonal are eliminated.

The Gaussian elimination algorithm can be made more efficient when the coefficient matrix of the linear system is banded. In a banded matrix all nonzero components tend to group around the diagonal. The number describing how

well the given matrix is banded is called the bandwidth of the matrix, and it is the width of a diagonal band (or strip) which completely encompasses all non-zero elements of a matrix. A 5-diagonal matrix has a bandwidth equal to 5. If the bandwidth of a given matrix is m, then the Gaussian elimination algorithm can be modified so that the running time is $O(m^2 n)$. This is achieved by modifying the original Gaussian elimination algorithm to perform row subtractions only in the areas where there are non-zero entries. The running time of Gaussian elimination on a 5-diagonal system of LEs is therefore $O(n)$.

4. Solving 5-diagonal systems of linear equations using L-Systems

Here we show how L-Systems can be used to solve a system of n linear equations which in matrix form can be written as $Ax = b$, and when the coefficient matrix A is a 5 diagonal matrix of the form:

$$A = \begin{bmatrix} a_{1,3} & a_{1,4} & a_{1,5} & 0 & 0 & 0 \\ a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & 0 & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & 0 \\ 0 & a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\vdots \qquad \cdots$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} a_{n-3,1} & a_{n-3,2} & a_{n-3,3} & a_{n-3,4} & a_{n-3,5} & 0 \\ 0 & a_{n-2,1} & a_{n-2,2} & a_{n-2,3} & a_{n-2,4} & a_{n-2,5} \\ 0 & 0 & a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & a_{n-1,4} \\ 0 & 0 & 0 & a_{n,1} & a_{n,2} & a_{n,3} \end{bmatrix}$$

The column vector $x = [x_1 \ldots x_n]^T$ represents the n unknowns, and the column vector $b = [b_1 \ldots b_n]^T$ represents the right hand sides of the equations. To represent the system of equations using an L-System string, we use a string of n modules `M`. Each module `M` has a set of values associated with it, representing all non-zero coefficients of a single row of the matrix, plus the corresponding entry of the solution vector b. The values are grouped into a single parameter of type `struct Row`:

```
struct Row
{
     double a1, a2, a3, a4, a5, rhs;
     Row () { a1 = a2 = a3 = a4 = a5 = rhs = 0.0; }
     Row (double p1, double p2, double p3, double p4, double p5, double pb)
            { a1 = p1; a2 = p2; a3 = p3; a4 = p4; a5 = p5; rhs = pb;}
};
```

The first module `M` represents the first row of the matrix A and the column vector b, the second module `M` represents the second row, etc. For example, consider the following system of 6 linear equations of 6 unknowns:

$$
\begin{aligned}
7x_1 + 8x_2 - 3x_3 &= 10 \\
4x_1 - x_2 + 3x_3 + 4x_4 &= -2 \\
-x_1 + x_3 + x_5 &= 7 \\
x_2 - 2x_3 + x_4 &= 0 \\
x_3 + 2x_4 + 3x_5 + 4x_6 &= 5 \\
2x_2 + 4x_5 + 6x_6 &= 8
\end{aligned}
$$

This system, in matrix form can be written as: $Ax = b$, where

$$
A = \begin{bmatrix} 7 & 8 & -3 & 0 & 0 & 0 \\ 4 & -1 & 3 & 4 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 2 & 4 & 6 \end{bmatrix}, \; x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \text{ and } b = \begin{bmatrix} 10 \\ -2 \\ 7 \\ 0 \\ 5 \\ 8 \end{bmatrix}.
$$

The L-System string of modules representing such a system would be:

```
M(Row(0,0,7,8,-3,10)) M(Row(0,4,-1,3,4,-2)) M(Row(-1,0,1,0,1,7)) M(Row(1,-2,1,0,0,0))
M(Row(1,2,3,4,0,5)) M(Row(2,4,6,0,0,8))
```

4.1. First phase - elimination below diagonal

As described in the previous section, the solution to the system of linear equation is found by performing a two-phase process. In the first phase, the coefficients below the diagonal are eliminated. This corresponds to re-writing each module `M` of the string corresponding to rows $2...n$, processing the string from left to right. Let us assume the first $k-1$ modules have been already rewritten, i.e. the first $k-1$ rows of A already have 0 's below the diagonal:

$$
A = \begin{bmatrix} & & & \cdots & & & \\ \cdots & 0 & a_{k-2,3} & a_{k-2,4} & a_{k-2,5} & 0 & 0 & 0 \\ & 0 & 0 & a_{k-1,3} & a_{k-1,4} & a_{k-1,5} & 0 & 0 \\ & 0 & a_{k,1} & a_{k,2} & a_{k,3} & a_{k,4} & a_{k,5} & 0 \\ & & & \cdots & & & & \end{bmatrix} \cdots
$$

To adjust the row k so that its two non-zero entries below diagonal are eliminated, proper multiples of rows $k-2$ and $k-1$ have to be subtracted from row k. This is achieved by simultaneously adjusting the coefficients of row k as follows:

Eq.22

$$\left.\begin{aligned} a_{k,1} &\rightarrow 0 \\ a_{k,2} &\rightarrow 0 \\ a_{k,3} &\rightarrow a_{k,3} - f_1 a_{k-2,4} - f_2 a_{k-1,4} \\ a_{k,4} &\rightarrow a_{k,4} - f_2 a_{k-1,5} \\ a_{k,5} &\rightarrow a_{k,5} \\ b_k &\rightarrow b_k - f_1 b_{k-2} - f_2 b_{k-1} \end{aligned}\right\} \text{ where } \begin{aligned} f_1 &= \frac{a_{k,1}}{a_{k-2,3}} \\ f_2 &= \frac{a_{k-2,3}a_{k,2} - a_{k-2,4}a_{k,1}}{a_{k-2,3}a_{k-1,3}} \end{aligned} .$$

The above substitutions cannot be applied to row 2, as there is only a single row above it. One solution is to treat row 2 as a special case. Another solution, which does not require handling of a special case, is to include a phony row 0, with coefficients $a_{0,3} = 1$ and $a_{0,1} = a_{0,2} = a_{0,4} = a_{0,5} = b_0 = 0$.

The L-System production that performs the pass from left to right - eliminating all entries in the coefficient matrix below the diagonal - is shown below:

```
M(r1) M(r2) << M(r3) :
{
     if (phase == LEFT_TO_RIGHT)
     {
             double f1 = r3.a1 / r1.a3;
             double f2 = (r1.a3 * r3.a2 - r1.a4 * r3.a1) / (r1.a3 * r2.a3);
             produce M(Row(0, 0, r3.a3-f1*r1.a5-f2*r2.a4, r3.a4-f2*r2.a5
                         , r3.a5, r3.rhs-f1*r1.rhs-f2*r2.rhs));
     }
}
```

This production effectively replaces each row with a new row, by subtracting from it the proper multiples of the 2 rows above it. In the end, the replaced row contains 0's to the left of the diagonal coefficient. The production rule uses fresh left context to gain access to the already modified 2 rows above the row to be adjusted. By using fresh left context, this production rule is applied to the modules in a single pass.

4.2. Second phase - elimination above diagonal

Once the first phase is completed, the coefficient matrix has the form:

$$A = \left[\begin{array}{|c|c|c|c|c|c|c|}\hline a_{1,3} & a_{1,4} & a_{1,5} & 0 & 0 & 0 & 0 \\ \hline 0 & a_{2,3} & a_{2,4} & a_{2,5} & 0 & 0 & 0 \\ \hline 0 & 0 & a_{3,3} & a_{3,4} & a_{3,5} & 0 & 0 \\ \hline \end{array}\right. \begin{array}{c} \vdots \\ \\ \\ \end{array} \left.\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array}\right]$$

$$\cdots$$

$$\left[\begin{array}{|c|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & a_{7,3} & a_{7,4} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & a_{n,3} \\ \hline \end{array}\right]$$

To finish the process of finding the solution, the coefficients above the diagonal are eliminated in the second phase. This is achieved by processing the rows from bottom to top, subtracting from each row the appropriate multiples of the two rows below. Assuming that k rows remain to be processed, the k-th row is adjusted by subtracting from it multiples of rows $k + 1$ and $k + 2$:

Eq.23

$$\left.\begin{aligned} a_{k,1} &\to 0 \\ a_{k,2} &\to 0 \\ a_{k,3} &\to a_{k,3} \\ a_{k,4} &\to 0 \\ a_{k,5} &\to 0 \\ b_k &\to b_k - f_1 b_{k+1} - f_2 b_{k+2} \end{aligned}\right\} \text{ where } \begin{aligned} f_1 &= \frac{a_{k,4}}{a_{k+1,3}} \\ f_2 &= \frac{a_{k,5}}{a_{k+2,3}} \end{aligned} .$$

Again, the above substitutions cannot be applied to row $n-1$, as it does not have two rows below it. In order to avoid writing an extra rule handling a special case, we add instead a phone row $n+1$ with the same coefficients as row 0, i.e. $a_{n+1,3} = 1$ and $a_{n+1,1} = a_{n+1,2} = a_{n+1,4} = a_{n+1,5} = b_{n+1} = 0$. The production rule which effects the second phase is:

```
M(r1) >> M(r2) M(r3) :
{
     if (phase == RIGHT_TO_LEFT)
     {
             produce M(Row(0, 0, r1.a3, 0, 0
                         , r1.rhs - r1.a4*r2.rhs/r2.a3 - r1.a5*r3.rhs/r3.a3));
     }
}
```

Similar to the production rule used in the left-to-right phase, this right-to-left production rule also uses fresh context. Combined with processing the string of modules from right-to-left, this production rule is applied to the whole string in a single pass. After the second phase is finished, the coefficient matrix has the form:

$$A = \begin{bmatrix} a_{1,3} & 0 & 0 & 0 & 0 & \vdots & 0 & 0 \\ 0 & a_{2,3} & 0 & 0 & 0 & & 0 & 0 \\ 0 & 0 & a_{3,3} & 0 & 0 & & 0 & 0 \\ & & & & & \cdots & & \\ 0 & 0 & 0 & 0 & 0 & & a_{7,3} & 0 \\ 0 & 0 & 0 & 0 & 0 & & 0 & a_{n,3} \end{bmatrix}$$

Since non-zero entries are only on the diagonal, retrieving the solution is trivial, i.e. $x_i = b_i / a_{i,3}$.

5. The complete L-System implementation

In Section 3 we have shown how the continuous equations describing the voltages and currents in the circuit have been discretized. The result of such a discretization is a set of linear equations, which must be solved at each time step. Since the set of linear equations is represented by a 5-diagonal coefficient matrix, they can be solved using L-Systems, as demonstrated in Section 4. In this Section we describe a complete L-System implementation of the solution (the complete source is given at the end of this report).

The overall operation of the L-System can be distinguished into 5 distinct phases:

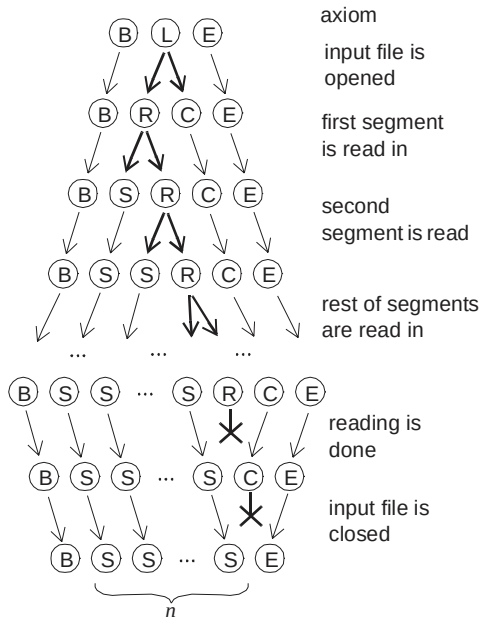
- Phase 0: a data-file describing the circuit is loaded;
- Phase I: the 5 diagonal coefficient matrix and the right hand-side is set up;

- Phase II: the entries below diagonal are eliminated;
- Phase III: the entries above diagonal are eliminated;
- Phase IV: the solution is extracted and simulation time is advanced.

Once the circuit has been loaded from the file, the L-System cycles through phases I-IV. Phase I, II and IV require the string processing to be done from left-to-right, and phase III needs the string to be processed from right-to-left. Since different production rules need to be applied in different phases, a global variable `phase` is used to denote the current phase. The symbolic names of the 4 phases are `SETUP`, `LEFT_TO_RIGHT`, `RIGHT_TO_LEFT` and `COLLECT`. Initially, `phase` is set to `SETUP` (line 53). At the end of each string rewrite, the `phase` is adjusted to reflect the next stage (lines 55-63). Notice that for phase III the string processing is reversed (line 60).

Phase 0: reading in the data-file

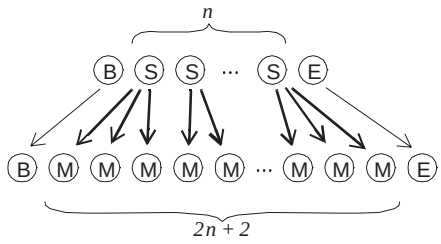
The L-System string is initialized through an axiom (line 65), to contain 3 modules: `B L` and `E`. Modules `B` and `E` denote the beginning and the end of the string, respectively, which are used to determine boundary case conditions. The parameter of module `L` is a string, which specifies the data-file from which the circuit will be loaded. Through decomposition rules (lines 68-98), the data-file is first opened, and then read in segment by segment. At the end of the decomposition, the string has the form "`B S S ... S E`". There is one module S for each circuit segment. The graphical representation of this process is illustrated in the figure on the right.



Each module `S` has a parameter of type struct Segment (defined on lines 17-21). The fields `Rh`, `Rv`, `Rp` and `Cap` are read in from the file, and represent the 3 resistances and a capacitance of the segment. The fields `I` and `V` contain the calculated current and voltage in the segment, and are both initialized to 0 to reflect the initial condition (Eq. 9).

Phase I: setting up the matrix representation

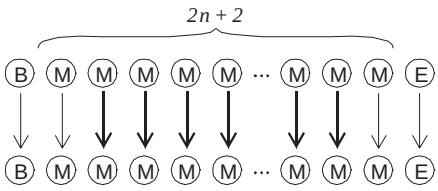
In phase I, the string is rewritten to represent the system of 5-diagonal linear equations, implemented by the productions of group `SETUP` (lines 103-120). This is achieved by replacing each module `S` with 2 modules `M`, where each module `M` represents a row in the coefficient matrix A and the corresponding row of the right hand side column vector b, as described in Section 4. If there are n segments in the circuit, there would be $2n + 2$ rows (or modules `M`).



The production rule on lines 103-109 corresponds to the general case, and is applied to all segments that have both neighbors. This rule is derived directly from Eq. 14 and Eq. 15. The production rule on lines 110-115 reflects the left-boundary case, and is only applied to the very first segment (represented by module `S` that immediately follows module `B`). The left boundary case production rule was derived to reflect Eq. 12 and Eq. 13. Finally, the right boundary case (lines 116-120) is applied to the right-most segment, and corresponds to Eq. 16 and Eq. 17. The right-most segment is determined by requiring the right context of the module `S` to be module `E`. Finally, notice how the extra rows are appended at the beginning and at the end of the string (line 111 and line 119).

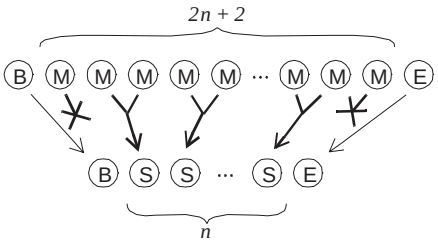
Phase II and Phase III: solving the system of equations

In phases II and III the string of modules is rewritten to represent a system of equations, where the corresponding coefficient matrix has non-zero entries only on its diagonal (lines 121-132). These two phases have been explained in detail in Section 4.



Phase IV: extracting the solution

In the last phase, the solution is extracted from the string (lines 133-148). First, the extra rows are eliminated by the production rules on lines 135-140. Then, for every pair of modules `M`, a modules `S` is produced (lines 141-145). The current for the produced module S is calculated from the first `M`, while the voltage is calculated from the second `M`.



Rendering

The circuit is rendered each time at the end of phase IV, done by the interpretation rules (lines 150-228). All of the results presented in the next section were rendered using this L-System, both the circuit and the graphs.

6. Results and conclusions

The output of the L-System program for a simple circuit composed of 4 different segments is shown in Figure 3. At the top of Figure 3 the circuit is rendered. At the bottom of the figure, the graphs of voltages and currents are displayed at 5 different points in time.

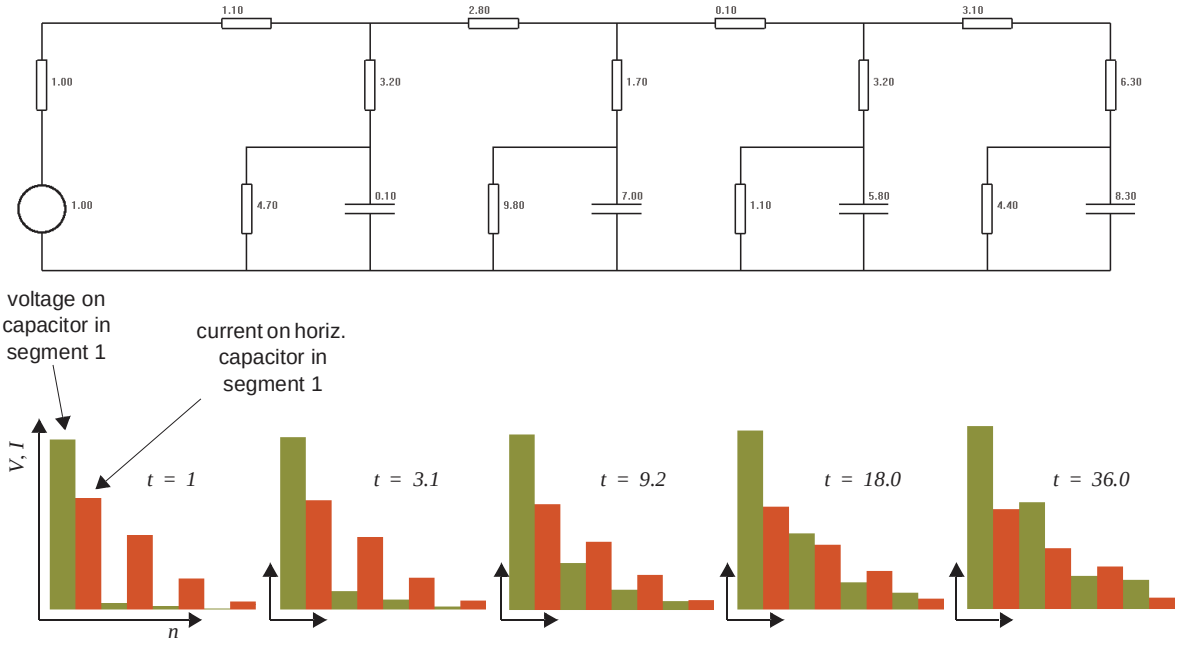


Figure 3: Example I - simple circuit composed of 4 different segments.

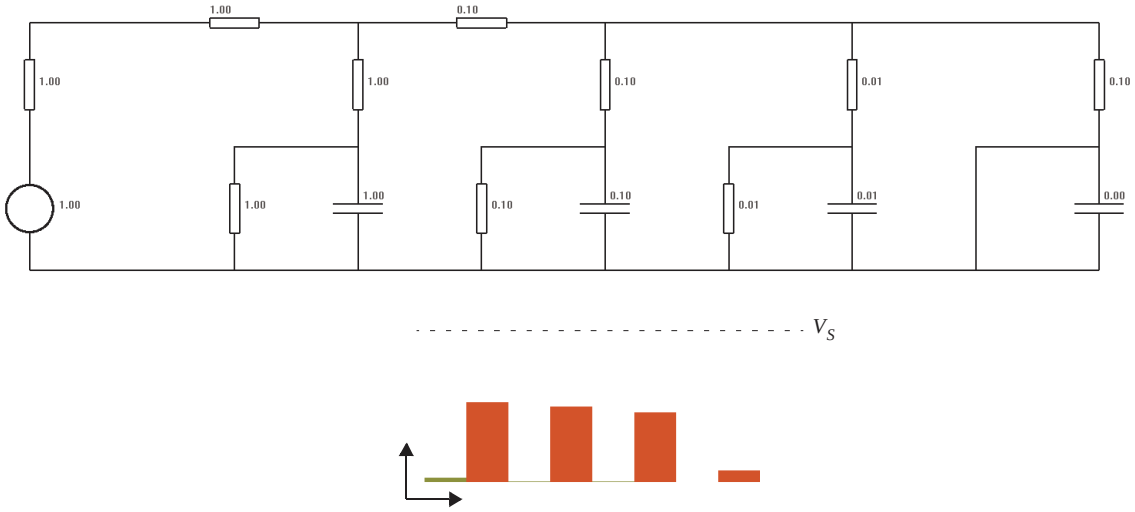


Figure 4: Example II - circuit with some resistances set to 0.

The next example illustrates the robustness of the presented method for solving the circuit analysis problem. The circuit in Figure 4 contains resistances and capacitances differing in magnitudes, some of which are even set to 0. Such a circuit leads to stiff equations, which are impossible to solve using forward integration methods. Our implicit method however, solves the problem successfully.

The circuit in Figure 5 is set up to simulate diffusion. It is composed of 50 identical segments. Because of the presence of the parallel resistors, less and less current propagates to the capacitors toward the right-hand-side of the circuit. As a result, the further the capacitor is from the voltage source, the less it will be charged. This is very similar to diffusion with decay, where the concentration of the chemical decreases as the distance from the source increases.

Finally, we have also simulated a circuit analogous to diffusion without decay, illustrated in Figure 6. This was achieved by setting the relative differences between the resistances of the parallel resistors and the other two types of resistors very high. As a result, given enough time - all capacitors are eventually charged to the same level.

In conclusion, we have successfully demonstrated how L-Systems can be used to solve systems of differential equations, while employing implicit finite differencing scheme. The core of this approach lies in the method of solving banded systems of linear equations using L-Systems, which we described in detail. Using fresh contexts, we were able to implement the solution to solve the equations in only two passes. The resulting L-System performs almost as fast as a straight C++ implementation, proving that L-Systems are a viable mechanism for solving systems of differential equations in modeling problems.

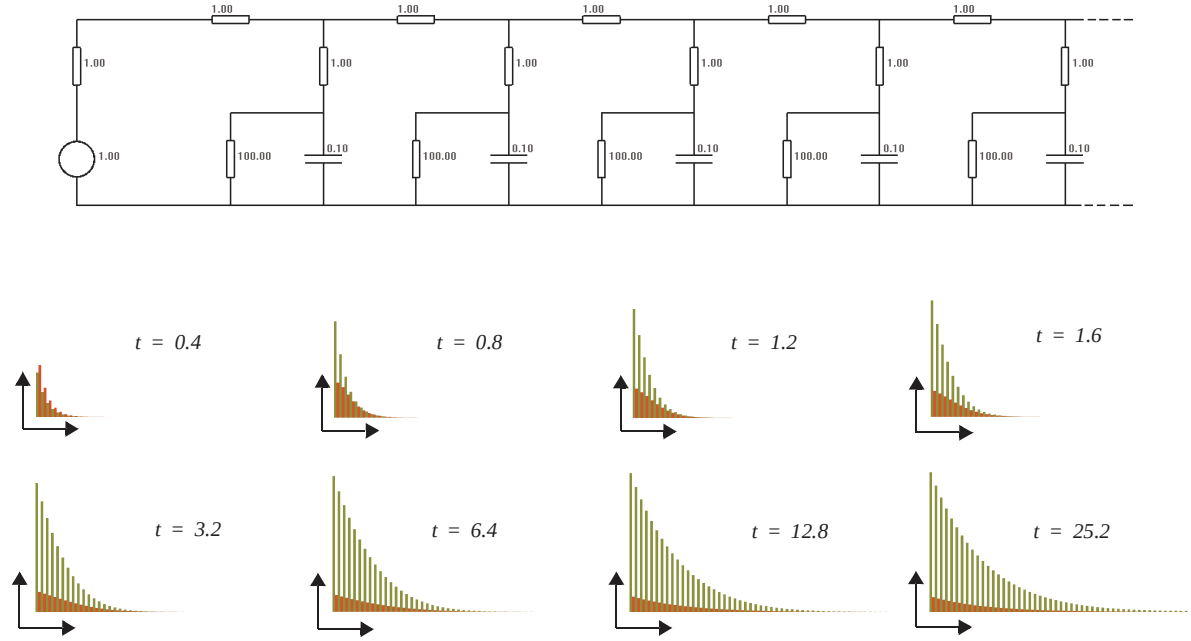


Figure 5: Example III - simulated diffusion, with decay.

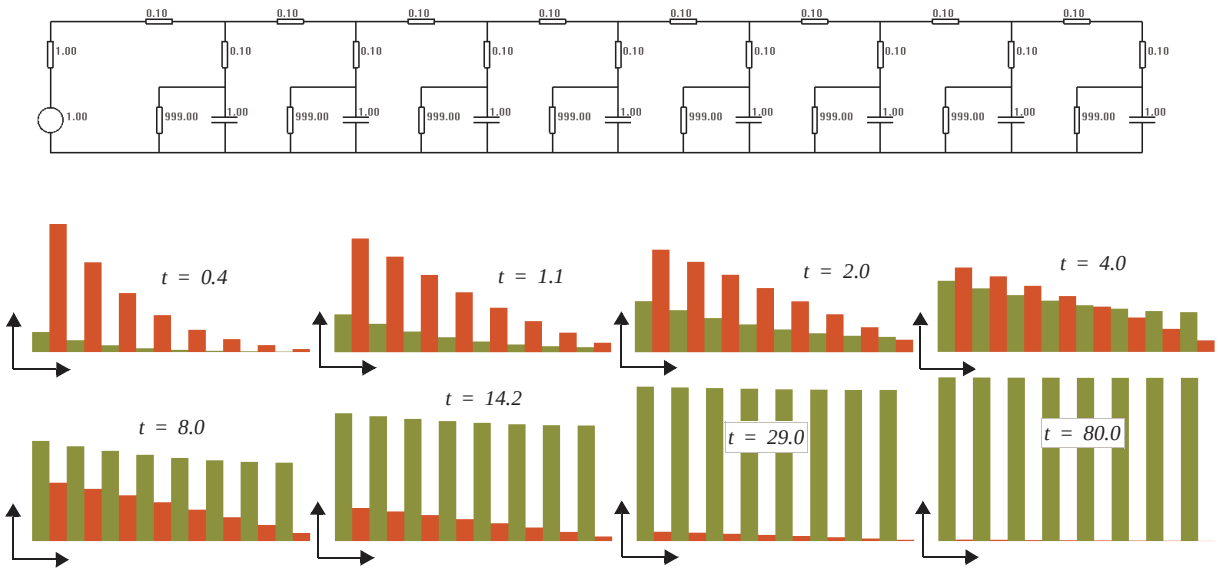


Figure 6: Example III - simulated diffusion without decay.

Appendix A: The complete L+C source code

```
#include <cmath>
#include <cstdlib>
#include <lpfgall.h>
#include <cstdio>
#include <cstdlib>
#include <string>
#include <cassert>
#include <stdarg.h>

using std::string;

const string fname = "circuit-1.dat"; FILE * fp;
double dt, curr_time, Vs, Rs; // time step, curr. time, Rs & Vs
float x, vscale;// used for rendering
bool draw_circuit;// whether to draw circuit

struct Segment
{   double Rh, Rv, Rp, Cap;// the resistances and the capacitance of a single segment
    double I, V;// current and voltage
    Segment () { Rh = Rv = Rp = Cap = I = V = 0.0; }
};

struct Row
{   double a1, a2, a3, a4, a5, rhs;
    Segment seg;
    Row () { a1 = a2 = a3 = a4 = a5 = rhs = 0.0; }
    Row (double pa1, double pa2, double pa3, double pa4, double pa5, double prhs, Segment & pseg)
       { a1 = pa1; a2 = pa2; a3 = pa3; a4 = pa4; a5 = pa5; rhs = prhs; seg = pseg;}
};

module B();                              // marks the beginning of the string
module E();                              // marks the end of the string
module L(string);                        // module that will load the file
module C();                              // closes the file
module R(long);                          // reads the file
module S(Segment);                       // contains the information about the segment
module M(Row);                           // represents one row of coeff. matrix & RHS
module Capacitor(double, double, double);// renders a capacitor
module ResistorV(double, double, double);// renders a vertical resistor
module ResistorH(double, double, double);// renders a horizontal resistor
module Emf(double, double, double);      // renders EMF
module Rectangle(double, double, double, double); // draw empty rectangle
module RectangleF(double, double, double, double); // draw filled rectangle
module LabS(double, double, string);     // draw a string

// phases of computation
#define SETUP 1
#define LEFT_TO_RIGHT 2
#define RIGHT_TO_LEFT 3
#define COLLECT 4
int phase;

Start: {phase = SETUP;Forward(); }
StartEach: {UseGroup (phase); }
EndEach:
{   switch (phase)
    {
    case SETUP:          phase = LEFT_TO_RIGHT;  Forward();   break;
    case LEFT_TO_RIGHT:  phase = RIGHT_TO_LEFT;  Backward();  break;
    case RIGHT_TO_LEFT:  phase = COLLECT;        Forward();   break;
    case COLLECT:        phase = SETUP;          Forward();   break;
    }
}

   Axiom: B() L(fname) E();
// ============================================================
```

```
67
68    decomposition:
69    maximum depth: 1000;
70 // ============================================================
71 L(fname) : // open the file for reading
72 {   fp = fopen (fname . c_str (), "r");
73     bool error = (fp == NULL);
74     long nseg;
75     error = error || (4 != fscanf (fp, "%lf %ld %lf %lf", & dt, & nseg, & Vs, & Rs));
76     if (error){
77        Printf ("Cannot open/read file %s.\n", fname . c_str ());
78        produce ;
79     }
80     draw_circuit = nseg < 10;
81     vscale = nseg;
82     curr_time = 0.0;
83     produce R(nseg) C();
84 }
85 C() : // Close the file
86 {   fclose (fp);
87     produce ;
88 }
89 R(n) : // Read another segment from the file
90 {   if (n == 0) produce ;
91     Segment s;
92     if (4 != fscanf (fp, "%lf %lf %lf %lf", & s.Rh, & s.Rv, & s.Rp, & s.Cap))
93     {
94        Printf ("Cannot read segment.\n");
95        produce ;
96     }
97     produce S(s) R(n-1);
98 }
99
100    production:
101    derivation length: 4;
102 // ============================================================
103    group SETUP:
104 // ----------------------------------------
105 S(sL) < S(sC) > S(sR) : // general case
106 {   produce M (Row (-sL.Rv, -1, sL.Rv+sC.Rh+sC.Rv, 1, -sC.Rv, 0, sC))
107        M (Row (0, -dt*sC.Rp, 2*sC.Rp*sC.Cap+dt, dt*sC.Rp, 0
108               , (2*sC.Rp*sC.Cap-dt)*sC.V+dt*sC.Rp*sC.I-dt*sC.Rp*sR.I, sC));
109 }
110 B() < S(sC) > S(sR) : // left boundary case
111 {   produce M (Row (0,0,1,0,0,0,sC))
112        M (Row (0, 0, Rs+sC.Rh+sC.Rv, 1,-sC.Rv, Vs, sC))
113        M (Row (0, -dt*sC.Rp, 2*sC.Rp*sC.Cap+dt, dt*sC.Rp, 0
114               , (2*sC.Rp*sC.Cap-dt)*sC.V+dt*sC.Rp*sC.I-dt*sC.Rp*sR.I, sC));
115 }
116 S(sL) < S(sC) > E() : // right boundary case
117 {   produce M (Row (-sL.Rv, -1, sL.Rv+sC.Rh+sC.Rv, 1, 0, 0, sC))
118        M (Row (0, -dt*sC.Rp, 2*sC.Rp*sC.Cap+dt, 0, 0, (2*sC.Rp*sC.Cap-dt)*sC.V+dt*sC.Rp*sC.I, sC))
119        M (Row (0,0,1,0,0,0,sC));
120 }
121    group LEFT_TO_RIGHT:
122 // ----------------------------------------
123 M(r1) M(r2) << M(r3) :
124 {   double k1 = r3.a1 / r1.a3;
125     double k2 = (r1.a3 * r3.a2 - r1.a4 * r3.a1) / (r1.a3 * r2.a3);
126     produce M(Row(0, 0, r3.a3-k1*r1.a5-k2*r2.a4, r3.a4-k2*r2.a5, r3.a5, r3.rhs-k1*r1.rhs-k2*r2.rhs,
r3.seg));
127 }
128    group RIGHT_TO_LEFT:
129 // ----------------------------------------
130 M(r1) >> M(r2) M(r3) :
131 {   produce M(Row(0, 0, r1.a3, 0, 0, r1.rhs - r1.a4 * r2.rhs / r2.a3 - r1.a5 * r3.rhs / r3.a3, r1.seg));
132 }
133    group COLLECT:
```

```
// ---------------------------------------
B() < M(r) : // discard the first phony row
{   produce ;
}
M(r) > E() : // discard the last phony row
{   produce ;
}
M(r1) M(r2) : // convert two equations to a circuit segment
{   r1.seg.I = r1.rhs / r1.a3;
    r1.seg.V = r2.rhs / r2.a3;
    produce S (r1.seg);
}
E() :
{   curr_time += dt;
}

   interpretation:
   maximum depth: 1000;
// ============================================================
B() : // draw the intial voltage & resistor
{   nproduce SetWidth(2);
    x = -1;
    if (draw_circuit)
    {   nproduce SetColor(7)
            Line2d(V2d(-0.33,0),V2d(-0.33,1))
            Line2d(V2d(-0.33,1),V2d(0,1))
            Line2d(V2d(-0.33,0),V2d(0,0))
            ResistorV(-0.33,0.75,Rs)
            Emf (-0.33, 0.25, Vs);
    }
    produce SetColor(6) Line2d(V2d(0,1.2),V2d(0,1.2+vscale*Vs));
}
S(s) : // draw the segment and corresponding portion of the graph
{   x = x + 1;
    if (draw_circuit)
    {   static char buff1 [4096]; sprintf (buff1, "I=%.3f", s.I);
        static char buff2 [4096]; sprintf (buff2, "V=%.3f", s.V);
        nproduce SetColor (1)
            Line2d(V2d(x,1),V2d(x+1,1))
            Line2d(V2d(x+1,1),V2d(x+1,0))
            Line2d(V2d(x+0.5,0.5),V2d(x+1,0.5))
            Line2d(V2d(x+0.5,0.5),V2d(x+0.5,0))
            Line2d(V2d(x,0),V2d(x+1,0))
            Capacitor(x+1,0.25,s.Cap)
            ResistorV(x+0.5,0.25,s.Rp)
            ResistorV(x+1,0.75,s.Rv)
            ResistorH(x+0.5,1,s.Rh)
            SetColor(5) MoveTo(x+0.65,1.02,0) Label(buff1)
            SetColor(4) MoveTo(x+1.04,0.5,0) Label(buff2);
    }
    produceSetColor(6) Line2d(V2d(x,1.2),V2d(x+1,1.2))   // axis
        SetColor(5) RectangleF(x+0.5,1.2,x+1,1.2+vscale*s.I)// Render calculated current
        SetColor(4) RectangleF(x,1.2,x+0.5,1.2+vscale*s.V);// Render calculated voltage
}
E() : // draw the time
{   static char buff [4096]; sprintf (buff, "Time: %.3f", curr_time);
    produce SetColor(1) LabS (vscale,1.2,buff);
}
Capacitor(cx, cy, val) : // draw a capacitor
{   static char buff [4096]; sprintf (buff, "%.2f", val);
    produce SetColor(2) RectangleF(cx-0.1,cy-0.02,cx+0.1,cy+0.02)
        SetColor(1) Line2d(V2d(cx-0.1,cy-0.02),V2d(cx+0.1,cy-0.02))
        Line2d(V2d(cx-0.1,cy+0.02),V2d(cx+0.1,cy+0.02))
        SetColor(6) LabS(cx+0.02,cy+0.04,buff);
}
ResistorV(cx, cy, val) : // draw a vertical resistor
{   if (val == 0) produce ;
    static char buff [4096]; sprintf (buff, "%.2f", val);
```

```
202     produce SetColor(2) RectangleF(cx-0.02,cy-0.1,cx+0.02,cy+0.1)
203         SetColor(1) Rectangle (cx-0.02,cy-0.1,cx+0.02,cy+0.1)
204         SetColor(6) LabS(cx+0.04,cy,buff);
205 }
206 ResistorH(cx, cy, val) : // draw a horizontal resistor
207 {   if (val == 0) produce ;
208     static char buff [4096]; sprintf (buff, "%.2f", val);
209     produce SetColor(2) RectangleF(cx-0.1,cy-0.02,cx+0.1,cy+0.02)
210         SetColor(1) Rectangle (cx-0.1,cy-0.02,cx+0.1,cy+0.02)
211         SetColor(6) LabS(cx-0.1,cy+0.04,buff);
212 }
213 Emf cx, cy, val) : // draw EMF
214 {   static char buff [4096]; sprintf (buff, "%.2f", val);
215     produce SetColor(1) MoveTo(cx,cy,0) Circle(0.1) SetColor(2) Circle(0.09)
216         SetColor(6) LabS(cx+0.12,cy,buff);
217 }
218 RectangleF(x1, y1, x2, y2) : // draw filled rectangle
219 {   produce SP () MoveTo(x1,y1,0) PP() MoveTo(x2,y1,0) PP() MoveTo(x2,y2,0) PP()
220         MoveTo(x1,y2,0) PP() EP ();
221 }
222 Rectangle(x1, y1, x2, y2) : // draw outline of a rectangle
223 {   produce Line2d (V2d (x1, y1), V2d (x2, y1)) Line2d (V2d (x2, y1), V2d (x2, y2))
224         Line2d (V2d (x2, y2), V2d (x1, y2)) Line2d (V2d (x1, y2), V2d (x1, y1));
225 }
226 LabS(x, y, s) : // draw label
227 {   produce MoveTo(x,y,0) Label(s.c_str ());
228 }
```

Integrating biomechanics into developmental plant models expressed using L-systems[1]

C. Jirasek [1], P. Prusinkiewicz [1] and B. Moulia [2]

[1] Department of Computer Science, University of Calgary, Alberta, Canada
[2] INRA, Station d'écophysiologie des plantes fourragères, Lusignan, France

Abstract

We present a method for incorporating the biomechanical model of the bending of branch axes introduced by Schaffer and Fournier *et al*. into developmental plant models expressed using L-systems. The models capture the impact of gravity, tropisms, contact between elements of a plant structure and contact with obstacles on the shape of branches. Sample plants modeled using this technique are compared with photographs of real plants.

Introduction

Plant architecture and its coupling with the environment play an essential role in the colonization of space by plants (see review in [9]). Consequently, comprehensive functional-structural plant models take into account physical, biological, and environmental processes that influence plant development. L-systems [14,15] provide a convenient theoretical and programming framework for the architectural modeling of plants, and have been used to model a variety of interactions between plants and their environment. Examples include the effects of local light on the development of the aerial architecture of plants, and the effects of water availability on root growth [9]. Nevertheless, the effects of gravity, tropisms, and contacts between organs have been captured by L-system models only in a simple manner [15]. The objective of our current work is to improve the representation of branch shape in L-system models by including the combined effects of gravity and tropisms according to the current state of the biomechanical analysis of these phenomena. Our approach is based on the model of axis growth and reorientation introduced by Schaffer [16] and Fournier *et al*. [4]. This model predicts a sigmoidal shape of branch axes by combining the notion of the gravitropic set angle (GSA) [2] with the laws of the theory of elasticity [8] applied to longitudinally and radially growing axes. In particular, it incorporates incremental changes in the amount of load-bearing material due to secondary growth, and captures the resulting "memorization" of branch shape [4,16].

[1] Published in: H.-Ch. Spatz and T. Speck (Eds.): Plant biomechanics 2000. Proceedings of the 3rd Plant Biomechanics Conference Freiburg-Badenweiler, August 27 to September 2, 2000. Georg Thieme Verlag, Stuttgart, pp. 615-624

A parallel technique for including biomechanical factors into architectural tree models has been proposed in the scope of the AMAP modeling system [3,5]. The biomechanical component of that model was implemented as an external module using the finite element method. In contrast, we incorporate the effects of weight and gravitropism on branch shape directly into the framework of L-systems. As a result, the system of equations that describes the biomechanical aspects of a plant becomes an inherent part of the model, and is automatically updated as the plant develops. Metaphorically speaking, the system of equations grows with the modeled plant. The proposed method makes it possible to address questions concerning plant axis shape that combine biomechanics with biological regulatory mechanisms [4,5] and with the trade-offs between various functions of the axes [12].

Mechanical model of a branch axis

We conceptualize the branch axis as an inextensible elastic rod of length L, with natural parameter $s \in [0,L]$ denoting the arc-length distance of a point **P** from the base of the rod. Each point **P** is associated with a local frame of reference defined by mutually orthogonal unit vectors **H**, **L** and **U** (heading, left, and up). We assume that vector **H** is tangent to the rod axis and vectors **L** and **U** are aligned with the principal axes of the cross-section of the rod (Figure 1). In a straight prismatic rod each local reference frame will be parallel to all others. In general, two successive reference frames separated by an infinitesimal rod segment of length ds may be rotated through an infinitesimal angular vector d$\mathbf{\Phi}$, which characterizes the curvature and twist at point **P**. A rod, and specifically a growing plant axis, may possess curvature and twist in the unloaded state [4]; we denote the corresponding angle of rotation at **P** by d$\underline{\mathbf{\Phi}}$. We call $\mathbf{\Omega} = d\mathbf{\Phi}/ds$ and $\underline{\mathbf{\Omega}} = d\underline{\mathbf{\Phi}}/ds$ the rates of rotation of the reference frame **HLU** along the rod, although s is a spatial coordinate and not time.

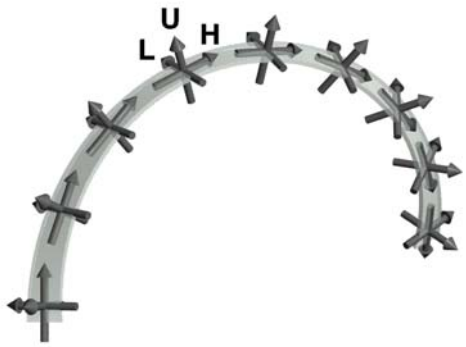


Figure 1. Local **HLU** frames of a sample rod

Given vectors **H**(0), **L**(0) and **U**(0) specifying the initial reference frame at s=0, the rate of rotation $\mathbf{\Omega}$ determines the reference frame **HLU** at any point on the rod as the solution to the differential equations:

$$d\mathbf{H}/ds = \mathbf{\Omega} \times \mathbf{H}, \quad d\mathbf{L}/ds = \mathbf{\Omega} \times \mathbf{L}, \quad d\mathbf{U}/ds = \mathbf{\Omega} \times \mathbf{U} . \qquad (1\text{-}3)$$

Since all vectors **H**(s) have unit length and are tangent to the axis at points **P**(s), $s \in [0,L]$, the axis shape is given by the integral:

$$\mathbf{P}(s) = \mathbf{P}(0) + \int_0^s \mathbf{H}(s)ds . \qquad (4)$$

To calculate the rates of rotation $\boldsymbol{\Omega}$ that determine the shape of the rod at a static equilibrium, we compare the internal moments ${}^{I}\mathbf{M}$, resulting from the reaction of the material to deformation, and external moments $\mathbf{M}$ acting on all points $\mathbf{P}$ of the rod. Let us consider the internal moments first. The infinitesimal rotations are vectors, thus the rotation rates are vectors as well, and can be decomposed along the axes **HLU**:

$$\Omega_H = \boldsymbol{\Omega}\cdot\mathbf{H}, \quad \Omega_L = \boldsymbol{\Omega}\cdot\mathbf{L}, \quad \Omega_U = \boldsymbol{\Omega}\cdot\mathbf{U}, \tag{5}$$

$$\underline{\Omega}_H = \underline{\boldsymbol{\Omega}}\cdot\mathbf{H}, \quad \underline{\Omega}_L = \underline{\boldsymbol{\Omega}}\cdot\mathbf{L}, \quad \underline{\Omega}_U = \underline{\boldsymbol{\Omega}}\cdot\mathbf{U}\ . \tag{6}$$

Since $\mathbf{L}$ and $\mathbf{U}$ are the principal axes of the cross-section of the rod, its elastic properties at $\mathbf{P}$ are captured by the torsional rigidity C_H and the flexural rigidities in the planes **HU** and **HL**, denoted C_L and C_U, respectively. The moment ${}^{I}\mathbf{M}$ due to the local deformation of the rod at $\mathbf{P}$ is equal to

$${}^{I}\mathbf{M} = {}^{I}M_H\,\mathbf{H} + {}^{I}M_L\,\mathbf{L} + {}^{I}M_U\,\mathbf{U}\ , \tag{7}$$

where:

$${}^{I}M_H = C_H\,(\underline{\Omega}_H - \Omega_H), \quad {}^{I}M_L = C_L\,(\underline{\Omega}_L - \Omega_L), \quad {}^{I}M_U = C_U\,(\underline{\Omega}_U - \Omega_U)\ . \tag{8-10}$$

By substituting equations (5-6) into (8-10) and then into (7), we obtain:

$${}^{I}\mathbf{M} = C_H\,((\underline{\boldsymbol{\Omega}} - \boldsymbol{\Omega})\cdot\mathbf{H})\mathbf{H} + C_L\,((\underline{\boldsymbol{\Omega}} - \boldsymbol{\Omega})\cdot\mathbf{L})\mathbf{L} + C_U\,((\underline{\boldsymbol{\Omega}} - \boldsymbol{\Omega})\cdot\mathbf{U})\mathbf{U}, \tag{11}$$

or, in dyadic notation,

$${}^{I}\mathbf{M} = (\underline{\boldsymbol{\Omega}} - \boldsymbol{\Omega})\cdot\mathbb{S}, \text{ where } \mathbb{S} = C_H\,\mathbf{HH} + C_L\,\mathbf{LL} + C_U\,\mathbf{UU}\ . \tag{12-13}$$

Let us denote by $\mathbf{K}$ the external force per unit length, acting on the rod at $\mathbf{P}$. The accumulated force $\mathbf{F}$ and the moment $\mathbf{M}$ caused by the "overhanging" rod segment *[s,L]* acting on the rod at $\mathbf{P}$ satisfy the equations [8]:

$$d\mathbf{F}/ds = -\mathbf{K} \quad \text{and} \quad d\mathbf{M}/ds = \mathbf{F}\times\mathbf{H}\ . \tag{14-15}$$

At a static equilibrium, the equation

$$\mathbf{M} + {}^{I}\mathbf{M} = \mathbf{0} \tag{16}$$

must be satisfied at each point $\mathbf{P}$ of the rod.

Differential equations (1-3,14-15), complemented with the algebraic equations (12,16), represent a two-point boundary problem with the unknown vectors $\mathbf{H}$, $\mathbf{L}$, $\mathbf{U}$, $\boldsymbol{\Omega}$, ${}^{I}\mathbf{M}$, $\mathbf{F}$, and $\mathbf{M}$. We solve these equations numerically using a simple relaxation technique. To this end, we divide the rod into short segments of length Δs_i, where index i ranges from 0 at the proximal (fixed) end of the rod to n at the distal (free) end, and apply the following algorithm:

Input: Vectors $\mathbf{H}(0)$, $\mathbf{L}(0)$, and $\mathbf{U}(0)$ that define the orientation of the **HLU** frame at the proximal end of the rod, force $\mathbf{F}(\mathrm{L}) = \mathbf{0}$ and moment $\mathbf{M}(\mathrm{L}) = \mathbf{0}$ at the free end, external force densities $\mathbf{K}_i$, rotation rates $\underline{\boldsymbol{\Omega}}_i$ that define the shape of the rod in the unloaded state, and the initial values of the rotation rates $\boldsymbol{\Omega}_i$ (for example, all equal to $\underline{\boldsymbol{\Omega}}_i$).

Output: The shape of the rod at a static equilibrium.

Step 1. Compute the orientation of the frame **HLU** at each point of the rod using a discretized version of equations (1-3). Since the orientation of the

frame at the proximal end of the rod is known, computation proceeds *outwards* from the proximal to the distal end, according to the formula:

$$\mathbf{H}_{i+1}^{\,t} = \mathbf{H}_i^t + \boldsymbol{\Omega}_i^t \times \mathbf{H}_i^t \, \Delta s_i , \qquad i = 0,\ldots,n\text{-}1, \qquad (17)$$

and analogous formulae for $\mathbf{L}_{i+1}^{\,t}$ and $\mathbf{U}_{i+1}^{\,t}$.

Step 2. Compute the distribution of the external forces and moments along the rod using a discretized version of equations (14-15). Since the boundary values at the distal end of the rod are known, computation proceeds *inwards* from the distal to the proximal end according to the formulae:

$$\mathbf{F}_{i-1}^{\,t} = \mathbf{F}_i^t + \mathbf{K}_i \, \Delta s_i , \qquad i = n,\ldots,1, \qquad (18)$$

$$\mathbf{M}_{i-1}^{\,t} = \mathbf{M}_i^t + \mathbf{H}_i^t \times \mathbf{F}_i^t \, \Delta s_i . \qquad i = n,\ldots,1. \qquad (19)$$

Step 3. Compute the unbalanced moments at nodes i between segments Δs_i and Δs_{i+1} using a combination of formulae (12,13,16):

$$\mathbf{E}_i^t = \mathbf{M}_i^t + {}^I\mathbf{M}_i^t = \mathbf{M}_i^t + (\underline{\boldsymbol{\Omega}}_i - \boldsymbol{\Omega}_i^t) \cdot (C_H \, \mathbf{HH} + C_L \, \mathbf{LL} + C_U \, \mathbf{UU}), \qquad (20)$$

then adjust the rotation rates $\boldsymbol{\Omega}_i$ proportionally to these unbalanced moments:

$$\boldsymbol{\Omega}_i^{t+1} = \boldsymbol{\Omega}_i^t + k \, \mathbf{E}_i^t , \qquad i = 0,\ldots,n\text{-}1. \qquad (21)$$

The parameter k is an empirically chosen constant that controls the speed of convergence to the solution.

Step 4. Repeat steps 1-4 until the magnitude of all unbalanced moments $|\mathbf{E}_i|$ decreases below a threshold value, then compute the shape of the rod using a discretized counterpart of equation (4):

$$\mathbf{P}_{i+1} = \mathbf{P}_i + \mathbf{H}_i \, \Delta s_i . \qquad \square \qquad (22)$$

The two-way information flow inherent in this algorithm has been described in the context of the analysis of chainlike robotic manipulators by Craig [1]. It is the cornerstone of the integration of mechanical phenomena into developmental models of plant architecture.

Model expression using L-systems

We assume that the reader is familiar with the formalism of L-systems and its application to the modeling of plant architecture, as described, for example, in [14,15]. The concept of computing numerical solutions of differential equations using L-systems is further discussed in [6].

An L-system captures the development of a plant using *rewriting rules* or *productions*. For example, the rule $\mathbf{A} \rightarrow \mathbf{IA}$ may be used to specify that at given time intervals an apex **A** will produce a internode **I** and recreate itself at the distal end of this internode. A repetitive application of this rule yields an axis composed of a sequence of internodes:

$$\mathbf{A} \Rightarrow \mathbf{IA} \Rightarrow \mathbf{IIA} \Rightarrow \mathbf{IIIA} \Rightarrow \mathbf{IIIIA} \Rightarrow \ldots \qquad (23)$$

Plant modules, such as the apex and the internodes, can be characterized using numerical parameters. Let us consider a simple example of a develop-

ing axis in which all internodes have the same length Δs and are subject to the same force **K** per unit length. We assume that the vectors **H**, **U** and **K** are coplanar, thus the axis will bend in the plane **HU** perpendicular to **L**. We also assume that the rigidity C_L is constant. An internode **I** is then completely specified by vector **H**, rotation rate $\boldsymbol{\Omega}$, external force **F** and external bending moment **M**. If all these parameters are assigned the initial value of **0**, the production $\mathbf{A} \rightarrow \mathbf{IA}$ will become $\mathbf{A} \rightarrow \mathbf{I}(\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0})\mathbf{A}$. The algorithm for computing the shape of the rod in equilibrium can be then expressed as follows:

Step 1. The outward propagation of orientations in a sequence of internodes (equation 17) is implemented by production

$$\mathbf{I}(\mathbf{H}_l, \boldsymbol{\Omega}_l, \mathbf{F}_l, \mathbf{M}_l) < \mathbf{I}(\mathbf{H}, \boldsymbol{\Omega}, \mathbf{F}, \mathbf{M}) \rightarrow \mathbf{I}(\mathbf{H}_l + \boldsymbol{\Omega}_l \times \mathbf{H}_l \Delta s, \boldsymbol{\Omega}, \mathbf{F}, \mathbf{M}) ,$$

where symbol < separates the left context from the strict predecessor of the production [14,15]. This production states that the header vector **H** in the module $\mathbf{I}(\mathbf{H}, \boldsymbol{\Omega}, \mathbf{F}, \mathbf{M})$ will acquire a new direction, calculated as a function of the header vector $\mathbf{H}_l$ and the rotation rate $\boldsymbol{\Omega}_l$ in the previous internode.

Step 2. The inward propagation of external forces and moments (equations 18 and 19) is expressed by production

$$\mathbf{I}(\mathbf{H}, \boldsymbol{\Omega}, \mathbf{F}, \mathbf{M}) > \mathbf{I}(\mathbf{H}_r, \boldsymbol{\Omega}_r, \mathbf{F}_r, \mathbf{M}_r) \rightarrow \mathbf{I}(\mathbf{H}, \boldsymbol{\Omega}, \mathbf{F}_r + \mathbf{K}_r \Delta s, \mathbf{M}_r + \mathbf{H}_r \times \mathbf{F}_r \Delta s),$$

where symbol > separates the strict predecessor from the right context.

Step 3. The remaining computations (equations 20 and 21) are performed under the simplifying assumption of the planar deformation of the rod. The unbalanced moment is then reduced to $\mathbf{M} + C_L \boldsymbol{\Omega}$, and the updated rotation rate $\boldsymbol{\Omega}$ is captured by production:

$$\mathbf{I}(\mathbf{H}, \boldsymbol{\Omega}, \mathbf{F}, \mathbf{M}) \rightarrow \mathbf{I}(\mathbf{H}, \boldsymbol{\Omega} + \mathrm{k}\,(\mathbf{M} + C_L \boldsymbol{\Omega}), \mathbf{F}, \mathbf{M}) .$$

In the complete L-system, these productions are guarded by conditions that ensure proper sequencing of the production applications (*Step 4*) and schedule the addition of new segments by the apex. Details are given in [7].

In general, the assignment of parameters to the internodes provides a mechanism for automatically increasing the number of variables that describe the plant as it grows. Variables in the neighboring modules are accessed using context-sensitive productions. Since L-systems can capture the development of branching structures and the information flow between their modules, the described technique extends from individual axes to entire plants.

Secondary growth, tropisms, and collisions

Radial (secondary) growth is simulated according to the pipe model [17], which postulates that the vascular strands originating in a newly added segment contribute to the girth of previous segments. In other words, the addition of a distal internode of cross-section A causes the addition of external layers of the same area to all preceding segments. Together the primary and secondary growth modify [4,16]:

- The linear density of the external forces **K**. For example, if the external forces are due exclusively to the axis weight, then $\mathbf{K} = A\rho\mathbf{g}$, where A is the area of the cross-section of the stem at a given point P, ρ is the average density of the stem material at this cross-section, and **g** is the acceleration of gravity.
- The torsional and flexural rigidities $C_H = GJ$, $C_L = EI_L$ and $C_U = EI_U$, where G is the shear modulus of the stem material, E is its Young's modulus, J is the torsional constant, and I_L and I_U are the second moments of area of the stem's cross-section. The values of constants J, I_L and I_U for different cross-sectional geometries are listed in [12].
- The curvature and twist of the branch axis at rest $\underline{\mathbf{\Omega}}$. Assuming that the principal axes of the cross-section of a new annual layer are aligned with the principal axes of the previous cross-section, each component of the rate of rotation at rest $\underline{\mathbf{\Omega}}'$ is calculated as a weighted average of the previous rotation at rest $\underline{\mathbf{\Omega}}$ and the current rotation vector $\mathbf{\Omega}$, *e.g.*:

$$\underline{\Omega}'_L = (C_L\underline{\Omega}_L + C'_L\Omega_L) / (C_L + C'_L). \quad (24)$$

  The constants C_L and C'_L denote the rigidities of the previous branch axis and of the newly added layer, respectively.

Equation (24) is based on the observation that a new layer added by secondary growth is "molded" on the existing branch segment, and thus may have different rest curvature and twist than previous layers [4,16]. The new rate of rotation $\underline{\mathbf{\Omega}}'$ (curvature and twist) of an axis segment at rest represents the auto-stress equilibrium between the new layer and the existing core within the segment's cross-section. Thus, the rest shape of an axis after a step of secondary growth partially memorizes the actual shape of this axis under load. A derivation of equation (24) is included in [7]. It is an alternative but equivalent formulation to that proposed in [4].

The secondary growth is incorporated into the model as follows. After a step of longitudinal growth, the girth of all internodes is recalculated according to the pipe model. The propagation of information from the distal to the proximal end, postulated by the pipe model, is implemented by right-context-sensitive productions. The rotation rates at rest and the rigidities of each internode are computed using the formulae described above and incorporated into an L-system production. The new mechanical equilibrium is then determined as discussed in the preceding sections for an axis without secondary growth. This cycle is repeated for each new internode produced by the apex.

Further extensions of the model include tropisms and collisions. Tropisms are simulated by rotating the frame of the newly inserted segment so that its tangent vector **H** becomes more closely aligned with the direction of a predefined tropism vector **T**. Contacts between structural elements (for example, grapes in a bunch) and between the structure and its environment (*e.g.*, branches partially laying on the ground) are simulated under the assumption that the colliding elements are elastic, and act on each other with forces proportional to the depth of penetration (penalty method [13].) This technique makes it possible to approximate the effect of contact on the deformation of the axes, although it is not precise enough to predict the local shape of the contact zone in the organs resting on each other.

Results

The described biomechanical model was applied to simulate the development and capture the structure of several plants. The simulations were carried out using the plant simulation software *cpfg* with open L-system support [9]. The open L-system extension was used to exchange forces between parts in contact. A complete implementation of the L-system models is presented in [7].

Figure 2 compares the S-shaped branches of a tree with the results of a simulation. The branches in the model bend downward due to their weight, but the branch tips arch upward due to a vertically oriented tropism vector. Figure 3 shows the results of simulating a hanging plant using the same model with different parameters. In the model of *Spiraea* sp. (Figure 4), twigs arch downward due to gravity, whereas the flower-bearing shoots stand upright due to a strong orthogravitropism. Figure 5 illustrates the effect of contact between fruits on the shape of fruit stems. The stems were assumed to elongate and increase in diameter uniformly throughout their length, with no new segments added during the simulation. Two applications of this model are shown in Figure 6. Figure 7 illustrates schematically the effect of contact with the ground on the shape of a growing axis affected by tropism. The same model without tropism was formally applied to recreate a cycad (Figure 8), with the leaves prevented from penetrating the ground plane by the collision-detecting mechanism.

Figure 2: A photograph and a model of S-shaped tree branches. The branches bend down due to gravity, but arch upward at the distal ends due to a tropism.

Figure 3: A photograph and two views of a model of a hanging plant. Branches hang down due to gravity, but are also influenced by an upward tropism.

Figure 4: A photograph, a model and a zoom into the modeled flower-bearing shoots of a *Spiraea* shrub. The twigs arch downward due to gravity, the flower-bearing shoots stand upright due to simulated orthogravitropism.

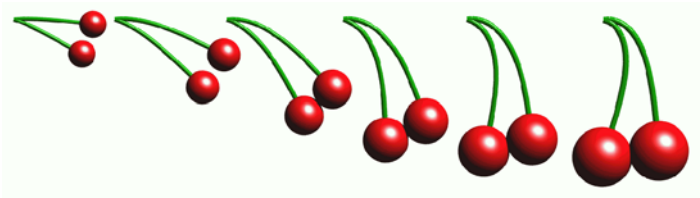


Figure 5: Growing cherries. The stems bend down as the fruits become heavier.

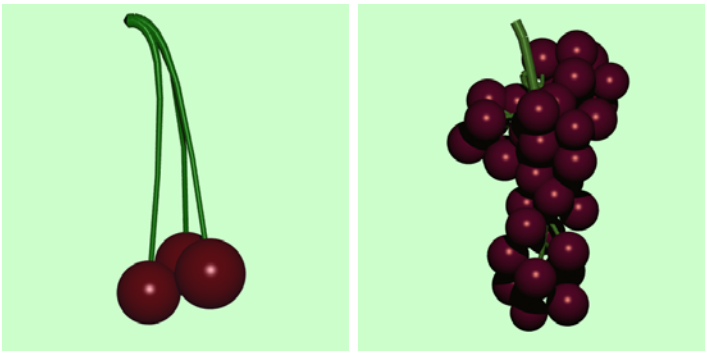


Figure 6: Models of cherries and grapes. The individual fruits rest against each other.

Figure 7: A growing orthotropic axis collides with the ground.

Figure 8: A photograph and two views of a model of a cycad. The lower leaves lie on the ground plane.

In plants and organs without organized secondary growth (*e.g.,* Figures 5 and 8), biomechanical effects of growth are manifested by diffuse primary radial expansion and secondary cell wall deposition, rather than the addition of external rings of cells. Nevertheless, processes of shape memorization have been reported in herbs [10] and the biomechanical principle involved is probably similar [11]. Therefore, we have applied equation (24) to qualitatively capture the shape memorization mechanism even in the absence of detailed information concerning the distribution of growth rates within the cross sections.

Conclusions

The described model makes it possible to visually capture the shape of branches resulting from the combined effect of weight, tropisms, and contact of organs with each other and with obstacles in the environment. The incorporation of biomechanics into L-systems makes it possible to explore different branching architectures relatively easily. Prospective extensions and applications of the model include: (a) incorporation of a mechanistic model of tropisms that associates bending of branch axes to differential growth; (b) simulation of the biological regulation of reaction wood formation, and its mechanical effects; and (c) testing of biological hypotheses relating plant architecture to biomechanics, for instance the impact of stresses in the mother branch axis on the formation of lateral branches.

Acknowledgments

The support for this work by a research grant, a postgraduate scholarship, and an equipment grant from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

References

[1] J. J. Craig (1989): *Introduction to robotics: mechanics and control.* Second edition. Addison-Wesley, Reading, 1989.

[2] J. Digby and R. D. Firn (1995): The gravitropic set-point angle (GSA): the identification of an important developmentally controlled variable governing plant architecture. *Plant, Cell and Environment*, 18: 1434-1440.

[3] Th. Fourcaud (1997): Relations entre croissance et biomécanique de l'arbre. In: J. Bouchon, Ph. de Reffye and D. Barthélémy (Eds.): *Modélisation et simulation de l'architecture des végétaux*. INRA Editions, Paris, pp. 350-382.

[4] M. Fournier, H. Bailleres, and B. Chanson (1994): Tree biomechanics: growth, cumulative prestresses, and reorientations. *Biomimetics*, 2 (3): 229-251.

[5] J. Gril, F. Blaise and M. Fournier (1992) Introduction de concepts mécaniques dans un logiciel de croissance des plantes. In: B. Thibaut (Ed.): *Architecture, Structure, Mécanique de l'Arbre IV*. LMGC, Montpellier, pp. 171-185.

[6] M. Hammel and P. Prusinkiewicz (1996): Visualization of developmental processes by extrusion in space-time. Proceedings of Graphics Interface '96, pp. 246-258.

[7] C. Jirasek (2000): A biomechanical model of branch shape in plants expressed using L-systems. M.Sc. Thesis, Department of Computer Science, University of Calgary.

[8] L. Landau and E. Lifshitz (1986): *Theory of elasticity*. Third edition. Butterworth Hinemann, Oxford.

[9] R. Mech and P. Prusinkiewicz (1996): Visual models of plants interacting with their environment. Proceedings of SIGGRAPH '96, pp. 397-410.

[10] B.Moulia., M. Fournier and D. Guitard (1994): Mechanics and form of the maize leaf : in vivo qualification of the flexural behaviour. *J. Mater. Sci*., 29 : 2359-2366.

[11] B.Moulia (1993) *Etude mécanique du port foliaire du maïs (Zea mays L.)*. Thèse de l'Université de Bordeaux I (UFR de Physique). 123pp.

[12] K. Niklas (1992): *Plant biomechanics*. The University of Chicago Press, Chicago.

[13] J. Platt and A. Barr (1988): Constraint methods for flexible models. *Computer Graphics*, 22 (4): 279-288.

[14] P. Prusinkiewicz, M. Hammel, J. Hanan and R. Mech (1997): Visual models of plant development. In: G. Rozenberg and A. Salomaa (Eds.): *Handbook of formal languages*, Vol. III, Springer, Berlin, pp. 535-597.

[15] P. Prusinkiewicz and A. Lindenmayer (1990): *The algorithmic beauty of plants*. Springer, New York.

[16] B. Schaffer (1990) Forme d'équilibre d'une branche d'arbre. *CR Acad. Sci. Paris,* 311 (2): 37-43.

[17] K. Shinozaki, K. Yoda, K. Hozumi and T. Kira (1964): A quantitative analysis of plant form - the pipe model theory. I. Basic analyses. *Japanese Journal of Ecology*, 14 (3): 97-104.

L-system description of subdivision curves*

Przemyslaw Prusinkiewicz, Faramarz Samavati
Colin Smith and Radoslaw Karwowski

Department of Computer Science
The University of Calgary

pwp|samavati|smithco|radekk@cpsc.ucalgary.ca

Abstract

In recent years, subdivision has emerged as a major geometric modeling technique. Algorithms for generating subdivision curves are often specified in terms of iterated matrix multiplication. Each multiplication maps a globally indexed sequence of points that represents a coarser approximation of the curve onto a longer sequence that represents a finer approximation. Unfortunately, an infinite set of matrices is needed to specify these mappings for sequences of points of arbitrary length. Thus, matrix algebra is not well attuned to the dynamic nature of subdivision. In addition, matrix notation and the use of indices obscure the local and stationary character of typical subdivision rules.

We introduce parametric context-sensitive L-systems with affine geometry interpretation as an alternative technique for specifying and generating subdivision curves. This technique is illustrated using Chaikin, cubic B-spline, and Dyn-Levin-Gregory (4-point) subdivision schemes as examples. L-systems formalize subdivision algorithms in an intuitive, concise, index-free manner, reflecting the parallel and local character of these algorithms. Furthermore, L-system specification of subdivision algorithms directly leads to their computer implementation.

1 Introduction

The definition and generation of smooth curves and surfaces specified by a small set of control points is a fundamental problem of geometric modeling. One class of solutions is based on the concept of subdivision: an iterative replacement of coarser representations of a curve or surface by finer representations. The first subdivision algorithm for curves was described almost thirty years ago [5], and algorithms for surfaces soon followed [4, 11]. Nevertheless, subdivision was not recognized as a practical modeling technique until the late 1990's, when is was successfully applied to character animation [10]. This

*To appear in the *International Journal of Shape Modeling*. Used by permission from World Scientific.

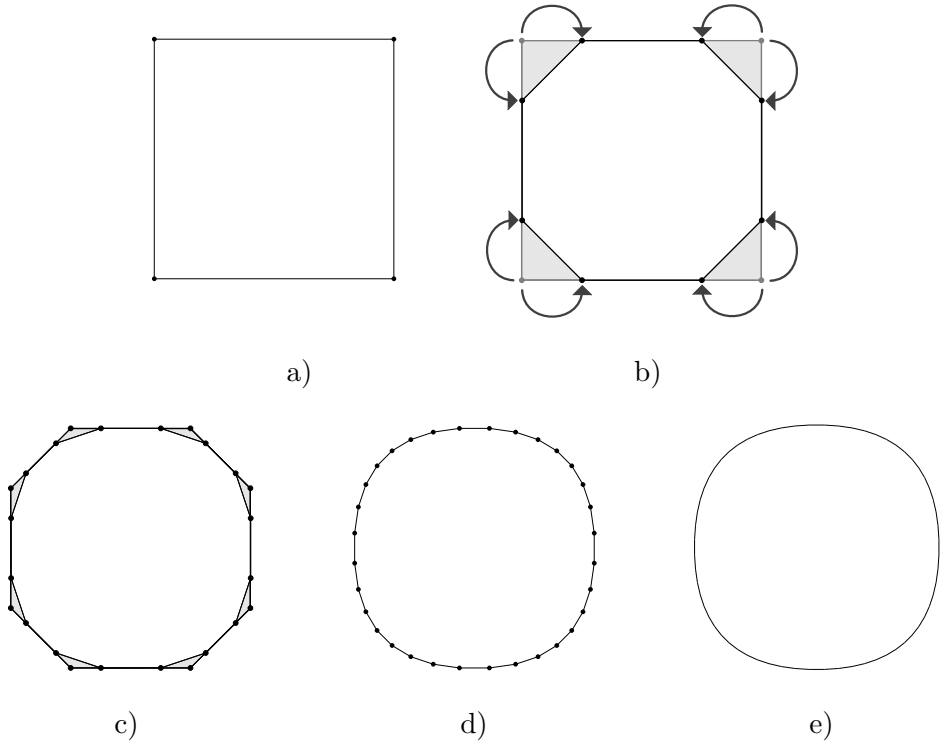


Figure 1: Curve generation using Chaikin's algorithm. a) A sample user-specified control polygon. b) The replacement of old vertices by pairs of new vertices in the first step of the algorithm. Shaded areas represent the cut-off corners. c–e) Illustration of the subsequent three subdivision steps.

development coincided with the explosion of research interest in subdivision curves and surfaces, which continues until now.

One appealing aspect of subdivision is that, at the intuitive level, it is easy to describe and understand. Unfortunately, this simplicity is not reflected in the index- and matrix-based notation often used to formalize subdivision algorithms. In this paper we propose context-sensitive parametric L-systems [22, 30] as an alternative formalism, rooted in formal language theory. L-systems make it possible to effectively specify growing sequences of symbols (words) without the use of indices. We extend this capability to polygonal approximations of subdivision curves. The L-system notation captures the local nature of subdivision algorithms in a formal yet intuitive manner, and leads directly to a computer implementation of these algorithms. This, in turn, is useful in practical applications, such as experimentation with different subdivision schemes, and expository presentation of subdivision algorithms.

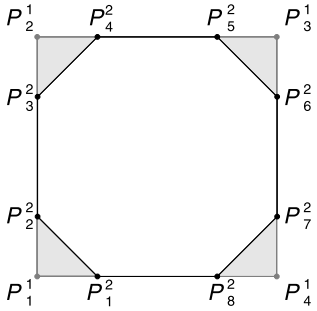


Figure 2: Example of the labeling of points in the first step of Chaikin's algorithm

2 Background

Let us consider the original Chaikin algorithm [5] to review the main concept of subdivision and its usual formalization. For simplicity, we will apply it to generate a closed curve, thus avoiding special-case rules that would be needed near the endpoints of an open curve. The initial approximation of the curve under construction is specified by a (circular) list of user-defined control points. Figure 1a shows a sample arrangement of these points, connected to form a polygon. The next approximation is obtained by cutting the corners of this polygon (Figure 1b). Each "old" vertex is replaced with a pair of "new" vertices, where each new vertex is situated one quarter of the distance from its parent point to one of its neighbors (Figure 1b). The subsequent approximations of the curve are obtained by iterating the same corner-cutting scheme (Figure 1c–e).

In the standard formalization of the subdivision process, points are globally enumerated and assigned unique labels. A possible labeling scheme is shown in Figure 2. The superscript k indicates the iteration step at which point P_i^k has been created. The subscript i is the ordering number of this point within the sequence of points created in the same step.

The positions of new points are expressed as affine combinations of the positions of old points. An affine combination of n points $P_1, P_2, \ldots, P_n$ is an expression of the form

$$\alpha_1 P_1 + \alpha_2 P_2 + \cdots + \alpha_n P_n, \tag{1}$$

where the scalar coefficients α_i add up to 1:

$$\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1. \tag{2}$$

The meaning of the affine combination (1) is derived from its transformation to the form

$$P = P_1 + \alpha_2(P_2 - P_1) + \cdots + \alpha_n(P_n - P_1), \tag{3}$$

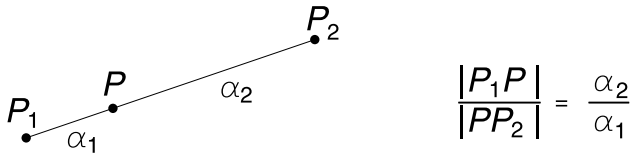


Figure 3: Definition of point P as an affine combination of points P_1 and P_2.

which is a well-defined expression of vector algebra. Specifically, for two points we obtain:

$$P = \alpha_1 P_1 + \alpha_2 P_2 = P_1 + \alpha_2(P_2 - P_1) = P_2 + \alpha_1(P_1 - P_2). \tag{4}$$

Thus, point P divides line $\overline{P_1P_2}$ in the ratio of $\alpha_2 : \alpha_1$ (Figure 3). Affine geometry and its applications to computer graphics have been described in detail by DeRose [8, 9]; for further insights see [20, 18].

Returning to Figure 2, the new point positions are related to the old point positions by the affine combinations:

$$P_1^2 = \frac{3}{4}P_1^1 + \frac{1}{4}P_4^1, \tag{5}$$

$$P_2^2 = \frac{3}{4}P_1^1 + \frac{1}{4}P_2^1, \tag{6}$$

and so on for the remaining points. In matrix notation,

$$\begin{bmatrix} P_1^2 \\ P_2^2 \\ P_3^2 \\ P_4^2 \\ P_5^2 \\ P_6^2 \\ P_7^2 \\ P_8^2 \end{bmatrix} = \begin{bmatrix} \frac{3}{4} & 0 & 0 & \frac{1}{4} \\ \frac{3}{4} & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} \\ \frac{1}{4} & 0 & 0 & \frac{3}{4} \end{bmatrix} \begin{bmatrix} P_1^1 \\ P_2^1 \\ P_3^1 \\ P_4^1 \end{bmatrix}. \tag{7}$$

This equation is generalized to arbitrary control polygons and arbitrary derivation steps by writing:

$$\begin{bmatrix} P_1^{k+1} \\ P_2^{k+1} \\ P_3^{k+1} \\ P_4^{k+1} \\ P_5^{k+1} \\ P_6^{k+1} \\ \vdots \\ P_{2n-2}^{k+1} \\ P_{2n-1}^{k+1} \\ P_{2n}^{k+1} \end{bmatrix} = \begin{bmatrix} \frac{3}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{1}{4} \\ \frac{3}{4} & \frac{1}{4} & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{3}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{4} & \frac{3}{4} \\ \frac{1}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{3}{4} \end{bmatrix} \cdot \begin{bmatrix} P_1^k \\ P_2^k \\ P_3^k \\ P_4^k \\ \vdots \\ P_{n-1}^k \\ P_n^k \end{bmatrix} \tag{8}$$

Subdivision schemes other than Chaikin's can be specified in a similar way, using different subdivision matrices [2, 33, 36, 37].

Related to the use of matrices is the use of indices to identify and order the points. Unfortunately, the index notation is not well attuned to the needs of subdivision. Due to the local character of subdivision, the creation of a pair of new points is based on the information about their parent old point and its neighbors. Indexing makes it possible to access this information only in a circular way, by first globally assigning consecutive numbers to all points, then referring to the neighbors of point P_i^k using index arithmetic: $i-1$ and $i+1$. This is more complicated than the verbal description, in which we would use terms such as "previous" and "next" (or "left" and "right") to refer to the neighbors of a given point. At the same time, the index notation is too powerful: by providing a unique label to each point it makes it possible to access points at random, in violation of the algorithm's locality. This is true in both the spatial and temporal domains: in the latter case, the use of indices makes it potentially possible to refer to points from arbitrary iteration steps, whereas only the information from the previous step is available and needed.

3 From stencils and masks to productions

One alternative to the index-based notation is the representation of subdivision rules using *stencils*. Sabin [32] defines them as follows:

> **Stencil.** The weights due to various old vertices in computing a given new one. Also the pattern of relative positions of the old vertices around the new one.

Stencils (and the related notion of *masks*) are usually represented as graphs that depict short subsequences of old and new points. These points are connected by arrows labeled

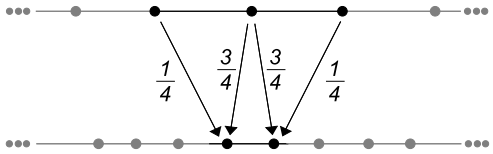


Figure 4: A stencil for Chaikin subdivision algorithm

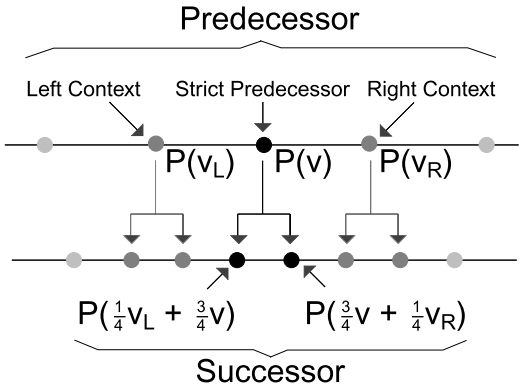


Figure 5: Chaikin's subdivision as a production

by coefficients α in the affine geometry combinations (Equation 1) that take the old points to the new ones. For instance, a stencil for the Chaikin subdivision algorithm is shown in Figure 4. For other examples see [14, 38].

Stencils provide an intuitive, index-free representation of subdivision rules. Unfortunately, this is not a precise representation. For example, Figure 4 shows that three old points are involved in creating a pair of new points. It does not explicitly specify, however, whether the same old points may also be involved in the production of other new points. Looking at the same problem from a different perspective, it is not clear to what extent the stencils may overlap when applied to various subsequences of old points.

We address this imprecision by recasting the notion of a stencil into the framework of formal language theory (Figure 5). A sequence of points is viewed as a word over some alphabet. A circular sequence of points approximating a closed curve is represented by a circular word [31, 35]. The stencil is a grammar production, with the predecessor representing a finite subsequence of old points, and the successor representing a subsequence of new points. The predecessor is partitioned into the strict predecessor, left context, and right context. The strict predecessor represents the old point that is rewritten or "consumed" by the production application, that is, cannot be used anymore. In Chaikin's construction, it is the vertex of a corner being cut off. The

context consists of the neighbors of the strict predecessor that provide the additional information needed to specify the successor points. When rewriting a sequence of old points, production predecessors may overlap, as long as no point is used more than once as a strict predecessor.

The Chaikin construction requires that each old point be replaced by two new ones in every iteration of the algorithm. This corresponds to the notion of parallel rewriting as defined for L-systems [22, 23], as opposed to the sequential rewriting defined for Chomsky grammars [7].

4 L-systems

L-systems were originally introduced as a rewriting mechanism acting on words over a finite alphabet [22, 23]. Soon afterward, however, they were extended to strings of symbols with numerical attributes [1, 24]. This concept was formalized as parametrized [6] and parametric [19, 30] L-systems. Here we use an extension of the latter formalism. For its more detailed presentation see [26, 30].

Parametric L-systems operate on strings of modules. A module is a letter from a finite alphabet V with optional numerical parameters. For example, the string

$$A(1.5)B(2.0, 3.0)A(4.5) \tag{9}$$

is a parametric word over the alphabet $V = \{A, B\}$.

Starting with an explicitly defined initial word, or axiom, an L-system generates a developmental sequence of words using a finite set of productions that operate on limited-length subwords. The actual parameters in each word correspond to the formal parameters in the productions. Arithmetic expressions in the successor of a production determine new parameter values. In the case of context-sensitive productions, the left and right contexts are separated from the strict predecessor by the metasymbols (*i.e.*, symbols that do not represent modules) $<$ and $>$, respectively.

A developmental sequence of words results from a sequence of derivation steps. In each step, productions are applied in parallel to all modules of the predecessor word, so that each module is the strict predecessor of some production. For example, by applying the production set

$$A(x) \rightarrow A(2x + 1) \tag{10}$$

$$A(w) < B(x, y) > A(x) \rightarrow A(w + x)A(y + x) \tag{11}$$

to the parametric word (9), we obtain after one derivation step:

$$A(4.0)A(3.5)A(7.5)A(10). \tag{12}$$

Recent extensions of parametric L-systems [13, 21] make it possible to use not only numbers, but also compound data structures as parameters. We use this feature

to represent points as vectors of coordinates, and we overload standard arithmetic operators to specify affine combinations of points. With this convention, the L-system production that specifies Chaikin's subdivision becomes:

$$P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\frac{1}{4} \cdot \mathbf{v}_l + \frac{3}{4} \cdot \mathbf{v})P(\frac{3}{4} \cdot \mathbf{v} + \frac{1}{4} \cdot \mathbf{v}_r) \tag{13}$$

This mathematical notation is closely reflected in the programming language `L+C`, which combines features of L-systems and the `C++` programming language [21]. `L+C` programs constitute an input to the graphical modeling program `lpfg` [21], which we have used to implement the algorithms presented in this paper. For example, the following `L+C` program generates the picture (curve with dots) shown in Figure 1d:

```
1   #include <lpfgall.h>
2   V2f v1(0, 0), v2(0, 1), v3(1, 1), v4(1, 0);
3   module P(V2f);
4
5   ring L-system: 1;
6   derivation length: 3;
7
8   Axiom: P(v1) P(v2) P(v3) P(v4) ;
9   P(vl) < P(v) > P(vr) :
10        { produce P(0.25*vl+0.75*v) P(0.75*v+0.25*vr); }
11
12  interpretation:
13  P(v) :  { produce LineTo2f(v) Circle(0.01) ; }
```

Line 1 is a reference to the file `lpfgall.h` that contains predefined constants and structure declarations. Specifically, our program makes use of two-dimensional vectors `V2f`. Line 2 defines four points that will be used in line 8 as the vertices of the control polygon. Line 3 declares module type `P` as being associated with one parameter — a vector of type `V2f`. Line 5 specifies that the L-system will operate on circular words, and line 6 specifies the required derivation length. Lines 9 and 10 are the essence of this program and contain the production responsible for the Chaikin subdivision. Finally, Line 13, preceded by the keyword in line 12, defines the homomorphism that will be applied at the end of the derivation (*c.f.* [28]). According to it, each point is represented as a small circle, connected by a line to its predecessor.

For compactness, in the following sections we will mainly use the mathematical notation exemplified by Equation 13, rather than complete program listings.

5 Inferring L-systems from subdivision matrices

Since subdivision curves are often defined using matrix notation [2, 33, 36, 37], the inference of L-systems from the subdivision matrices is an important practical problem. Unfortunately, it cannot entirely be resolved by algorithmic means, because dots in the general subdivision matrices, *e.g.* (8), require an interpretation. Furthermore, as we are going to see, many equivalent L-systems can be inferred from the same matrix, thus the inference process involves an element of decision.

As described in Section 2, the subdivision matrix globally maps an old sequence of points onto a new sequence. In contrast, an L-system production replaces an individual old point by a subsequence of new points. We must therefore partition the sequence of new points into subsequences, and establish a one-to-one mapping between the old points and these subsequences. For example, a mapping for the Chaikin subdivision (Equation 8) may take point P_i^k to points P_{2i-1}^{k+1} and P_{2i}^{k+1}, where $i = 1, 2, \ldots, n$. An instance of this mapping for $i = 3$ is illustrated below:

$$\begin{bmatrix} P_1^{k+1} \\ P_2^{k+1} \\ P_3^{k+1} \\ P_4^{k+1} \\ P_5^{k+1} \\ P_6^{k+1} \\ \vdots \\ P_{2n-2}^{k+1} \\ P_{2n-1}^{k+1} \\ P_{2n}^{k+1} \end{bmatrix} = \begin{bmatrix} \frac{3}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{1}{4} \\ \frac{3}{4} & \frac{1}{4} & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{3}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{4} & \frac{3}{4} \\ \frac{1}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{3}{4} \end{bmatrix} \begin{bmatrix} P_1^k \\ P_2^k \\ P_3^k \\ P_4^k \\ \vdots \\ P_{n-1}^k \\ P_n^k \end{bmatrix} \tag{14}$$

The encircled points in the column matrices show that old point P_3^k will be replaced by new points P_5^{k+1} and P_6^{k+1}. In the subdivision matrix, the same replacement is indicated by encircling column 3, which represents the contribution of point P_3^k to the matrix multiplication, and rows 5 and 6, which yield points P_5^{k+1} and P_6^{k+1} of the result. The shaded area includes non-zero elements of these rows, and thus identifies the production predecessor and the coefficients of the affine combinations that will yield the successor points. The position of the encircled column with respect to this area partitions the predecessor into left context, strict predecessor, and right context. The replacement of point P_3^k by points P_5^{k+1} and P_6^{k+1} can therefore be written as a production,

$$P_{i-1}^k(\mathbf{v}_l) < P_i^k(\mathbf{v}) > P_{i+1}^k(\mathbf{v}_r) \rightarrow P_{2i-1}^{k+1}(\frac{1}{4} \cdot \mathbf{v}_l + \frac{3}{4} \cdot \mathbf{v}) P_{2i}^{k+1}(\frac{3}{4} \cdot \mathbf{v} + \frac{1}{4} \cdot \mathbf{v}_r), \tag{15}$$

where $i = 3$. The regular form of the subdivision matrix in Equation 14 suggests that production (15) applies for any values $i, k = 1, 2, \ldots$. This observation leads to the general L-system production for Chaikin subdivision in Equation 13.

The decision to replace point P_i^k with the points P_{2i-1}^{k+1} and P_{2i}^{k+1} was an arbitrary one. In general, there is an equivalent one-production L-system that generates the same Chaikin curve (up to a cyclical permutation of points) by taking point P_i^k to a pair of consecutive points $P_{j-1}^{k+1}\ P_j^{k+1}$ $(i = 1, 2, \ldots, n;\ j \equiv 2i + d \bmod 2n)$ for any integer d. The mapping performed by production 13 and illustrated by Equation 14 corresponds to $d = 0$. Two other mappings, corresponding to $d = -1$ and $d = 1$, are indicated below:

$$\begin{bmatrix} \frac{3}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{1}{4} \\ \frac{3}{4} & \frac{1}{4} & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{3}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{4} & \frac{3}{4} \\ \frac{1}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{3}{4} \end{bmatrix}, \qquad \begin{bmatrix} \frac{3}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{1}{4} \\ \frac{3}{4} & \frac{1}{4} & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{3}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{4} & \frac{3}{4} \\ \frac{1}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{3}{4} \end{bmatrix}. \tag{16}$$

The resulting L-system productions are, respectively:

$$P(\mathbf{v}_l) < P(\mathbf{v}) \rightarrow P(\tfrac{3}{4} \cdot \mathbf{v}_l + \tfrac{1}{4} \cdot \mathbf{v}) P(\tfrac{1}{4} \cdot \mathbf{v}_l + \tfrac{3}{4} \cdot \mathbf{v}), \tag{17}$$

$$P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\tfrac{3}{4} \cdot \mathbf{v} + \tfrac{1}{4} \cdot \mathbf{v}_r) P(\tfrac{1}{4} \cdot \mathbf{v} + \tfrac{3}{4} \cdot \mathbf{v}_r). \tag{18}$$

Productions 17 and 18 lack the symmetry of production 13, but are shorter and in this sense simpler than production 13. Other values of constant d yield productions that are also asymmetric, but relatively longer. For example, for $d = 3$ we obtain:

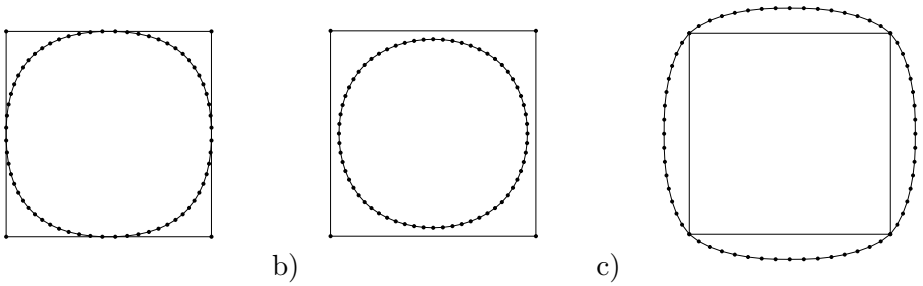
$$\begin{bmatrix} \frac{3}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{1}{4} \\ \frac{3}{4} & \frac{1}{4} & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{3}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{4} & \frac{3}{4} \\ \frac{1}{4} & 0 & 0 & 0 & \cdots & 0 & \frac{3}{4} \end{bmatrix}, \tag{19}$$

$$P(\mathbf{v}) > P(\mathbf{v}_r)P(\mathbf{v}_{rr})P(\mathbf{v}_{rrr}) \rightarrow P(\frac{3}{4}\cdot\mathbf{v}_{rr} + \frac{1}{4}\cdot\mathbf{v}_{rrr})P(\frac{1}{4}\cdot\mathbf{v}_{rr} + \frac{3}{4}\cdot\mathbf{v}_{rrr}). \tag{20}$$

This production, along with other productions obtained for $|d| > 1$, appears to be of limited interest, because new points are increasingly distant from the old points they replace, contrary to the intuition of the Chaikin algorithm.

L-system productions for other subdivision algorithms can be inferred in a similar way. For example, below we present two views of the subdivision matrix for the cubic B-spline subdivision (c.f. [15, 33]):

$$\begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & \cdots & 0 & 0 & \frac{1}{2} \\ \frac{3}{4} & \frac{1}{8} & 0 & 0 & \cdots & 0 & 0 & \frac{1}{8} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & \cdots & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{8} & 0 & \frac{1}{2} & \frac{1}{2} & \cdots & 0 & \frac{1}{8} & \frac{3}{4} \end{bmatrix}, \qquad \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & \cdots & 0 & 0 & \frac{1}{2} \\ \frac{3}{4} & \frac{1}{8} & 0 & 0 & \cdots & 0 & 0 & \frac{1}{8} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & \cdots & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{8} & 0 & \frac{1}{2} & \frac{1}{2} & \cdots & 0 & \frac{1}{8} & \frac{3}{4} \end{bmatrix}. \tag{21}$$



a) b) c)

Figure 6: A comparison of curves generated with (a) Chaikin, (b) cubic B-spline, and (c) Dyn-Levin-Gregory subdivision algorithms, using the same control polygon.

The corresponding L-system productions are, respectively:

$$P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\tfrac{1}{8}\cdot\mathbf{v}_l + \tfrac{3}{4}\cdot\mathbf{v} + \tfrac{1}{8}\cdot\mathbf{v}_r)P(\tfrac{1}{2}\cdot\mathbf{v} + \tfrac{1}{2}\cdot\mathbf{v}_r), \tag{22}$$

$$P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\tfrac{1}{2}\cdot\mathbf{v}_l + \tfrac{1}{2}\cdot\mathbf{v})P(\tfrac{1}{8}\cdot\mathbf{v}_l + \tfrac{3}{4}\cdot\mathbf{v} + \tfrac{1}{8}\cdot\mathbf{v}_r). \tag{23}$$

Figure 6b shows a sample curve generated using either of these productions. Similar to Chaikin subdivision (Figure 6a), the cubic B-spline subdivision yields a curve that approximates the control polygon. In contrast, Dyn-Levin-Gregory 4-point subdivision [12] (see also [33]) generates an interpolating curve (Figure 6c). Its subdivision matrix, complemented with one of the possible mappings of an old point to new points, is given below:

$$\begin{bmatrix}
\frac{9}{16} & -\frac{1}{16} & 0 & 0 & 0 & \cdots & 0 & -\frac{1}{16} & \frac{9}{16} \\
1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
\frac{9}{16} & \frac{9}{16} & -\frac{1}{16} & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
-\frac{1}{16} & \frac{9}{16} & \frac{9}{16} & -\frac{1}{16} & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & -\frac{1}{16} & \frac{9}{16} & \frac{9}{16} & -\frac{1}{16} & \cdots & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & -\frac{1}{16} & \frac{9}{16} & \frac{9}{16} & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \cdots & \frac{9}{16} & \frac{9}{16} & -\frac{1}{16} \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\
-\frac{1}{16} & 0 & 0 & 0 & 0 & \cdots & -\frac{1}{16} & \frac{9}{16} & \frac{9}{16} \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1
\end{bmatrix}. \tag{24}$$

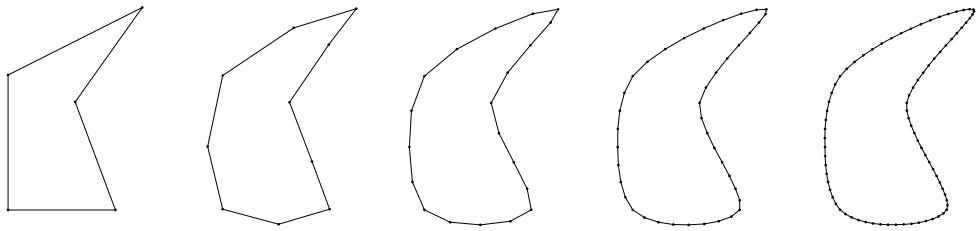


Figure 7: Five steps of a curve generation using the Dyn-Levin-Gregory algorithm

The resulting L-system production is:

$$\begin{aligned} &P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r)P(\mathbf{v}_{rr}) \rightarrow \\ &\quad P(\mathbf{v})P(-\tfrac{1}{16} \cdot \mathbf{v}_l + \tfrac{9}{16} \cdot \mathbf{v} + \tfrac{9}{16} \cdot \mathbf{v}_r - \tfrac{1}{16} \cdot \mathbf{v}_{rr}). \end{aligned} \tag{25}$$

According to this production, the predecessor old point will be replaced by a copy of itself, followed by a newly inserted point. A different choice for mapping of old and new points results in an alternative production, in which the newly inserted point precedes the copy of the old point:

$$\begin{aligned} &P(\mathbf{v}_{ll})P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow \\ &\quad P(-\tfrac{1}{16} \cdot \mathbf{v}_{ll} + \tfrac{9}{16} \cdot \mathbf{v}_l + \tfrac{9}{16} \cdot \mathbf{v} - \tfrac{1}{16} \cdot \mathbf{v}_r)P(\mathbf{v}). \end{aligned} \tag{26}$$

The interpolating character of this subdivision scheme is further illustrated in Figure 7, which shows that points from the previous step are preserved in the next step.

Descriptions of subdivision schemes are often expressed in terms of "even" and "odd" points [38]. Odd points are newly created in the given algorithm step, whereas even points are the old points. The position of even points is preserved in the interpolating schemes, or adjusted in the approximating schemes. The L-system expression of subdivision rules makes the distinction between odd and even points unnecessary.

6 Subdividing open curves

Subdivision of open curves proceeds in a manner similar to the subdivision of closed curves, except that special subdivision rules must be applied near the curve endpoints. For example, let us consider the inference of an L-system for the Chaikin subdivision of an open curve, given the following subdivision matrix [33]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} & \cdots & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{3}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{4} & \frac{3}{4} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \tag{27}$$

Unlike previously considered matrices, which had n columns and $2n$ rows, this matrix has only $2n-2$ rows. Thus, it is no longer possible to substitute two new points for each old point. We address this problem assuming that the first and last old point will be replaced by single points, and the remaining old points will be replaced by pairs of new points. This leads to the following L-system productions:

$$\#\# < P(\mathbf{v}) \rightarrow P(\mathbf{v}) \tag{28}$$

$$\#P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\frac{1}{2} \cdot \mathbf{v}_l + \frac{1}{2} \cdot \mathbf{v})P(\frac{3}{4} \cdot \mathbf{v} + \frac{1}{4} \cdot \mathbf{v}_r) \tag{29}$$

$$P(\mathbf{v}_{ll})P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r)P(\mathbf{v}_{rr}) \rightarrow P(\frac{1}{4} \cdot \mathbf{v}_l + \frac{3}{4} \cdot \mathbf{v})P(\frac{3}{4} \cdot \mathbf{v} + \frac{1}{4} \cdot \mathbf{v}_r) \tag{30}$$

$$P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r)\# \rightarrow P(\frac{1}{4} \cdot \mathbf{v}_l + \frac{3}{4} \cdot \mathbf{v})P(\frac{1}{2} \cdot \mathbf{v} + \frac{1}{2} \cdot \mathbf{v}_r) \tag{31}$$

$$P(\mathbf{v}) > \# \rightarrow P(\mathbf{v}) \tag{32}$$

$$\# \rightarrow \# \tag{33}$$

We assume that the control polygon is represented by a sequence of at least four modules $P(\mathrm{v})$, delimited by modules #. Productions 28 and 32 state that the first and the last point of the curve will be rewritten by themselves, as specified by the first and the last row of subdivision matrix 27. Production 29 is associated with the second and third point of the subdivision matrix, and production 31 is associated with the third and second last row of that matrix in a symmetric way. Production 30 captures subdivision away from the endpoints. In essence, it is the same production as production 13 for the Chaikin subdivision of closed curves. The additional context terms, $P(\mathbf{v}_{ll})$ and $P(\mathbf{v}_{rr})$, assure that production 30 will not be applied too close to the endpoints of the curve. Finally, production 33 rewrites the endmarkers by themselves.

The above L-system can be simplified using the following conventions [26, 30]:

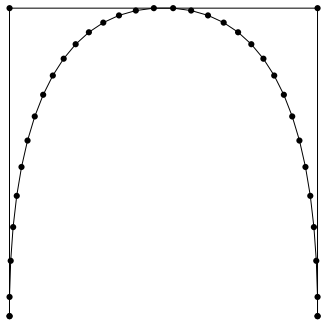


Figure 8: An open control polygon and the resulting Chaikin subdivision curve obtained using L-system productions 28 – 33 or 34 – 36.

- if no production for rewriting a particular module is explicitly listed, this module will be rewritten into itself;
- if more than one production could be used to rewrite the same module, the production that appears first in the ordered production list will be applied.

Under these conventions, the open-curve Chaikin subdivision can be defined using the following L-system productions:

$$\#P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\frac{1}{2} \cdot \mathbf{v}_l + \frac{1}{2} \cdot \mathbf{v})P(\frac{3}{4} \cdot \mathbf{v} + \frac{1}{4} \cdot \mathbf{v}_r) \tag{34}$$

$$P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r)\# \rightarrow P(\frac{1}{4} \cdot \mathbf{v}_l + \frac{3}{4} \cdot \mathbf{v})P(\frac{1}{2} \cdot \mathbf{v} + \frac{1}{2} \cdot \mathbf{v}_r) \tag{35}$$

$$P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\frac{1}{4} \cdot \mathbf{v}_l + \frac{3}{4} \cdot \mathbf{v})P(\frac{3}{4} \cdot \mathbf{v} + \frac{1}{4} \cdot \mathbf{v}_r) \tag{36}$$

These L-systems provide a complete and compact specification of the Chaikin subdivision algorithm for open curves, and directly lead to its computer implementation (*c.f.* Section 4). An application example is given in Figure 8. The reference to the curve endpoints using context and markers is less error-prone than the use of numerical limits for index values. The same methodology can be used to specify L-systems for other subdivision schemes.

7 Reverse subdivision

Bartels and Samavati introduced the notion of reverse subdivision, in which the number of points representing a curve or surface is gradually reduced, while the resulting approximations are kept within tolerable error bounds [2, 33]. Specifically, local reverse subdivision [2] inverts the paradigm of the forward subdivision: instead of replacing

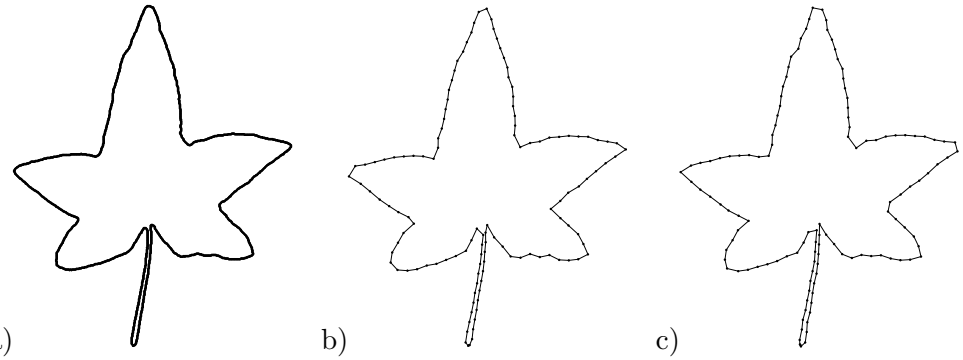


Figure 9: Reverse subdivision of a scanned ivy leaf contour. a) The input contour with 1925 points. b, c) Two approximations of the input contour obtained using production 38. Both approximations consist of 122 points, and have been obtained after four reverse subdivision steps using different circular permutations of the input string.

individual old points by subsequences of new points, it replaces subsequences of old points by individual new points. Bartels and Samavati specify this process using reverse subdivision matrices. For example, the matrix for the reverse Chaikin subdivision of a closed curve is [2]:

$$\begin{bmatrix} \frac{3}{4} & -\frac{1}{4} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & -\frac{1}{4} & \frac{3}{4} \\ -\frac{1}{4} & \frac{3}{4} & \frac{3}{4} & -\frac{1}{4} & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{4} & \frac{3}{4} & \frac{3}{4} & \cdots & -\frac{1}{4} & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & -\frac{1}{4} & \cdots & \frac{3}{4} & \frac{3}{4} & -\frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & -\frac{1}{4} & \frac{3}{4} & \frac{3}{4} & -\frac{1}{4} \end{bmatrix} \tag{37}$$

This matrix has $2n$ columns and n rows, which implies that pairs of predecessor points will be replaced by individual points. Using the grouping indicated by the encircled row, columns, and the shaded area, we obtain the following production:

$$P(\mathbf{v}_l) < P(\mathbf{v}_a)P(\mathbf{v}_b) > P(\mathbf{v}_r) \rightarrow P(-\frac{1}{4} \cdot \mathbf{v}_l + \frac{3}{4} \cdot \mathbf{v}_a + \frac{3}{4} \cdot \mathbf{v}_b - \frac{1}{4} \cdot \mathbf{v}_r) \tag{38}$$

Formally, this production is not consistent with the definition of L-systems, because its strict predecessor is not a single module. Nevertheless, an extension called pseudo-L-systems [25] makes it possible to use such productions. In a pseudo-L-system derivation step, strict predecessors are assumed to partition the predecessor string without overlaps. This is a source of nondeterminism, since different partitions may exist, leading to

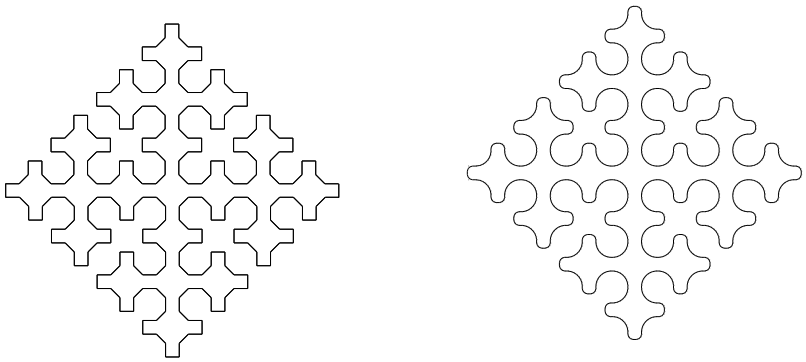


Figure 10: A Sierpinski space-filling curve (a) and its smooth version obtained using subdivision (b).

different results. For example, a circular word of length $2n$ can be partitioned into pairs $(1,2),(3,4),\ldots,(n-1,n)$ or $(2,3),(4,5),\ldots,(n,1)$. The existence of different results of a reverse subdivision step implies that the same original curve may be approximated in more than one way.

Reverse subdivision can be used, for example, to reduce the number of points approximating a measured curve. An example of such an application is shown in Figure 9.

8 Conclusions

We proposed context-sensitive parametric L-systems with affine geometry interpretation as a formal method for specifying subdivision algorithms for curves. L-systems formalize the notion of stencils and provide an intuitive yet compact and complete description of subdivision algorithms.

L-systems capture the local character of subdivision rules and the dynamic character of the subdivision process. This compatibility is closely related to the biological motivation of L-systems. They were originally proposed to describe the growth of linear structures made of locally communicating discrete elements. Subdivision can obviously be seen as an instance of such growth.

An important feature of L-system notation is that it identifies a module by its state and neighborhood. This stands in a contrast to standard mathematical notation, in which elements of a sequence are identified by indices. The index-free notation simplifies the specification and implementation of a dynamical system with dynamic structure [17]. The indices, if present, must be recalculated each time the number or configuration of components change, and thus do not provide convenient, stable identifiers of system elements. In addition, the use of indices obscures the local character of subdivision rules.

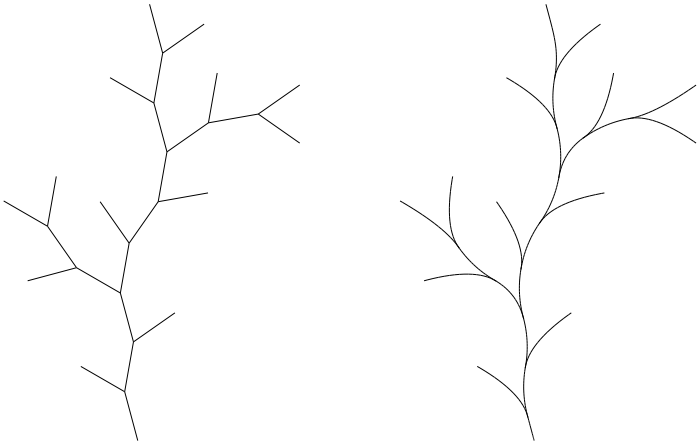


Figure 11: A branching structure (a) and the result of its smoothing (b).

We have considered the inference of L-systems, given subdivision matrices. We illustrated this inference using Chaikin, cubic B-spline, and Dyn-Levin-Gregory (4-point) subdivision schemes as examples. In addition to closed curves, discussed in more detail, we presented an example of an open curve subdivision, based on Chaikin's scheme. We have also shown that an extension of L-systems can be used to specify and implement local reverse subdivision algorithms.

We have implemented the programming language `L+C`, which makes it possible to specify L-systems as input to the modeling program, `lpfg`. This facilitates experimentation with various subdivision schemes, because not only the subdivision parameters, but also the entire subdivision algorithms, can easily be specified and modified. This makes L-systems particularly useful in research and teaching of subdivision curves.

Several problems relating L-systems to subdivision remain open for further research. For example, we observed that L-systems with affine geometry interpretation can also be used to generate fractals. This echoes the relation between subdivision curves and fractals pointed out by Warren and Weimer [37]. The possibility of integrating fractals and subdivision curves using the same L-system formalism is interesting from the theoretical perspective and may have useful applications. For instance, Figure 10 shows a finite approximation of the Sierpinski space-filling curve [34], and the result of its smoothing using Chaikin subdivision. The resulting curve is a kolam pattern, a representative of patterns that were developed as folk art in India and have attracted mathematical attention because of their self-similar structure [16, 27, 29].

Another open problem is the extension of subdivision algorithms to branching structures. An example of such a structure is shown in Figure 11a, and the result of its smoothing using Chaikin's algorithm is shown in Figure 11b. As pointed out by Bloomenthal [3], the use of curved lines increases the perception of realism in many models

of organic forms. The formalism of L-systems is useful in describing branching forms, and therefore may provide a convenient general basis for subdividing branching curves as well.

Acknowledgement

We thank Lynn Mercer for editorial help. The support of the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

References

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

[2] R. H. Bartels and F. F. Samavati. Reversing subdivision rule: local linear conditions and observations on inner products. *Journal of Computational and Applied Mathematics*, 119:29–67, 2000.

[3] J. Bloomenthal. *Skeletal design of natural forms*. PhD thesis, University of Calgary, January 1995.

[4] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[5] G. Chaikin. An algorithm for high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

[6] T. W. Chien and H. Jürgensen. Parameterized L systems for modelling: Potential and limitations. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer systems: Impacts on theoretical computer science, computer graphics, and developmental biology*, pages 213–229. Springer-Verlag, Berlin, 1992.

[7] N. Chomsky. Three models for the description of language. *IRE Trans. on Information Theory*, 2(3):113–124, 1956.

[8] T. DeRose. Three-dimensional computer graphics. A coordinate-free approach. Manuscript, University of Washington, 1992.
`http://grail.cs.washington.edu/pub/`.

[9] T. DeRose. A coordinate-free approach to geomeric programming. In W. Strasser and H.-P. Seidel, editors, *Theory and practice of geometric modeling*, pages 291–305. Springer-Verlag, Berlin, 1989.

[10] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. Proceedings of SIGGRAPH 98 (Orlando, Florida, July 19–24, 1998), pages 85–94, ACM SIGGRAPH, New York, 1998.

[11] D. Doo and M. Sabin. Analysis of the behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10(6):356–360, 1978.

[12] N. Dyn, J. Gregory, and D. Levin. A four-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4:257–268, 1987.

[13] K. A. Erstad. L-systems, twining plants, Lisp. Master's thesis, University of Bergen, Norway, January 2002. `http://www.ii.uib.no/~knute/lsystems/`.

[14] K. Joy *et. al.* On-line geometric modeling notes. Computer Science Department, University of California, Davis.
`http://graphics.cs.ucdavis.edu/CAGDNotes`.

[15] G. Farin. *Curves and surfaces for CAGD. A practical guide. Fifth edition.* Morgan Kaufmann, San Francisco, 2002.

[16] P. Gerdes. Reconstruction and extension of lost symmetries: examples from the Tamil of South India. *Computers Math. Applic.*, 17(4–6):791–813, 1989.

[17] J.-L. Giavitto and O. Michel. MGS: A programming language for the transformation of topological collections. Research Report 61-2001, CNRS - Université d'Evry Val d'Esonne, Evry, France, 2001.

[18] R. Goldman. On the algebraic and geometric foundations of computer graphics. *ACM Transactions on Graphics*, 21(1):52–86, January 2002.

[19] J. S. Hanan. *Parametric L-systems and their application to the modelling and visualization of plants.* PhD thesis, University of Regina, June 1992.

[20] M. Hausner. *A vector space approach to geometry.* Dover Publications, Mineola, 1998.

[21] R. Karwowski. *Improving the process of plant modeling: The L+C modeling language.* PhD thesis, University of Calgary, September 2002.

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

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

[24] 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.

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

[26] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Měch. Visual models of plant development. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages,* Vol. III*: Beyond words*, pages 535–597. Springer, Berlin, 1997.

[27] P. Prusinkiewicz and J. Hanan. *Lindenmayer systems, fractals, and plants*, volume 79 of *Lecture Notes in Biomathematics*. Springer-Verlag, Berlin, 1989 (second printing 1992).

[28] P. Prusinkiewicz, J. Hanan, and R. Měch. An L-system-based plant modeling language. In M. Nagl, A. Schürr, and M. Münch, editors, *Applications of graph transformations with industrial relevance*, Lecture Notes in Computer Science 1779, pages 395–410. Springer-Verlag, Berlin, 2000.

[29] P. Prusinkiewicz, K. Krithivasan, and M. G. Vijayanarayana. Application of L-systems to algorithmic generation of South Indian folk art patterns and karnatic music. In R. Narasimhan, editor, *A perspective in theoretical computer science — commemorative volume for Gift Siromoney*, pages 229–247. World Scientific, Singapore, 1989. Series in Computer Science Vol. 16.

[30] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[31] A. Rosenfeld. A note on cycle grammars. *Information and Control*, 27:374–377, 1975.

[32] M. Sabin. Subdivision surfaces. Tutorial notes, Shape Modeling International 2002 (Banff, Canada, May 18, 2002), 25 pp.

[33] F. F. Samavati and R. Bartels. Multiresolution curve and surface representation: reversing subdivision rules by least-squares data fitting. *Computer Graphics Forum*, 18(2):97–119, June 1999.

[34] W Sierpiński. Sur une nouvelle courbe qui remplit tout une aire plaine. *Bull. Acad. Sci. Cracovie, Série A*, pages 462–478, 1912. Reprinted in W. Sierpiński, *Oeuvres choisies*, S. Hartman et al., editors, pages 52–66, PWN – Éditions Scientifiques de Pologne, Warsaw, 1975.

[35] G. Siromoney, R. Siromoney, and T. Robinson. Kambi kolam and cycle grammars. In R. Narasimhan, editor, *A perspective in thepretical computer science. Commemorative volume for Gift Siromoney*, pages 267–300. World Scientific, Singapore, 1989.

[36] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for computer graphics: theory and applications*. Morgan Kaufman, San Francisco, CA, 1996.

[37] J. Warren and H. Weimer. *Subdivision methods for geometric design*. Morgan Kaufman, San Francisco, CA, 2002.

[38] D. Zorin, P. Schröder, A. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for modeling and animation. SIGGRAPH 2000 Course Notes.

L-system Implementation of Multiresolution Curves Based on Cubic B-Spline Subdivision

K. Poon*, L. Bateman, R. Karwowski, P. Prusinkiewicz and F. Samavati
University of Calgary

Abstract

It has been previously shown that L-systems can be used to generate subdivision and reverse subdivision curves [Prusinkiewicz et al. 2003]. In this paper we show that L-systems can also be used to generate multiresolution curves. The L-system description captures the locality of the concept of multiresolution curves.

1 Introduction

Finkelstein and Salesin [1994] introduced multiresolution curves as a curve representation method. The multiresolution representation supports the ability to change the overall "sweep" of a curve while maintaining its fine details, or "character". Finkelstein and Salesin used a wavelet-based notation, which uses "filters", which are represented as large matrices. Bartels and Samavati [2000] introduced a general approach to generate local filters of multiresolution curves based on reverse subdivision. In this paper, we present context-sensitive parametric L-systems as an alternative method for representing the local filters. This idea is an extension of the L-system based method for generating subdivision curves presented by Prusinkiewicz et al [2003]. The L-system notation for multiresolution curves leads to a simpler, more intuitive implementation of multiresolution curves by eliminating the need for index and matrix notation used in the traditional approach.

2 Multiresolution Curves

Multiresolution curves, introduced in [Finkelstein and Salesin 1994] allow editing of a curve's character without affecting its sweep (Figure 1) and the editing of a curve's sweep without affecting its character (Figure 2).

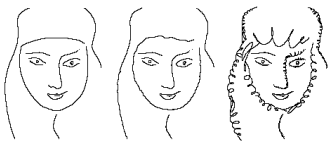


Figure 1: Editing a curve's character without affecting its sweep (from [Finkelstein and Salesin 1994]).

Multiresolution analysis can be broken into two parts: analysis and synthesis. During analysis, the original curve is coarsened. Detail information lost in this coarsening is stored. During synthesis the curve is rebuilt by performing subdivision on the coarse curve, then adding the stored detail information.

*e-mail: klpoon@cpsc.ucalgary.ca

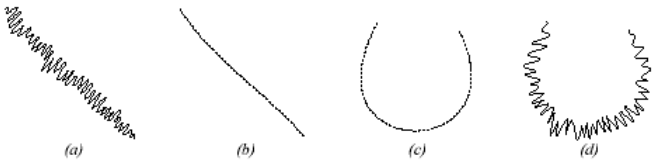


Figure 2: Editing a curve's sweep without affecting its character (from [Finkelstein and Salesin 1994]).

2.1 Matrix Notation

In matrix notation, the original curve, C^n, is represented as a vector of m points:

$$C^n = [C_1^n, C_2^n, C_3^n, ..., C_m^n]^T. \quad (1)$$

After one step of analysis the coarse curve, C^{n-1}, is a vector of m' points:

$$C^{n-1} = [C_1^{n-1}, C_2^{n-1}, C_3^{n-1}, ..., C_{m'}^{n-1}]^T. \quad (2)$$

The stored detail information, D^{n-1}, is a vector of $m - m'$ points:

$$D^{n-1} = [D_1^{n-1}, D_2^{n-1}, D_3^{n-1}, ..., D_{m-m'}^{n-1}]^T. \quad (3)$$

Analysis can be represented as two matrix multiplications:

$$C^{n-1} = A^n C^n \quad (4)$$

$$D^{n-1} = B^n C^n. \quad (5)$$

Synthesis can be represented with the matrix equation:

$$C^n = P^n C^{n-1} + Q^n D^{n-1}, \quad (6)$$

where P^n is the subdivision matrix and Q^n is the detail-restoring matrix, as determined by B-spline wavelets. A^n and B^n must satisfy biorthogonality condition:

$$\left[\frac{A^n}{B^n}\right] = [P^n | Q^n]^{-1}. \quad (7)$$

P^n is a known banded matrix for most curve schemes, however, it is not easy to compute the A^n, B^n and Q^n matrices. Finkelstein and Salesin focus on the cubic B-spline subdivision scheme. A banded but complicated Q^n is computed using B-spline wavelets. Consequently, Q^n is a local filter. A^n and B^n in that setting are full matrices, i.e., they are global filters. In order to have linear time analysis operations in equations 4 and 5, two banded linear systems are solved.

Analysis and synthesis can be carried out recursively (Figures 3 and 4).

2.2 Multiresolution Based on Reverse Subdivision

Samavati and Bartels [1999] introduced a general technique for generating multiresolution filters by reversing subdivision. This technique works for any subdivision scheme, and Q^n is a very simple matrix. However, A^n and B^n are still full matrices, i.e., global filters. In a subsequent work [Bartels and Samavati 2000], several sets of local multiresolution filters are generated based on reversing

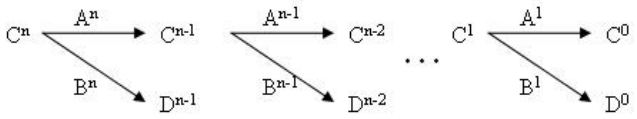


Figure 3: Applying analysis recursively (from [Finkelstein and Salesin 1994]).

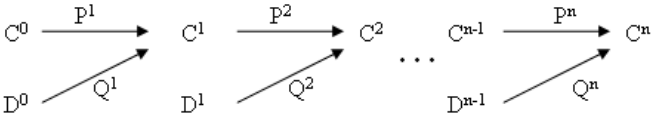


Figure 4: Applying synthesis recursively.

of subdivision schemes. We use here some of these filters. There are different filters for points near or at the endpoints of an open curve. For simplicity, we only discuss the general, not endpoint filters, although our implementation includes endpoint filters as well.

Analysis for a cubic B-spline multiresolution representation of a small, closed curve is given by the equations:

$$C^{n-1} = A^n C^n \tag{8}$$

$$\begin{bmatrix} C_1^{n-1} \\ C_2^{n-1} \\ C_3^{n-1} \\ C_4^{n-1} \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & 2 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 2 \end{bmatrix} \begin{bmatrix} C_1^n \\ C_2^n \\ C_3^n \\ C_4^n \\ \vdots \\ C_8^n \end{bmatrix}$$

and

$$D^{n-1} = B^n C^n \tag{9}$$

$$\begin{bmatrix} D_1^{n-1} \\ D_2^{n-1} \\ D_3^{n-1} \\ D_4^{n-1} \end{bmatrix} = \begin{bmatrix} \frac{3}{2} & -1 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 \\ \frac{1}{4} & -1 & \frac{3}{2} & -1 & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & -1 & \frac{3}{2} & -1 & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 & \frac{3}{2} & -1 \end{bmatrix} \begin{bmatrix} C_1^n \\ C_2^n \\ C_3^n \\ C_4^n \\ \vdots \\ C_8^n \end{bmatrix}$$

The corresponding synthesis equation is:

$$C^n = P^n C^{n-1} + Q^n D^{n-1} \tag{10}$$

$$\begin{bmatrix} C_1^n \\ C_2^n \\ C_3^n \\ C_4^n \\ \vdots \\ C_8^n \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{8} & 0 & \frac{1}{8} & \frac{3}{4} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{3}{4} & \frac{1}{8} & 0 & \frac{1}{8} \end{bmatrix} \begin{bmatrix} C_1^{n-1} \\ C_2^{n-1} \\ C_3^{n-1} \\ C_4^{n-1} \end{bmatrix}$$

$$+ \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 1 \\ \frac{1}{4} & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} D_1^{n-1} \\ D_2^{n-1} \\ D_3^{n-1} \\ D_4^{n-1} \end{bmatrix}$$

This matrix notation implies a global mapping of old points to new points at each step, but as we can see from the sparse, banded structure of the multiresolution matrices, it is possible to describe both the synthesis and analysis in local terms.

In the local approach, A, B, P and Q are filters that are applied only in the immediate neighborhood of the target points. At each iteration the filters remain the same. This contrasts the local approach from the global approach, in which the matrices change size at each analysis and synthesis iteration because they must operate on a different number of points at each iteration.

L-systems are consistent with the notion of a local algorithm. They directly capture the locality of the multiresolution algorithm.

3 L-systems

L-systems [Lindenmayer 1968] are string-rewriting systems. An L-system consists of an *alphabet*, V, an *axiom*, ω, and a set of *productions*, P, defined over V. Each production in P replaces one or more letters of V with zero or more letters in V. A *word*, x, in the system is a sequence of letters in V. The system's current state is represented by a word. The axiom, ω, is a special word, which represents the system's initial state. At each time step, the production rules are applied in parallel to each of the letters in the current word to produce a new word.

Parametric L-systems [Prusinkiewicz and Lindenmayer 1990] extend the basic concept of L-systems by assigning additional attributes (*parameters*) to L-system symbols. A parameter can be a number, or a C++ structure [Karwowski 2002]. A *module* in a parametric L-system consists of a letter in V and zero or more parameters. In parametric L-systems, a word consists of a sequence of modules.

We implemented multiresolution curves in the language L+C, which adds L-system constructs to C++ [Karwowski 2002]. In L+C, the syntax for declaring a module with parameters is:

```
module identifier(list of parameter types).
```

The following code declares a module, `C`, which represents a point. `C` has a parameter of type `V2f`, which is a pre-defined L+C type that represents a 2D point or vector:

```
module C(V2f);
```

L+C supports context-sensitive productions. A context sensitive production replaces a module, called the strict predecessor, using information from neighboring modules, called the left and right context. The syntax for a context sensitive production is

```
lcontext < strict predecessor > rcontext:
{
  ...
}
```

The following code replaces each point with two points, each of which is a linear combination of the point being replaced and its left or right neighbor.

```
P(vl) < P(v) > P(vr):
{
  produce P(0.25vl+0.75v) P(0.75v+0.25vr);
}
```

It is possible in L+C to look to the new string for context. The << and >> symbols means look to the left and right, respectively, in the produced string.

L+C also supports table L-systems [Rozenberg 1973]. This allows us to divide productions into groups. We specify which group of productions should be used at each derivation step. For example, the following L+C code segment applies the production A→AB in even steps and A→AC in odd steps.

```
int step = 0;
StartEach:{
  if(step%2 == 0) UseGroup(0);
  else UseGroup(1);
  step++;
}
group 0:
A(): {
  produce A() B()
};
group 1:
A(): {
  produce A()C()
};
endgroup
```

4 Implementing Cubic B-spline Multiresolution Curves in L+C

4.1 Topology

During multiresolution analysis, detail information accumulates. We keep detail information associated with those points it will be used to restore, but after each iteration of analysis the amount of detail we need to store for each point will more than double. To solve this problem we use a tree data structure on which L+C operates.

Tree structures can be represented within an L-system string using special branch symbols [and]. The symbol [denotes the start of a branch and the symbol] denotes the end of a branch. The branches can be nested to create trees. For example, we interpret the string A [[B] C [D]] as the tree in Figure 5.

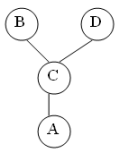


Figure 5: The tree represented by the string A [[B] C [D]].

Below we present, step-by-step, the operation of an L-system that performs the topological changes that occur during multiresolution analysis. The L-system converts a string of modules which represent points into a string in which every other module represents detail information. It then associates each piece of detail information with a point and repeats the process, storing detail information in a tree structure. During synthesis, we need to strips off layers of branching in order to access the detail information that is associated with each point.

Figure 6 shows changes in a string and the topological changes it represents. C is a point and D is detail information. The downward arrow shows the topology changes during analysis. The upward arrow shows the topology changes during the synthesis.

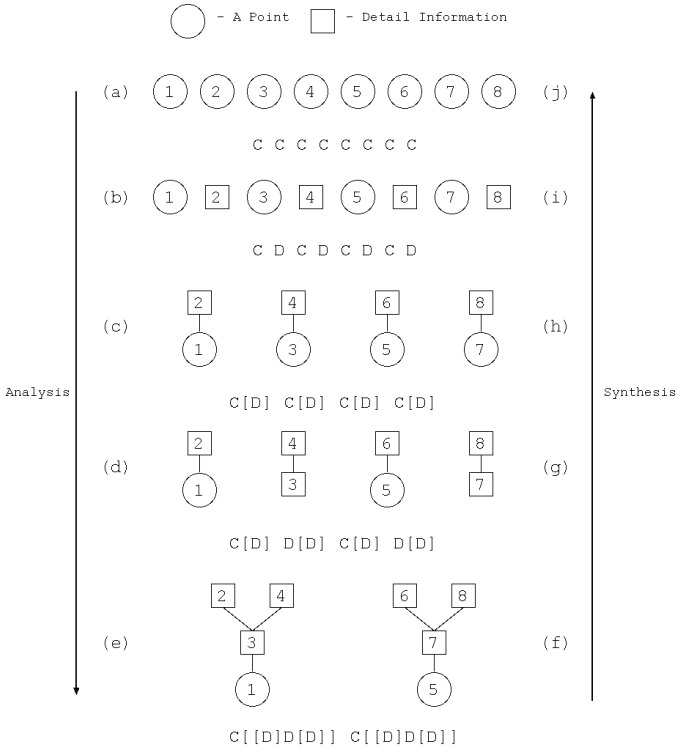


Figure 6: Topology changes during synthesis and analysis. (a) The original points. (b) and (d) Every other point is converted to detail information. (c) and (e) Each piece of detail information is associated with a point. Detail information from the previous steps is stored as branches of the detail trees. (f) Original string at start of synthesis. (g) and (i) Outer branches are stripped away. (h) and (j) Detail information is converted back into points.

4.2 L-systems Rules for Multiresolution

In this section we present our L+C implementation of multiresolution curves. This implementation is limited to closed curves. The full code listing for closed curves is given in Appendix A.

4.3 Analysis

During analysis the symbol C represents a point. The symbol Dt represents a point that is about to become a detail vector. The symbol D represents a detail vector.

We begin with a sequence of points that represents the original curve:

```
C C C C C C C ... C C
```

During analysis, even points become detail information and odd points become a point on the coarsened curve. There are two separate analysis phases.

During phase zero, we change the type of every other point to a type that represents detail information. The L+C production that changes every other point, `C`, into a point that is about to be converted into detail information, `Dt`, is:

```
C(vl)<< C(v) : {
  produce Dt(v);
}
```

If the module to the left of the strict predecessor in the produced string is a `C`, this rule replaces the strict predecessor with a `Dt`. After this phase, the string has the form:

```
C Dt C Dt C Dt C Dt...
```

In phase one we calculate the positions of the coarse points and the value of the detail vectors. The two main rules are the reverse subdivision rule, corresponding to the application of filter A, and the detail-storage rule, corresponding to the application of filter B. The reverse subdivision rule coarsens a set of fine points. An L-system implementation of reverse subdivision has been presented in [Prusinkiewicz et al. 2003]. This implementation replaces a pair of points with a single point. Our implementation replaces a single point with a point. This difference arises because we convert every other point to detail information, whereas in plain reverse subdivision, we do not need to store the detail information. Our L+C code to perform reverse subdivision is:

```
Dt(vl) < C(v) > Dt(vr):{
  produce EB() C(-0.5*vl + 2*v + -0.5*vr) SB() H();
}
```

This production calculates the new location of a point, based on the reverse subdivision coefficients given by the A matrix in equation 8. An end branch module is inserted before the new coarse point and a start branch module is inserted after the new coarse point. This puts the detail information to the right of each point into a tree associated with that point. An `H` module is placed after each start branch module as a block that prevents modules within the tree from "seeing" modules outside the tree.

The detail information lost in the reverse subdivision process is stored within the `D` modules. The following L+C production implements the detail storage filter B:

```
Dt(vll) C(vl) < Dt(v) > C(vr) Dt(vrr) :{
  produce
   D(0.25*vll + -1*vl + 1.5*v + -1*vr + 0.25*vrr);
}
```

This production calculates the detail information to store based on coefficients given by the B matrix in equation 9.

For example, after the first iteration of analysis, the string has the form:

```
C [ D ] C [ D ] C [ D ] C [ D ]... ,
```

and after three iterations of analysis the string has the form:

```
C [ [ [ D ] D [ D ] ] D [ [ D ] D [ D ] ] ]... .
```

4.4 Synthesis

There are two synthesis phases. In phase zero, the productions strip away one layer of bracketing. In phase one, detail information is used to replace each coarse point by one of its two original fine points. The corresponding detail vector is replaced with the other fine point.

The production rule that strips away the start brackets is:

```
C(v) < SB() H() : {
  produce ;}
```

This production rule removes all starting brackets that are directly to the right of a point. Only the outer brackets are removed.

The production rule that strips away the end brackets is:

```
EB() > C(v) : {
  produce ;}
```

This production rule removes all end brackets that are to the left of a point. Again, only the outer brackets are removed.

For example, the string with the form:

```
C [ [ [ D ] D [ D ] ] D [ [ D ] D [ D ] ] ] C...
```

will have the form:

```
C [ [ D ] D [ D ] ] D [ [ D ] D [ D ] ] C...
```

after one iteration of phase zero. Notice that only the outer brackets have been removed. Now the leftmost `C` can "see" the fourth `D` from the left.

In phase one, subdivision is performed on the coarse points, `C`. Detail information stored in adjacent `D` vectors are added to the subdivided points to restore the original fine points. The two productions for this phase calculate the new location of a fine point using coefficients from the P and Q matrices given in equation 10.

This is the rule that replaces each coarse point with one of the restored fine points:

```
C(vll) D(vl) < C(v) > D(vr) C(vrr) : {
  produce C(0.125*vll + 0.75*v + 0.125*vrr
              + 0.25*vl + 0.25*vr);}
```

The fine point is a combination of a point created by subdivision: $0.125 * vll + 0.75 * v + 0.125 * vrr$, and detail information, $0.25 * vl + 0.25 * vr$.

This is the rule that replaces each module representing detail with a module representing a restored fine point

```
C(vl) < D(v) > C(vr): {
  produce C(0.5*vl + 0.5*vr + v);}
```

The fine point is a combination of a point created by subdivision, $0.5 * vl + 0.5 * vr$, and detail information, v.

After one iteration of phase one, the string has the form:

```
C [ [ D ] D [ D ] ] C [ [ D ] D [ D ] ] C... .
```

Once synthesis has been performed the same number of times analysis was performed, the original fine curve is restored.

The productions we discussed above deal with closed curves (we assume the required left and right context is always present). The complete listing of the L+C code for multiresolution representation of closed curves based on the above productions is given in Appendix A.

5 Extensions

5.1 Open Curves

We have also implemented multiresolution for open curves. The difference between the code for the open curve case and the closed curve case is that there are special endpoint rules for the open curve case. These special endpoint rules are included in Appendix B.

5.2 Aligning Detail with Normal

When we restore detail during synthesis, it has the same x-y orientation as the detail in the original curve. If the modified curve has a different slope than the original curve, the detail will look incorrect. To address this problem, Finkelstein and Salesin [finkelstein:multi] also introduced the idea of aligning the detail with the normal of the curve.

We approximate the normal, $\vec{N}$ at a given point, v, by finding the vector perpendicular to an approximated tangent vector, $\vec{T}$. We approximate $\vec{T}$ by taking the difference of the two points adjacent to v:

$$\vec{T} = v_r - v_l \tag{11}$$

During synthesis, instead of adding the detail directly back into the curve, we multiply the signed magnitude of the detail by the normal vector and add this aligned detail to the curve.

We have included the modified synthesis rules for aligning detail with the normal in Appendix C.

6 Results

Figure 7 shows a open curve which is coarsened, has its sweep modified, then is reconstructed. Figure 8 shows a branching structure with a modified sweep. Figures 9 shows a leaf with two different modified sweeps. Figure 10 compares the results of sweep modification of a curve (a). In 10 (b), the detail is not aligned with the normal. In 10 (c), the detail is aligned with the normal.

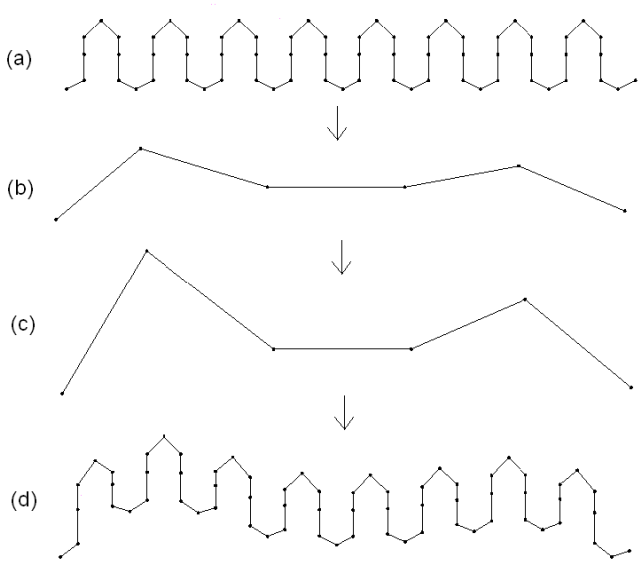


Figure 7: (a) Original curve. (b) Coarsened curve. (c) Modified coarse curve. (d) Reconstructed curve.

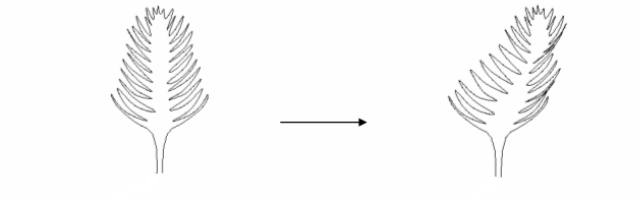


Figure 8: A branching structure with modified sweep.

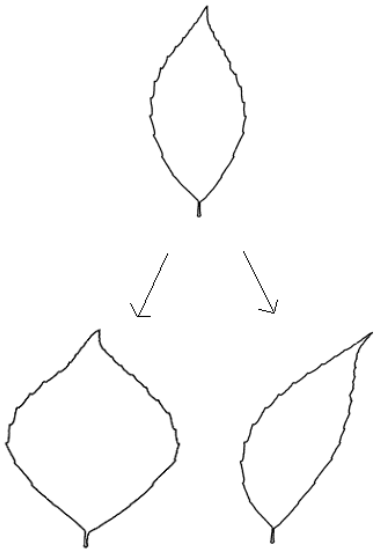


Figure 9: A scanned leaf with two modified sweeps.

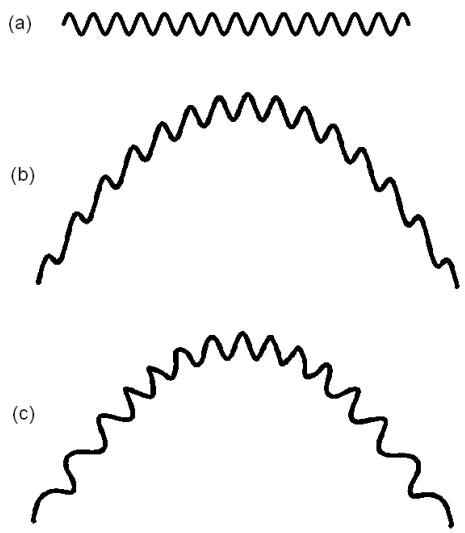


Figure 10: (a) Original curve. (b) Curve with modified sweep and detail not aligned with normal. (c) Curve with modified sweep and detail aligned with normal.

References

BARTELS, R. H., AND SAMAVATI, F. F. 2000. Reversing subdivision rules: Local linear conditions and observations on inner products. *Journal of Computational and Applied Mathematics 119*, 1–2, 29–67.

FINKELSTEIN, A., AND SALESIN, D. 1994. Multiresolution curves. In *Proceedings of SIGGRAAPH '94*, 261–268.

KARWOWSKI, R. 2002. *Improving the Process of Plant Modeling: The L+C Modeling Language*. PhD thesis, University of Calgary.

LINDENMAYER, A. 1968. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology 18*, 280–315.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1990. *The Algorithmic Beauty of Plants*. Springer, New York.

PRUSINKIEWICZ, P., SAMAVATI, F., SMITH, C., AND KARWOWSKI, R. 2003. L-system description of subdivision curves. To appear in the International Journal of Shape Modeling.

ROZENBERG, G. 1973. T0L systems and languages. *Information and Control 23*, 357–381.

SAMAVATI, F. F., AND BARTELS, R. H. 1999. Multiresolution curve and surface representation by reversing subdivision rules. *Computer Graphics Forum 18*, 2, 97–120.

7 Appendix A - The L+C Code for Multiresolution B-spline Representation of Closed Curves

```
#include <lpfgall.h>

//Step at which to switch between coarsing and
refinement
#define SWITCH 8
#define NUMSTEPS 16

// The phase types
#define ANALYSIS_0 0
#define ANALYSIS_1 1
#define SYNTHESIS_0 2
#define SYNTHESIS_1 3

// String end marker
module E();
// An obstacle for context-matching purposes
module H();

// A point module
module C(V2f);
// Detail information module
D(V2f);
// A point that is about to be converted to detail
Dt(V2f);

int step;

Start: { step = 0;} StartEach: {
    // set the phase based on the step
    if(step < SWITCH)
        if(step%2==0) UseGroup(ANALYSIS_0);
        else UseGroup(ANALYSIS_1);
    else
        if(step%2==0) UseGroup(SYNTHESIS_0);
        else UseGroup(SYNTHESIS_1);
}

EndEach: { step++;}

derivation length: NUMSTEPS;

ring L-system: 1;

// axiom that defines initial curve goes here

/*********************************************
 * Analysis (Reverse Subdivision)
 *********************************************/

////////////////////////////////////////////////
// ANALYSIS_0: change every other C into a D
////////////////////////////////////////////////
group ANALYSIS_0:

C(vl) << C(v) : {
    produce Dt(v) ;
}

////////////////////////////////////////////////
//ANALYSIS_1: Analysis (Perform reverse
//                      subdivision)
// (Store coarse points in C's and detail in D's)
////////////////////////////////////////////////
group ANALYSIS_1:

// C rule
Dt(vl) < C(v) > Dt(vr): {
    produce EB() C(-0.5*vl + 2*v -0.5*vr) SB() H();
}

// D rule
Dt(vll) C(vl) < Dt(v) > C(vr) Dt(vrr) : {
    produce D(0.25*vll -1*vl + 1.5*v -
              1*vr + 0.25*vrr);
}

/*********************************************
 * Synthesis (Subdivision)
 *********************************************/

////////////////////////////////////////////////
// SYNTHESIS_0: Eliminate outermost brackets
// [Branch] S [Branch] ]  --> [Branch] S [Branch]
////////////////////////////////////////////////
group SYNTHESIS_0:

C(v) < SB() H() : {
    produce ;
}

EB() > C(v) : {
    produce ;
}

////////////////////////////////////////////////
// SYNTHESIS_1: Synthesis (Perform subdivision)
//     (Use information from coarse points, C, and
//     details, D, to restore original points, C)
////////////////////////////////////////////////
group SYNTHESIS_1:

// C rule
D(dl) C(vll) D(vl) < C(v) > D(vr) C(vrr) D(dr) : {
    produce C(0.125*vll + 0.75*v + 0.125*vrr
              + 0.25*vl + 0.25*vr);
}
```

```
// D rule
D(dl) C(vl) < D(v) > C(vr) D(dr): {
    produce C(0.5*vl + 0.5*vr + v);
}

endgroup

/*********************************************
 * Drawing
 *********************************************/

interpretation:

C(v) : {
    produce SetColor(7) MoveTo2f(v) Circle(0.1) ;
}

Dt(v) : {
    produce SetColor(4) MoveTo2f(v) Circle(0.1) ;
}
```

8 Appendix B - Special Rules for Multiresolution B-spline Representation of Open Curves

```
/*********************************************
 * Analysis (Reverse Subdivision)
 *********************************************

/* Left-most endpoint rules */

// C rules
E() < C(v) : {
    produce C(v);
}

E() C(vl) < C(v): {
    produce C(-1*vl + 2*v) SB() H();
}

// D rule

E() C(vll) C(vl) < Dt(v) > C(vr) Dt(vrr): {
    produce D(0.75*vll -1.5*vl + 1.125*v -
              0.5*vr + 0.125*vrr);
}

/* Right-most endpoint rules */

// C rules

C(v) > E(): {
    produce C(v);
}

C(v) > C(vr) E(): {
    produce EB() C(-1*vr + 2*v);
}

// D rule

Dt(vll) C(vl) < Dt(v) > C(vr) C(vrr) E(): {
    produce D(0.75*vrr -1.5*vr + 1.125*v -
              0.5*vl + 0.125*vll);
}

/*********************************************
 * Synthesis (Subdivision)
 *********************************************/

/* Left Endpoint Rules */

// C rules

E() < C(v): {
    produce C(v);
}

E() C(vl) < C(v) : {
    produce C(0.5*vl + 0.5*v);
}

E() C(d) C(vll) D(vl) < C(v) > D(vr) C(vrr): {
    produce C(0.1875*vll + 0.6875*v + 0.125*vrr
              + 0.25*vl + 0.25*vr);
}

// D rule

E() C(d) C(vl) < D(v) > C(vr) : {
    produce C(0.75*vl + 0.25*vr + v);
}

/* Right Endpoint Rules */

// C rules

C(v) > E(): {
    produce C(v);
}

C(v) > C(vr) E(): {
    produce C(0.5*vr + 0.5*v);
}

C(vll) D(vl) < C(v) > D(vr) C(vrr) C(d) E(): {
    produce C(0.1875*vrr + 0.6875*v + 0.125*vll
              + 0.25*vr + 0.25*vl);
}

// D rule

C(vl) < D(v) > C(vr) C(d) E(): {
    produce C(0.75*vr + 0.25*vl + v);
}
```

9 Appendix C - Synthesis Rules for Alignment of Detail with Normal Vectors

```
// returns the length of vec
double vecLength(V2f vec) {
  double length = sqrt(vec.x*vec.x + vec.y*vec.y);
  return length;
}

// calculate a normal vector given a tangent vector
V2f normalFromTangent(V2f tangent) {
```

```
    // find the length of the tangent so we can
    // normalize
    double length = vecLength(tangent);

    V2f normal;
    if(length != 0){
        // the normal is the normalized tangent
        // rotated by pi/2
        normal.x = -(1.0/length)*tangent.y;
        normal.y = (1.0/length)*tangent.x;
    } else{
        normal.x = 0.0;
        normal.y = 1.0;
    }
    return normal;
}

// calculate the point with detail restored in the
// normal direction
 V2f getDetailAddedPoint(V2f point, V2f detail,
                         V2f vLeft, V2f vRight) {
    // approximate the tangent based on
    // neighboring points
    V2f tangent = vRight - 1*vLeft;

    // get the approximate normal vector
    V2f normal = normalFromTangent(tangent);

    // get the magnitude and direction of the
    // detail vector
    double length = vecLength(detail);
    if(detail.y < 0) length = -length;

    // produce the point
     return (point + length*normal);
}

////////////////////////////////////////////////
// Phase 1: Synthesis (Perform subdivision)
// (Use information from coarse points, C, and
// details, D, to restore original points, C)
////////////////////////////////////////////////
group SYNTHESIS_1:

/* Main Rules */

// C rule

D(dl)
    C(vll) D(vl) < C(v) > D(vr) C(vrr)
                                D(dr) : {
    V2f point = 0.125*vll + 0.75*v + 0.125*vrr;
    V2f detail = 0.25*vl + 0.25*vr;
    V2f dPoint =
        getDetailAddedPoint(point, detail, vll, vrr);
    produce C(dPoint);
}

// D rule

D(dl)
    C(vl) < D(v) > C(vr)
                D(dr): {
    V2f point = 0.5*vl + 0.5*vr;
    V2f detail = v;
    V2f dPoint =
        getDetailAddedPoint(point, detail, vl, vr);
    produce C(dPoint);
}
```

Relational Specification of Surface Subdivision Algorithms

Colin Smith, Przemyslaw Prusinkiewicz, and Faramarz Samavati
Department of Computer Science
University of Calgary

Abstract

Many polygon mesh algorithms operate in a local manner, yet are formally specified using global indexing schemes. We address this discrepancy by defining a set of local operations on polygon meshes in relational, index-free terms. We also introduce the vv programming language to express these operations in a machine-readable form. We then apply vv to specify several surface subdivision algorithms. These specifications can be directly executed by the corresponding modeling software.

1 Introduction

Ideally, a problem description should clearly reflect its nature. The data should be organized to reflect the relations inherent in the problem, and the language used to describe the solution should focus on its essence. Superfluous elements should be avoided, as they obfuscate the nature of the problem and its solution. When the data and relations are elegantly organized, the solution often becomes simple and easy to understand and implement.

The particular problem that we address in this paper is that of dealing with local properties and local transformations of polygon meshes. Locality is one of the most fundamental characteristics of systems. It means that: (a) a neighborhood relation is defined on the elements of the system, and (b) each element of the system changes its state according to its own state and the state of its neighbors, to the exclusion the elements positioned farther away. The search for, and study of, local mechanisms that underpin observed phenomena has been one of the central and fruitful directions in natural sciences, from physics to biology.

Many computer graphics algorithms also have local character. A good example is given by subdivision algorithms in geometric modeling. Their locality is intuitively captured when subdivision algorithms are described in terms of *masks* [31] (also referred to as *stencils* [27]). The prevalent formal definitions of subdivision algorithms, however, do not take advantage of the simplicity and locality of the masks, but rely on a global enumeration (indexing) of the polygon mesh elements. The appeal of indices is that they are a standard mathematical notation, and are closely coupled with the array data structures supported by most programming languages. Unfortunately, they are deficient in several respects:

- The use of indices does not adhere to the philosophy of describing local processes in local terms. For example, one can write the expressions $i - 1$ or $i + 1$, which refer to the immediate neighbors in a linear structure, as easily as $i - 100$ or $2i$, which do not.
- The indexed elements are often not arranged into a regular grid. This complicates the indexing scheme and index arithmetic, and makes them error-prone.
- The indexed elements must be dynamically renumbered as their number and arrangement change.
- Indexed notation has questionable value from the viewpoint of algorithm implementation, which may use a different indexing scheme than that used to specify the algorithm, or rely on pointers rather than index arithmetic when identifying neighbors.
- Index-heavy notation is hard to read.

We address these deficiencies by introducing the *vertex-vertex polygon mesh representation*, and the corresponding set of operations, the *vertex-vertex algebra*, which make it possible to describe local operations on polygon meshes in relational terms. This means that we identify elements of the structure with respect to each other, avoiding absolute identifiers such as coordinates or indices. We also introduce vv, an extension of the C++ programming language, for expressing these operations in a machine-readable form. This results in the *language + engine* modeling paradigm, which simplifies the implementation of individual algorithms by treating them as input to a multi-purpose modeling program. We demonstrate the usefulness of this paradigm by presenting concise vv specifications of several subdivision algorithms.

2 Background

Local modifications to structured objects are the essence of development. Consequently, the problem of referring to the neighbors often occurs in biologically-motivated models of computation. In cellular automata, for example, the neighbors of a given cell may be specified using index arithmetic, or in a relational manner, using the directions north, south, east and west. Giavitto and Michel [10] explored the advantages of the relational approach and generalized it to arbitrary regular tessellations (group-based fields). They have also proposed a programming language to capture locally defined structures that, unlike cellular automata and group-based-fields, may grow and dynamically reconfigure [11].

Growing geometric structures with linear and branching topology, have been constructed since the late 1960s as models of multicellular organisms, in particular plants. The relational approach to the identification of the elements of these structures is exemplified by the formalism of L-systems [19, 24]. A structure is represented by a sequence of symbols. This sequential arrangement automatically determines the neighborhood relations between the elements.

L-systems with affine geometry interpretation have recently been shown to provide a compact formal specifications of subdivision algorithms for curves [25]. For example, Chaikin's corner-cutting algorithm [6] is given by the production:

$$P(v_l) < P(v) > P(v_r) \rightarrow P\left(\tfrac{1}{4}v_l + \tfrac{3}{4}v\right) P\left(\tfrac{3}{4}v + \tfrac{1}{4}v_r\right). \qquad (1)$$

Here $P(x)$ denotes a point at location x; symbols $<$, $>$ and $\rightarrow$ separate the left context, the strict predecessor, the right context, and the successor of the production; and the arithmetic operators specify the affine combinations of the argument points. In a process akin to cell division in a developing organism, this production replaces the parent point by two descendant points, with the locations dependent on the context (Figure 1).

The simplicity, clarity, and compactness of the L-system specification of subdivision curves, combined with the possibility of executing them using an L-system-based modeling software [25], have motivated our quest for an extension that could be applied to polygon meshes as well. Unfortunately, the existing grammar-based formalisms fall short of this goal. *Map L-systems* [20, 24] can generate the topology of some polygon meshes, but do not offer flexible control over the resulting geometry and are difficult to specify. Similarly, *graph grammars* [26] extend formal grammars from

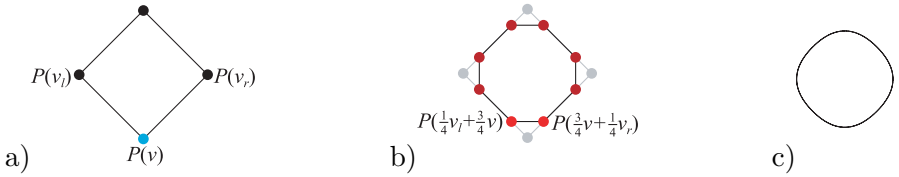


Figure 1: Chaikin subdivision process described by an L-system production (Equation 1). a) The initial polygon. Labels refer to an arbitrarily chosen point $P(v)$. b) The result of the first iteration of the algorithm. c) The curve after several subdivision steps.

strings to graphs. However, the development of graph theory has been focused on the context-free case, and more general formulations are difficult to use.

We attribute the deficiencies of graph grammars to the loss of information that occurs during production application. The predecessor is removed from the structure before the successor is inserted into it, and thus details of the predecessor's connections are not available for reconnecting the successor. To address this problem, we propose to operate in a more gradual manner, possibly adding new nodes and edges before the old ones have been removed. The old elements may thus serve as a scaffolding for introducing the new ones. This technique preserves the purely local operation of the grammar-based approaches, but departs from their declarative character, because modifications to structures are now specified as sequences of imperative operations.

The ease of performing local operations on polygon meshes depends on the mesh representation, which should be conducive to relational information gathering and mesh transformations. Well known examples of such representations include the *winged-edge* [4], and *quad-edge* [12] representations. Pursuing objectives closer to ours, Egli and Stewart [9] applied *cellular complexes* [23] to specify Catmull-Clark [5] subdivision in a relational manner. Lienhardt [18] showed that local operations involved in subdivision algorithms can be defined using *G-maps* [16, 17]. More recently, Velho [29] developed a method for describing subdivision algorithms using stellar operators [15] that act on a *half-edge* structure [22].

We have selected yet another representation, based on the mathematical notion of *graph rotation systems* [8, 30]. A rotation system associates each vertex of a polygon mesh with an oriented circular list of its neighboring vertices. A set of these lists, defined for each vertex, completely represents

the topology of a 2-manifold mesh [30]. Graph rotation systems have been introduced to computer graphics by Akleman, Chen and Srinivasan [1, 2, 3] as a formal basis for the *doubly linked face list* representation of 2-manifold meshes. Akleman et al. have also defined a set of operations on this representation, which they used to implement interactive polygon mesh modeling tools. Below we introduce *vertex-vertex systems* as a different data structure related to the graph rotation systems. It makes it possible to implement a set of graph manipulation operations in an intuitive and efficient manner.

3 Vertex-vertex systems

3.1 Definitions

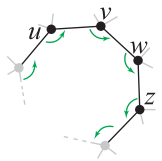


Figure 2: A polygon identification in a graph rotation system.

Let U be an enumerable set, or the *universe*, of elements called *abstract vertices*. We assume that U is ordered by a relation $<$; this assumption simplifies the implementation of many algorithms (Section 4). Next, let $N : U \mapsto 2^U$ be a function that takes every vertex $v \in U$ to a finite subset $v^\star \subset U$ of other vertices ($v \notin v^\star$). We call the set $v^\star$ the *neighborhood*, and its elements the *neighbors*[1] of v. Finally, let the *vertex set* $S \subset U$ be a finite subset of the universe U, and N_S be the restriction of the neighborhood function N to the domain S; thus $N_S(v) = v^\star$ if $N(v) = v^\star$ and $v \in S$ (the elements of $v^\star$ may lay outside S). We call the pair $\langle S, N_S \rangle$ a *vertex-vertex structure* over the set S with neighborhood N_S.

An *undirected graph* over a vertex set S is a vertex-vertex structure over S, in which: (a) all neighborhoods are included in S (the vertex set S is *closed* with respect to the function N), and (b) vertex u is in the neighborhood of v if an only if vertex v is in the neighborhood of u ($u \in v^\star$ if and only if $u \in v^\star$, the *symmetry condition*). The pairs (u, v) of vertices that are in the neighborhood of each other are called *edges* of the graph. An edge is *oriented* if the pair (u, v) is considered different from (v, u).

A *vertex-vertex rotation system*, or *vertex-vertex system* for short, is a vertex-vertex structure in which the vertices in each neighborhood form a cyclic permutation (i.e., are arranged into a circular list). A *graph rotation system* is a vertex-vertex system that is both a graph and a vertex-vertex

[1] Our terminology is motivated by the practice of referring to adjacent cells in a grid as neighbors. It should not be confused with the definition of neighborhood in topology.

rotation system.

A *polygon mesh* is a collections of vertices, edges bound by vertex pairs, and polygons bound by sequences of edges and vertices. A mesh is a *closed 2-manifold* if it is everywhere locally homeomorphic to an open disk, and a *2-manifold with boubdary* if it is everywhere locally homeomorphic to an open disk or half-disk. A manifold is *orientable* if it has two sides [30].

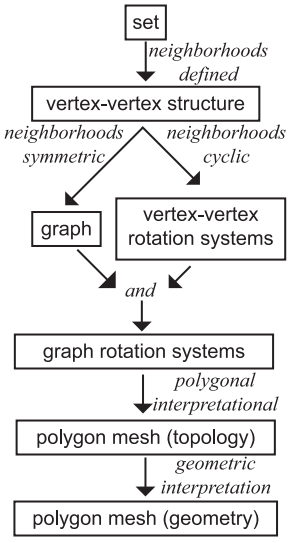


Figure 3: Relations between notions pertinent to vertex-vertex systems

A *polygonal interpretation* of a vertex-vertex system maps it into a polygon mesh. The interpretations that we consider in this paper are variants of the *Edmonds' permutation technique* [8, 30, 2], which is defined for connected graph rotation systems. It defines polygons of the mesh using the following algorithm (Figure 2). Given an oriented edge (u, v) in S, we find the oriented edge (v, w) such that w immediately follows u in the *cyclic* neighborhood of v. Next, we find the oriented edge (w, z) such that z immediately follows v in the neighborhood of w. We continue this process until we return to the starting point u. The resulting *orbit* (cyclic permutation) of vertices $u, v, w, z, \ldots$ and the edges that connect them are the boundaries of a polygon. By considering all such orbits in S, we obtain a polygon mesh with polygons on both sides of each (unoriented) edge. From this construction it immediately follows that the resulting mesh is a uniquely defined, orientable, closed 2-manifold (see [30] for a formal proof).

A function f defined on a vertex set S assigns a *property* $f(v)$ to each vertex $v \in S$. In addition to the neighborhoods defined above, vertex properties may include for example a label (drawn from a finite or an infinite set), position, normal vector, and color.

Vertex *positions* are a crucial aspect of the *geometric interpretation* of vertex-vertex systems. We will consider geometric interpretations in which edges are drawn as straight lines between vertices, and polygons are properly defined if their vertices and edges are coplanar.

The above progression of notions is summarized in Figure 3. It suggests that polygon meshes can be manipulated using operations defined on sets (set-theoretic operations), vertex-vertex systems and graphs (topological operations), and polygon meshes (geometric operations). The crucial problem is the manipulation of topology. We address it by introducing a set of op-

erations that modify at most one neighborhood at a time, and transform a vertex-vertex system into another vertex-vertex system. The individual operations do not necessarily transform graphs into graphs, because they may create *incomplete neighbors* that violate the symmetry condition ($u \in v^\star$ but $v \notin u^\star$).

3.2 The vertex-vertex algebra

The *vertex-vertex algebra* is the class of vertex-vertex rotation systems with a set of operations defined on them. We introduce these operations using a mathematical notation that combines standard and new mathematical symbols. We also present the equivalent expressions and statements of the vv language. A further description of this language and its implementation is given in Section 3.3.

3.2.1 Set-theoretic operations

In the vv language, vertex sets are a predefined data type. A set S is created using the declaration mesh S, and is in existence according to the standard scoping rules of C++. The vv language supports a subset of the standard set operations, listed in Table 1. In addition to operations that return a set as the result, vv includes iteration operators for flow control in vv programs.

Name	Math. notation	vv statement
set creation	let $S \subset U$	mesh S
assignment	$S = T$	$S = T$
union	$S = S \cup T$	merge S with T
addition of an element	$S = S \cup \{v\}$	add v to S
removal of an element	$S = S - \{v\}$	remove v from S
iteration over a set	$\forall v \in S$	forall v in S
iteration over neighbors	$\forall x \in v^\star$	forall x in v

Table 1: Set-theoretic operations supported by the vv language

3.2.2 Topological operations

Topological operations are the core of the vertex-vertex algebra. They are divided into three groups: *query*, *selection*, and *editing* operations. Query operations return information about vertices. Selection operations return an element of a vertex neighborhood. Editing operations modify a vertex-

vertex system. Definition of these operations are given in Table 2. The last column in this table refer to the illustrations in Figure 4.

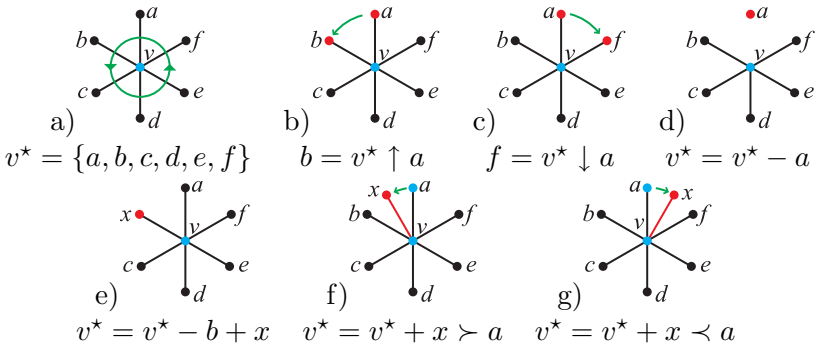


Figure 4: Examples of operations in the vertex-vertex algebra. a) Setting the initial neighborhood of vertex v. b-g) The results of selection and editing operations applied to v.

3.2.3 Geometric operations

We use the standard functional notation $f(v)$ or vv expression v\$f to associate property f with a vertex v. A special case is the position of a vertex, denoted $\overline{v}$ or v\$pos. Positions can be assigned explicitly, by referring to an underlying coordinate system, or result from affine geometry combinations and vector operations applied to the previously defined points. We use the standard C++ operator overloading mechanism to extend arithmetic operators to positions and vectors.

3.2.4 Coordination operations

Operations of the vertex-vertex algebra are commonly iterated over vertex sets. This raises important questions concerning the sequencing of these individual operations. For example, if the same operation is to be performed on a pair of neighboring vertices u and v, the results may be different depending on whether u is modified first, v is modified first, or both vertices are modified simultaneously. To eliminate the unwanted dependence on the execution sequence, we introduce the *coordination operation* synchronize S, which creates a copy $`v$ of each vertex $v$ in the set $S$. All subsequent operations on the vertices $v \in S$ (until the next *synchronize* statement) do not affect the vertices $`v$, which continue to store the "old" values of vertex

Name	Math. notation	vv statement	Description	Note	Fig
		Query operations			
membership	$v \in x^\star$	is x in v	true iff vertex x is in the neighborhood of v		
order	$x < v$	$x < v$	true iff vertex x precedes vertex v in the universe U		
valence	$\|v^\star\|$	valence v	returns the number of neighbors of vertex v		
		Selection operations			
any	let $v \in x^\star$	any in v	returns a random neighbor of v	1	
next	$v^\star \uparrow x$	nextto x in v	returns vertex that follows x in the neighborhood of v	2	b
previous	$v^\star \downarrow x$	prevto x in v	returns vertex that precedes x in the neighborhood of v	2	c
		Editing operations			
create	let $v \in U$	vertex v	create vertex		
set neighborhood	$v^\star = \{a, b, c\}$	make $\{a, b, c\}$ nb_of v	set the neighborhood of v to the given circular list	3	a
erase	$v^\star = v^\star - x$	erase x from v	remove x from the neighborhood of v if $v \in x^\star$	4	d
replace	$v^\star = v^\star - a + x$	replace a with x in v	substitute x for a in the neighborhood of v	5	e
splice after	$v^\star + x \succ a$	splice x after a in v	insert x immediately after a in the neighborhood of v	5	f
splice before	$v^\star + x \prec a$	splice x before a in v	insert x immediately before a in the neighborhood of v	5	g

1) returns the null vertex if $v^\star$ is empty.
2) returns the null vertex if $x \notin v^\star$.
3) not defined (error reported) if v appears in the list, or the same vertex in listed twice.
4) no effect if $v \notin x^\star$.
5) no effect if $v \notin a^\star$; not defined (error reported) if $x = v$ or $v \in x^\star$.

Table 2: Topological operations of the vertex-vertex algebra

attributes. For example, '$v$$pos denotes the position of vertex v at the time when the *synchronize* statement was last issued, whereas $v$$pos denotes the current position of v. Similarly, '$v^\star$ and $v^\star$ denote the old and current neighborhoods of v. The use of old attributes instead of the current ones makes it possible to iterate over the elements of a set in any order without affecting the iteration results.

3.3 Implementation of vertex-vertex systems

The software implementation of vertex-vertex systems is a set of programs and libraries collectively called the vv environment. The central component of this environment is vvlib, a C++ library containing data structures and functions implementing the vertex-vertex polygon mesh representation and algebra. The user can refer to these structures and functions directly from a program written in C++, or from a program written in the vv language.

The vv language extends C++ with keywords and expressions specific to the vertex-vertex algebra. In order to be executed, a vv program is first translated to a C++ program, with the keywords and expressions specific to vv translated into calls to the vvlib library. This C++ program is then compiled into a dynamically linked library (DLL). The modeling program, called vvinterpreter, loads this DLL, runs, and produces the graphical output. This whole processing sequence is automated: from the user's perspective, the vvinterpreter treats the vv program as an input and runs accordingly. The methodology that we have used to implement the vv language closely follows that developed for L+C, an extension of C++ with programing constructs based on L-systems [13].

4 Subdivision algorithms

To illustrate the usefulness of the vertex-vertex algebra, we provide compact descriptions of several subdivision algorithms. These descriptions are expressed in the vv language and can be directly executed by vvinterpreter.

4.1 Insertion of a Vertex

One particularly simple routine that also happens to be of mush use in writing subdivision algorithms is the insertion of a new vertex between two neighbouring vertices. So, we first define a function that creates a new vertex x and inserts it between two given vertices p and q (Algorithm 1).

```
1 vertex insert(vertex p, vertex q) {
2   vertex x;
3   make {p, q} nb_of x;
4   replace p with x in q;
5   replace q with x in p;
6   return x;
7 }
```

Algorithm 1: Code and illustration of the insertion of a vertex x between vertices p and q. Vertex x replaces p as the neighbor of q and q as the neighbor of p; vertices p and q become neighbors of x.

4.2 Polyhedral Subdivision

One of the simplest possible subdivision schemes is polyhedral subdivision [28] for triangular meshes. The scheme simply inserts a new vertex at the midpoint of each edge such that each triangle in the mesh is subdivided into four co-planar triangles. While the geometry of the polygon mesh does not change, the topology is subdivided.

The program (Algorithm 2) that implements polyhedral subdivision consists of two loops. The first loop (lines 5 to 12) considers iterates over the existing pairs of neighbouring vertices in the set S and inserts a new vertex between them (Figure 5a). The new vertices are added to the set NV and are assigned a position at the midpoint of the pair of vertices.

The second loop (lines 13 to 18) inserts new edges by redefining the neighborhoods of the new points. The intervening neighborhoods and the result of insertion are shown in Figure 5b,c.

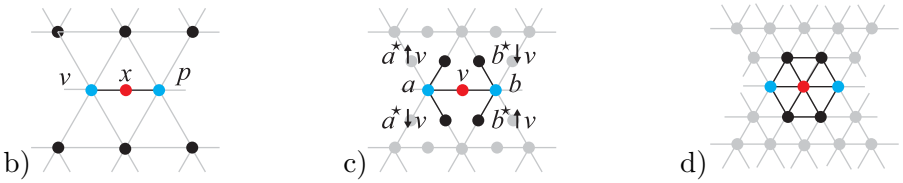


Figure 5: Illustration of the polyhedral subdivision algorithm implemented using vertex-vertex systems. a) The vv identification of points involved in the application of the mask to a new vertex x. b) The vv identification of vertices that will become neighbors of v. c) The mesh with all the new edges of v added.

```
 1 void polyhedral(mesh& S) {
 2   synchronize S;
 3   mesh NV;
 4
 5   forall v in S {
 6     forall p in ‘v {
 7       if (p < v) continue;
 8       vertex x = insert(v, p);
 9       x$pos = (p$pos + v$pos) / 2.0;
10       add x to NV;
11     }
12   }
13   forall v in NV {
14     vertex a = any in v;
15     vertex b = nextto a in v;
16     make {nextto v in b, b, prevto v in b,
17       nextto v in a, a, prevto v in a} nb_of v;
18   }
19   merge S with NV;
20 }
```

Algorithm 2: The polyhedral subdivision algorithm.

4.3 Loop algorithm

The Loop subdivision scheme [21] is topologically equivalent to the polyhedral subdivision scheme, in the sense that both operate on triangular meshes and subdivide a triangle into four triangles in an iteration step. The vertex-vertex implementations of both schemes have, therefore, a similar structure. The difference is in the placement of vertices. The Loop uses a mask to place new vertices and uses another mask to adjust the positions of old vertices (Figure 6). The implementation of the Loop subdivision algorithm is given by Algorithm 3.

4.4 Butterfly algorithm

The butterfly subdivision scheme for surfaces [7] is an interpolating scheme for triangular polygon meshes. The complete vv program that implements it for closed surfaces is given by Algorithm 4.

The algorithm for butterfly subdivision, like that for Loop subdivision,

```
 1 void loop(mesh& S) {
 2   double pi2 = 6.2832;
 3   synchronize S;
 4   mesh NV;
 5
 6   forall v in S {
 7     double n = valence v;
 8     double w = (0.625-pow(0.325 + 0.25*cos(pi2/n),2.0))/n;
 9     v$pos *= (1.0 - (double(n) * w));
10     forall p in ‘v {
11       v$pos += w * ‘p$pos;
12       if (p < v) continue;
13       vertex x = insert(v, p);
14       x$pos = 3.0/8.0 * ‘v$pos + 3.0/8.0 * ‘p$pos
15             + 1.0/8.0 * ‘(nextto p in ‘v)$pos
16             + 1.0/8.0 * ‘(prevto p in ‘v)$pos;
17       add x to NV;
18     }
19   }
20   forall v in NV {
21     vertex a = any in v;
22     vertex b = nextto a in v;
23     make {nextto v in b, b, prevto v in b,
24       nextto v in a, a, prevto v in a} nb_of v;
25   }
26   merge S with NV;
27 }
```

Algorithm 3: The Loop subdivision algorithm.

is topologicaly similar to the polyhedral subdivision. However, unlike Loop subdivision, the mask for the placement of new vertices requires the positions of vertices beyond the 1-ring. This mask and the corresponding vv identification of the intervening vertices are shown in Figure 7a,b.

4.5 $\sqrt{3}$ algorithm

Kobbelt's $\sqrt{3}$-subdivision [14] changes the topology of a triangular mesh in a manner different from the butterfly and Loop schemes (Figure 9). The corresponding vv implementation is given by Algorithm 5. In the first loop

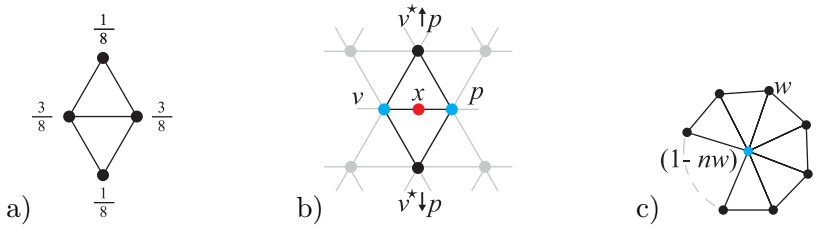


Figure 6: a) The Loop mask for a new vertex. b) The vv identification of points involved in the application of the mask to a new vertex x. c) The Loop mask for old vertices.

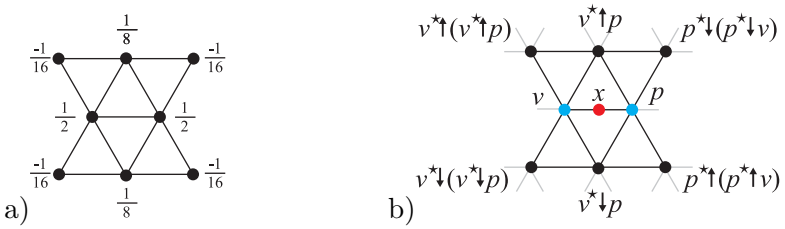


Figure 7: Illustration of the butterfly algorithm implemented using vertex-vertex systems. a) The mask. b) The vv identification of points involved in the application of the mask to a new vertex x.

of the algorithm, a new vertex c is created at the centroid of each triangle (lines 11 to 15). The neighborhoods are then updated such that each triangle is divided into three, that is each vertex v, x, y of the original triangle is connected to c, and the vertices v, x, y form the neighborhood of c (lines 16 to 19, c.f. Figure 9b). In the second loop (lines 23 to 31, Figure 9c), the topology is updated by flipping all the edges between pairs of old vertices.

5 Conclusions

We have addressed the problem of specifying polygon mesh algorithms in a concise and intuitive manner. To this end, we introduced a set of operations for locally changing the topology of a mesh, and we defined these operations in terms of relations between mesh elements. We have focused on subdivision algorithms as an application area, and we have shown that the resulting vertex-vertex algebra leads to a very compact and intuitive specifications of some of the best known algorithms.

```
 1 void butterfly(mesh& S) {
 2   double k = 1.0/16.0, l = 1.0/8.0, m = 1.0/2.0;
 3   synchronize S;
 4   mesh NV;
 5
 6   forall v in S {
 7     forall p in ‘v {
 8       if (p < v) continue;
 9       vertex x = insert(v, p);
10       x$pos = m * ‘v$pos + m * ‘p$pos
11         + l * ‘(prevto p in ‘v)$pos
12         + l * ‘(nextto p in ‘v)$pos
13         - k * ‘(nextto (nextto p in ‘v) in ‘v)$pos
14         - k * ‘(nextto (nextto v in ‘p) in ‘p)$pos
15         - k * ‘(prevto (prevto p in ‘v) in ‘v)$pos
16         - k * ‘(prevto (prevto v in ‘p) in ‘p)$pos;
17       add x to NV;
18     }
19   }
20   forall v in NV {
21     vertex a = any in v;
22     vertex b = nextto a in v;
23     make {nextto v in b, b, prevto v in b,
24       nextto v in a, a, prevto v in a} nb_of v;
25   }
26   merge S with NV;
27 }
```

Algorithm 4: The butterfly subdivision algorithm.

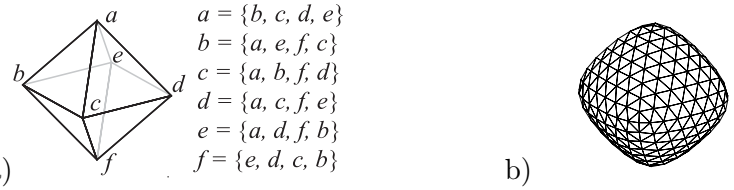


Figure 8: The butterfly algorithm in action. (a) An initial polyhedron and the vertex-vertex specification of its topology. (b) The polyhedron after three subdivision steps.

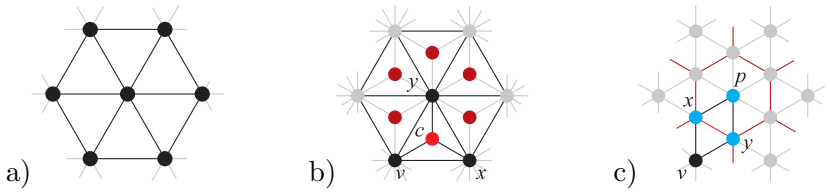


Figure 9: Mesh topology changes in the $\sqrt{3}$ scheme. a) A portion of the original mesh. b) The mesh after the insertion of central points, and subdivision of triangles. c) The mesh after the flip operation.

We have also designed vv, a programming language based on the vertex-vertex algebra, and we implemented a modeling environment in which vv programs can be executed. In addition to the subdivision algorithms described in this paper, we used vv to generate fractals and aperiodic tilings, simulate growth of multicellular biological structures, and create procedural textures on non-regular meshes. In these tests, we found vv programs extremely conducive to rapid prototyping and experimentation with polygon mesh algorithms.

Our implementation of the vertex-vertex algebra was guided by the elegance of programming constructs, rather than performance. For example, profiling of vv programs showed that approximately 50% of the algorithm execution time is spent on dynamic memory management. It is an interesting open question, if vertex-vertex systems could reach the speed of the fastest implementations of polygon mesh algorithms.

Another interesting class of problem is related to the temporal coordination of vertex-vertex operations. The synchronization mechanism introduced in Section 3.2.4 is in fact a method for simulating parallelism on a sequential machine. This suggests that it may be useful to extended vv with constructs for explicitly specifying parallel rather than sequential execution of operations. Such an extension could further clarify vv programs, and lead to their effective implementation on parallel processors with a suitable architecture.

References

[1] E. Akleman and J. Chen. Guaranteeing the 2-manifold property for meshes with doubly linked face list. *International Journal of Shape Modeling*, 5(2):149–177, 2000.

```
 1 void sqrt3(mesh& S) {
 2   synchronize S;
 3   mesh NV;
 4
 5   forall v in S {
 6     double pi2 = 6.28;
 7     double n = valence `v;
 8     double w = (4.0 - 2.0 * cos(pi2 / n)) / 9.0;
 9     v$pos *= (1.0 - w);
10     forall x in `v {
11       v$pos += `x$pos * w / n;
12       vertex y = nextto x in `v;
13       if (x < v || y < v) continue;
14       vertex c;
15       c$pos = (`v$pos + `x$pos + `y$pos) / 3.0;
16       make {v, x, y} nb_of c;
17       splice c after x in v;
18       splice c after y in x;
19       splice c after v in y;
20       add c to NV;
21     }
22   }
23   forall v in S {
24     forall p in `v {
25       if (p < v) continue;
26      vertex x = nextto p in v;
27       vertex y = prevto p in v;
28       splice y after v in x; splice x after p in y;
29       erase p from v; erase v from p;
30     }
31   }
32   merge S with NV;
33 }
```

Algorithm 5: The algorithm for $\sqrt{3}$ subdivision.

[2] E. Akleman, J. Chen, and V. Srinivasan. A new paradigm for changing topology during subdivision modeling. In *Pacific Graphics 2000*, pages 192–201, October 2000.

[3] E. Akleman, J. Chen, and V. Srinivasan. A prototype system for robust, interactive and user-friendly modeling of orientable 2-manifold meshes. In *Proceedings of Shape Modeling International 2002*, pages 43–50, May 2002.

[4] B. Baumgart. Winged edge polyhedron representation. Technical Report STAN-CS-320, Stanford University, 1972.

[5] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[6] G. Chaikin. An algorithm of high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

[7] N. Dyn, D. Levin, and J. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.

[8] J. Edmonds. A combinatorial representation of polyhedral surfaces (abstract). *Notices of the American Mathematical Society*, 7:646, 1960.

[9] R. Egli and N. F. Stewart. A framework for system specification using chains on cell complexes. *Computer-Aided Design*, 32:447–459, 2000.

[10] J.-L. Giavitto and O. Michel. Declarative definition of group indexed data structures and approximations of their domains. In *Proceedings of the 3rd ACM SIGPLAN Conference on Principles and Practice of Declarative Programming PPDP-01*, 2001.

[11] J.-L. Giavitto and O. Michel. MGS: A programming language for the transformation of topological collections. Research Report 61-2001, CNRS - Université d'Evry Val d'Esonne, Evry, France, 2001.

[12] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[13] R. Karwowski. *Improving the process of plant modeling: the L+C modeling language*. PhD thesis, University of Calgary, August 2002.

[14] L. Kobbelt. $\sqrt{3}$-subdivision. In *Computer Graphics*, 2000.

[15] W. Lickorish. Simplicial moves on complexes and manifolds. In *Proceedings of the Kirbyfest*, volume 2, pages 299–320, 1999.

[16] P. Lienhardt. Subdivisions de surfaces et cartes généralisées de dimension 2. *Informatique Théorique et Applications*, 25(2):171–202, 1991.

[17] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-aided Design*, 23(1):59–82, 1991.

[18] P. Lienhardt. Subdivision par opérations locales, 2001. Manuscript, Université de Poitiers, November 2001.

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

[20] A. Lindenmayer and G. Rozenberg. Parallel generation of maps: Developmental systems for cell layers. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Graph grammars and their application to computer science; First International Workshop*, Lecture Notes in Computer Science 73, pages 301–316. Springer-Verlag, Berlin, 1979.

[21] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, The University of Utah, August 1987.

[22] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988.

[23] R. Palmer and V. Shapiro. Chain models of physical behavior for engineering analysis and design. *Research in Engineering Design*, 5:161–184, 1993.

[24] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[25] P. Prusinkiewicz, F. Samavati, C. Smith, and R. Karwowski. L-system description of subdivision curves. Submitted, June 2002.

[26] G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation*. World Scientific, Singapore, 1997.

[27] M. Sabin. Subdivision surfaces, 2002. Shape Modeling International 2002 Tutorial Notes, 25 pp.

[28] E. Stollnitz, T. DeRose, and D. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufman Publishers, Inc., 1996.

[29] L. Velho. Stellar subdivision grammars. Submitted, January 2003.

[30] A. White. *Graphs, groups and surfaces*. North-Holland, Amsterdam, 1973.

[31] D. Zorin, P. Schröder, A. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for modeling and animation, 2000. SIGGRAPH 2000 Course Notes 23.

Design and implementation of the L+C modeling language

Radoslaw Karwowski and Przemyslaw Prusinkiewicz
Department of Computer Science
University of Calgary

Abstract

L-systems are parallel grammars that provide a theoretical foundation for a class of programs used in procedural image synthesis and simulation of plant development. In particular, the formalism of L-systems guides the construction of declarative languages for specifying input to these programs. We outline key factors that have motivated the development of L-system-based languages in the past, and introduce a new language, L+C, that addresses the shortcomings of its predecessors. We also describe the implementation of L+C, in which an existing language, C++, was extended with constructs specific to L-systems. This implementation methodology made it possible to develop a powerful modeling system in a relatively short period of time.

1. Background

L-systems were conceived as a rule-based formalism for reasoning on developing multicellular organisms that form linear or branching filaments [Lindenmayer, 1968]. Soon after their introduction, L-systems also began to be used as a foundation of visual modeling and simulation programs, and computer languages for specifying the models [Baker and Herman, 1972]. Subsequently they also found applications in the generation of fractals [Szilard and Quinton, 1979; Prusinkiewicz, 1986] and geometric modeling [Prusinkiewicz et al., 2003]. A common factor uniting these diverse applications is the treatment of structure and form as a result of development. A historical perspective of the L-system-based software and its practical applications is presented in [Prusinkiewicz, 1997].

According to the L-system approach, a developing structure is represented by a string of symbols over a predefined alphabet V. These symbols represent different components of the structure (e.g., points and lines of a geometric figure, cells of a bacterium, apices and internodes of a plant). The process of development is characterized in a declarative manner using a set of productions over the alphabet V. During the simulation of development, these productions are applied in parallel steps to all symbols of the string, thus capturing the development in discrete time slices.

Lindenmayer [1971] observed that L-system productions can be specified using standard notation of formal language theory. In the simplest, context-free case, productions have the form:

$$predecessor \rightarrow successor$$

where *predecessor* is a letter of alphabet V and *successor* is a (possibly empty) word over V. For example, the division of a cell A into cells B and C can be written as $A \rightarrow BC$. In the context-sensitive case, productions are often written as

$$lc < predecessor > rc \rightarrow successor\ ,$$

where symbols < and > separate the strict predecessor from the left context *lc* and the right context *rc* [Prusinkiewicz and Hanan,1989]. Both contexts are words over *V*. For example, the production pair:

$$Y < A > O \rightarrow LYS$$
$$O < A > Y \rightarrow SYL$$

describes asymmetric division of a mother cell *A* into a short daughter cell *S* and long daughter cell *L*, separated by a cell wall *Y*. The sequence of these cells in the filament is guided by the state of the walls that delimit the mother cell, which may be young (*Y*) or old (*O*). Obviously, a complete description of the filament's development would also require productions that characterize the growth of cells and walls over time.

Early L-system-based programming languages closely followed the above notation [Baker and Herman, 1982; Prusinkiewicz and Hanan, 1989]. The needs for expressing increasingly complex models led, however, to the addition of constructs found in other programming languages. A pivotal moment in this evolution was the introduction of parametric L-systems [Prusinkiewicz and Hanan, 1990; Hanan, 1992] and related constructs [Chien and Jurgensen, 1992], which associated numerical attributes to L-system symbols, similar to those found in attribute grammars [Knuth, 1968]. This created a need for calculating new parameter values (in the production successor) on the basis of old ones (found in the predecessor and its context). According to the original definition of parametric L-systems [Prusinkiewicz and Hanan, 1990; Hanan, 1992], these calculations were specified as arithmetic operations on the argument parameters, e.g.

$$A(x) < B(y) > C(z) \rightarrow D(x+y)\ E(y+z)\ .$$

In modeling practice, however, entire procedures soon became needed to calculate new parameter values. Recognizing this need, Hanan [1992] introduced the following syntax for L-system productions:

$$lc < predecessor > rc\ \{\alpha\} : cond\ \{\beta\} \rightarrow successor\ .$$

Here α and β are C-like compound statements, and *cond* is a logical expression that guards production application. A production is applied in stages. First, it is determined whether production predecessor *pred*, surrounded by the left context *lc* and the right context *rc*, matches the given symbol in the string. If this is the case, the compound statement α is executed, and condition *cond* is evaluated. If the result of this evaluation is non-zero ('true'), the second compound statement β is executed. On this basis, parameters values in the production successor are then determined, and the successor is inserted into the resulting string. For example, the following is a valid production:

$$A(x) < B(y) > C(z)\ \{r = x*x + y*y + z*z;\} : r > 2\ \{t = x+y+z;\} \rightarrow D(t)\ E(2*t).$$

At the top level, an L-system with productions in the above form operates in a declarative fashion, by rewriting elements of a string according to their type, context, and the associated parameters. Within each production, however, calculations are performed sequentially, using constructs borrowed from an imperative language. This combination of paradigms suggests two strategies for translating L-system-based languages into a representation directly used by simulation programs [Prusinkiewicz and Hanan, 1992]:

- extend the formal notation for productions with constructs borrowed from an imperative language, or
- extend an existing imperative language with constructs inherent in L-systems.

The modeling program cpfg [Hanan, 1992] and its modeling language [Prusinkiewicz et al., 2000] are representative of the first approach. The interpreter of the cpfg language was constructed following the standard steps of lexical analysis, parsing, and object code generation. Nevertheless, in spite of well-developed methodology for translator construction (e.g. [Aho et al, 1986]), construction of a compiler for a comprehensive language is a large task. Consequently, the cpfg language only includes a limited subset of C-like statements; for example, it does not support user-definable functions and typed parameters associated with the modules. As a result, while simple L-system models can be expressed using cpfg language in an elegant, compact manner, specification and maintenance of larger models becomes difficult.

An alternative approach, first suggested in [Prusinkiewicz and Hannan 1992], is to create an L-system-based programming environment by extending an existing language with support (classes, libraries) specific to L-systems. Using this approach, Hammel [1996] implemented differential L-systems [Prusinkiewicz et al., 1993] in SIMULA, and Erstad [2002] implemented an L-system-based programming environment in LISP. Both implementations preserve the syntax of the underlying languages (SIMULA and LISP). In contrast, Karwowski [2002] implemented the L-system-based programming language L+C by extending the syntax of C++ [Sievanen]. We describe here the design and implementation of this language.

2. The L+C modeling language

The key new elements introduced in the L+C modeling language are:

- typed module parameters, including all primitive and compound data types (structures) supported by C++
- productions with multiple successors
- extension of the notion of context-sensitivity with the 'new context' constructs, which speed up information transfer across simulated structures.

In addition, by virtue of being based on the C++ language, L+C has the full expressive power of C++. In particular, user-defined functions are supported as in C++.

At the top level, an L+C program is a set of declarations for:

- Structures and classes,
- Global variables,
- Functions,
- Modules,
- The axiom,
- The derivation length,
- Productions,
- Decomposition rules,
- Interpretation rules,
- Control statements.

The declarations of structures, classes, variables and functions have exactly the same syntax and meaning as in C++. The remaining declarations are specific to L+C, and are described below.

2.1. Module declarations

Modules are the elements of the L-system string. A module consist of an identifier (which must follow the C++ syntax [Stroustroup, 1991]) and an optional list of parameters. In L+C modules have to be declared before they can be used. Declaration specifies the number and types of parameters that are associated with the given module type using the following syntax:

`module` *identifier* (*parameter-list*$_{opt}$);

Examples of valid module declarations are:

```
module A(); // module A with no parameters
module N(float); // module N with one parameter of type float
module Metamer(int, MetamerData); // module Metamer with a
                        // parameter of type int and
                        // a user-defined type MetamerData
```

2.2. Axiom declaration

The axiom declaration specifies the initial L-system string using the following syntax:

`axiom`: *parametric-string*;

where the *parametric-string* must be non-empty. Assuming that the modules have been declared as in Section 2.1, and `s_init` is a structure of type `MetamerData`, the following is a valid axiom declaration:

```
axiom: Metamer(1,s_init) N(0.25) A();
```

2.3. Derivation length specification

Derivation length is the number of derivation steps for the simulation. It is specified using the syntax:

`derivation length`: *integer-expression*;

2.4. Specification of productions

The syntax of productions is a combination of the formal L-system notation and the C++ syntax for function definition. In general, it has the syntax:

predecessor:
{
    production body
}

The predecessor has one of the following forms:

new-left-context << *left-context* < *strict-predecessor* > *right-context* :
left-context < *strict-predecessor* > *right-context* >> *new-right context*:

The strict predecessor specifies the part of the string being rewritten by the production. It can be a single module, as assumed in the usual definition of L-systems, or a string of several modules, as defined for pseudo-L-systems [Prusinkiewicz, 1986]. The optional left and right contexts are strings of modules that need to be in the neighborhood of the strict predecessor in order for the production to apply. The new contexts specify the modules that must be present in the neighborhood of the production successor, in the string being derived. This information is easily available if the string is being rewritten in a particular direction: from left to right in the case of new left context, and from right to left in the case of new right context (Figure 1). In theory, two-sided new context could also be defined, but its implementation is more difficult and, therefore, it is not supported by L+C.

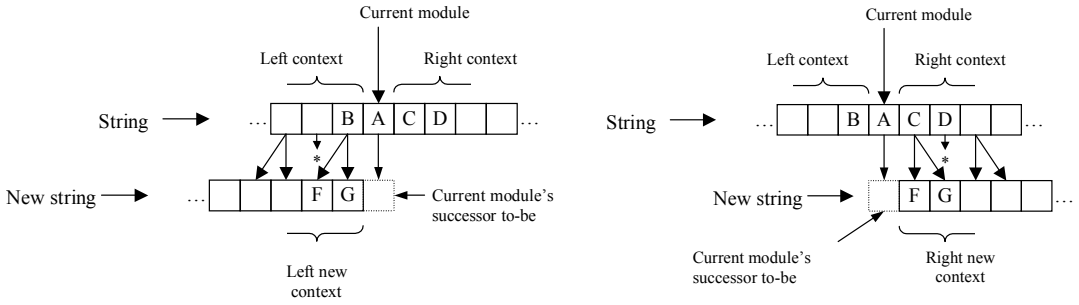


Figure 1. Context of L-system productions. Left new context is available if the successor string is built left-to-right (left figure). Right new context is available if the successor string is built right to left (right figure).

The parameters that appear in the production predecessor are formal parameters. All the formal parameters of every module in a production predecessor must be listed, even if they are not used in the production body. An example of a valid production predecessor that uses the modules declared in Section 2.1 is:

```
Metamer(i_l, d_l) N(w) < Metamer(i, d) > A()
```

Formal parameters have types determined by the declarations of the respective modules. They are bound to the actual parameters in the string during production application [Prusinkiewicz and Hanan, 1990]. The scope of the formal parameters is the same as the scope of formal parameters in C++ functions.

The production body is a compound statement that may contain any code allowed inside a C++ function. In addition, the production body may include one or more `produce` statements, which specify possible successors of the production. The `produce` statement has the syntax:

`produce` *parameteric-string*$_{opt}$;

where *parameteric-string* is defined as in the axiom (Section 2.2). Each `produce` statement is implicitly followed by a `return` statement. Thus, if several produce statements are present in the production body, the first statement executed terminates the production application. Typically, the choice of alternative successors is controlled by C++ conditional statements.

2.5. Decomposition rules

As defined by Lindenmayer [1968], L-systems operate in discrete derivation steps. Each step consists of a (conceptually) parallel application of suitable productions to all symbols in the predecessor string. This parallelism is intended to capture progression of time by a given interval, the same for all components of the modeled structure. Thus, for example, the L-system production $A \rightarrow BC$ expresses the idea "module *A* develops into modules *B* and *C* over a given time interval." In practice, it is also often necessary to express the idea that a given module is a compound module, consisting of several elements. A logical analysis of the notions "develops over time" and "consists of" was presented by Woodeger [1937]. Prusinkiewicz et al. [2000, 2001] showed that, in a grammar setting, these notions correspond to L-system productions and Chomsky context-free productions, respectively. In L+C, Chomsky productions are called decomposition rules. They are specified using the same syntax as context-free L-system productions, and are identified using the keyword `decomposition`, as in the following example:

```
decomposition:
Metamer(i, d) : { produce Internode(i, d) Leaf(d) Bud();}
```

This production characterizes a `Metamer` as a compound module consisting of an `Internode`, a `Leaf`, and a `Bud`. Obviously, all modules must have been declared earlier in the L+C program.

The integration of decomposition rules into the L-system framework affects the way in which a derivation step is performed [Prusinkiewicz et al., 2000]. In L+C, decomposition rules are applied recursively, after the definition of the initial string by the `axiom` statement (Section 2.2) and after each step of standard L-system production applications (Section 2.4).

2.6. Interpretation rules

Structures generated with L-systems may be visualized by assigning a graphical interpretation to a predefined set of modules [Szilard and Quinton, 1979; Prusinkiewicz, 1986, Prusinkiewicz et al., 2003]. For example, in L+C, a predefined module `F(float)` draws a line of a given length in the current direction (as defined in the turtle geometry [Abelson and diSessa, 1982]); `Line2D (point2D, point2D)` draws a line between two given points, and `SetColor(int)` assigns a color to geometric primitives. From the user perspective, however, it is often more convenient to express the model in terms of modules inherent in the modeling domain (e.g., apices, internodes, and leaves in the case of plant models) rather than directly in terms of modules with a geometric interpretation (e.g., points, lines, and polygons). In order to separate these conceptual and visual aspects of model specification, Kurth [1994] introduced the notion of interpretation rules. Interpretation rules are similar to decomposition rules in that they are context-free Chomsky productions, and are applied recursively, after each derivations step (specifically, after the decomposition rules have been applied). In contrast to decomposition rules, however, interpretation rules do not affect the outcome of the following derivation steps. Instead, they are applied "on the side", producing modules that are passed to the graphical part of the modeling program, and discarded once they have been interpreted (Figure 2).

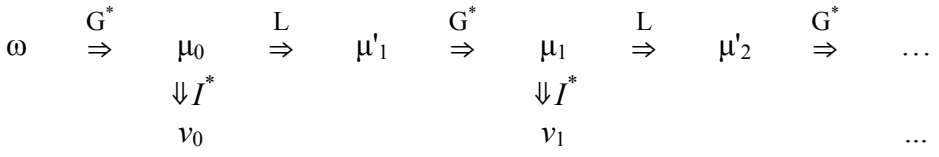


Figure 2. Generation of a developmental sequence using an L-system with decomposition and interpretation rules. Beginning with the axiom ω, the progressions of strings μ_1, μ_2, μ_3,… results from the interleaved application of decomposition rules G and L-system derivation steps L. The interpretation rules I map strings μ_i into strings ν_i , which are interpreted graphically.

In L+C, interpretation rules are identified using the keyword `interpretation`, as in the following example:

```
interpretation:
Internode(i, d) : { produce SetColor(1) F(d.length); }
```

The above production specifies that module `Internode` will be represented graphically as a straight line, (`F`) drawn using color with index `1`. The line length is specified by field `length` in data structure `d`.

2.7. Control statements

Control statements were introduced by Hanan [1992] (see also Prusinkiewicz et al., 2000]) to specify procedures that are executed at specific points during an L-system-based derivation. In L+C, they are specified using the syntax:

```
Start|StartEach|EndEach|End:
{
        compound statement
}
```

The control statements are executed as follows:

- `Start` is executed at the beginning of the program,
- `StartEach` is executed before every derivation step,
- `EndEach` is executed after every derivation step,
- `End` is executed after the last derivation step.

Any code that is allowed inside a C++ function can be specified as the *compound statement*. Typical uses of the control statements include initialization of global variables, opening and closing of I/O streams, and reporting of simulation statistics after each simulation step.

2.8. Example

A sample L+C program that generates a branching structure is presented below:

```
1  #include <lpfgall.h>
2  #include <math.h>
3
4  const int Delay = 1;
5  const float BranchingAngle = 45.0;
6  const float LengthGrowthRate = 1.33;
7
8  derivation length: 17;
```

```
 9
10  struct InternodeData
11  { float length, area; };
12
13  module A(int,float);
14  module Metamer(float);
15  module Internode(InternodeData);
16
17  Start: { Backward(); }
18  ignore: Right;
19
20  axiom: A(0,BranchingAngle);
21
22  A(t,angle) :
23  {
24    if (t<0) // young apex
25      produce A(t+1,angle);
26    else     // mature apex
27      produce Metamer(angle) A(0,-angle);
28  }
29
30  Internode(id) >> SB() Internode(id2) EB() Internode(id3) :
31  {
32    id.area = id2.area + id3.area;
33    id.length *= LengthGrowthRate;
34    produce Internode(id);
35  }
36
37  Internode(id) >> Internode(idr) :
38  {
39    id.area = idr.area;
40    id.length *= LengthGrowthRate;
41    produce Internode(id);
42  }
43
44  Internode(id) >> A(t,angle):
45  {
46    id.length *= LengthGrowthRate;
47    produce Internode(id);
48  }
49
50  decomposition:
51  Metamer(angle) :
52  {
53    InternodeData id = {1, 1};
54    produce
55        Internode(id)
56        SB() Right(angle) A(-Delay,angle) EB()
57        Internode(id);
58  }
59
60  interpretation:
61  Internode(id) :
62  {
63    produce SetColor(2) SetWidth(pow(id.area,.5)) F(id.length);
64  }
```

The modeled structure consists of three types of modules, which are given biologicaly meaningful names `A`, `Metamer`, and `Internode` (lines 13-15). The process of string derivation is performed backward (from right to left) as indicated in the Start statement (line 17). In the process of context matching module Right (used to specify the branching angle in line 56) is ignored (line 18). The initial structure defined by the axiom is a single

apex. Its parameters characterize the developmental stage and the branching angle of the next branch that will be produced by this apex. According to the first production (lines 22-28), an immature apex will grow older, and a mature apex will produce a metamer, over the time interval associated with a derivation step. The decomposition rule (lines 51-58) specifies that the metamer consists of two internode segments, and a lateral branch delimited by the language-predefined modules `SB()` (start branch) and `EB()` (end branch). The branch initially consists of a lateral apex, placed at a given `angle` with respect to its supporting internode. The development of internodes is described by the three productions in lines 30 to 48. These productions specifies that an internode will grow in length by factor `LengthGrowthRate` per derivation step. They also determine the cross-section area of each internode as the sum of the cross-sections of internodes supported by it. Specifically, the new context construct is used to accumulate the cross-section of branches when moving from the apices toward the base of the structure. Finally, the interpretation rule (lines 61-64) specifies that each internode will be visualized as a line of length and width determined by the internode parameters. The structure generated by this L-system is shown in Figure 3.

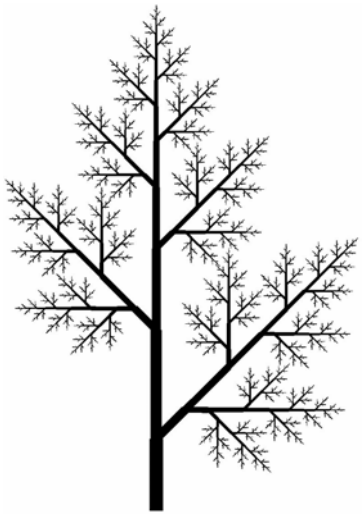


Figure 3. Example of a structure generated by the sample L-system.

3. Implementation of the L+C translator

The main difference between L+C and C++ is not at the level of syntax, but at the level of the programming paradigm: L+C is a declarative language, whereas C++ is an imperative language. Furthermore, L+C programs operate in a specific topological space [Giavitto and Michel, 2001, 2002] of a linear or branching string, whereas C++ does not presuppose any such space. Despite these differences, most of the L+C grammar is the C++ grammar. Given that, the process of compiling and executing an L+C program consists of translating some specific L+C constructs into C++, while leaving other constructs are left intact. This leads to the modeling system design shown in Figure 4.

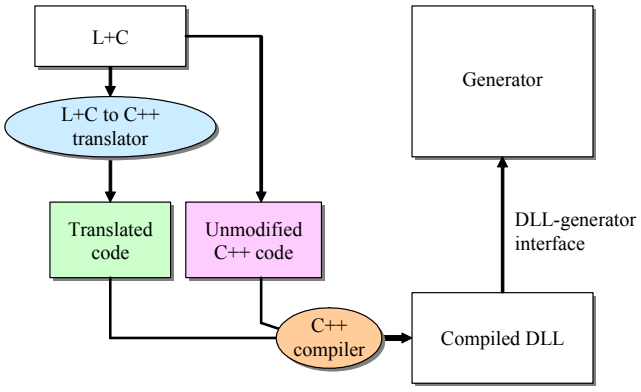


Figure 4. Components of our modeling system

Based on this design, the translator divides the input L+C code into two categories: the constructs specific to L+C are translated into C++ code, while the remaining C++ code is passed verbatim to the compiler. The resulting C++ code is then compiled using a standard C++ compiled as a DLL (dynamic link library). The actual execution of the L+C program is performed by a fixed component of a modeling program, called the generator (Figure 4). For the user's convenience, modified L+C program can be translated, compiled and ran without a need for restarting the modeling program. Consequently, the L-system string derivation is performed based only on the information that can be provided by the DLL at run-time (since the generator is a fixed component and is not recompiled for every L+C program).

The DLL includes the interfacing information that makes the generator and the compiled L+C program communicate. We present this interface from the perspective of string derivation by the generator. The core of the generator is the `Execute()` function:

```
void Execute()
{
   Start();
   Axiom();
   DecomposeString();
   for (int i=0; i<DerivationLength(); ++i)
   {
      StartEach();
      Derive();
      DecomposeString();
      EndEach();
   }
   End();
}
```

where the functions written in boldface are defined in the process of translating the L+C program to C++ as follows:

- `Start()`, `StartEach()`, `EndEach()` and `End()` execute the compound statements specified in the corresponding L+C control statements (Section 2.7);
- `Axiom()` creates the initial L-system string (Section 2.2),
- `DerivationLength()` returns the value specified in the L-system `derivation length` statement (Section 2.3).

The translation of the L+C control statements into C++ functions is straightforward. For example, the L+C `Start` statement is translated as follows:

Original code

```
Start:
{
  …
}
```

Translated code

```
void Start()
{
  …
}
```

Analogous substitutions are made for the other L+C control statements. To process the `derivation length` statement, the translator replaces the L+C keyword with a C++ function prototype:

Original code

```
derivation length: 3;
```

Translated code

```
int DerivationLength() { return 3; }
```

In order to present the translation of productions, let us consider the following L+C production as an example:

```
module A(data, float);
module B(int, float);

A(dl, xl) < B(n, a) :
{
   if (a>xl)
      produce B(n+1, xl);
   else
      produce B(n-1, xl);
}
```

Elements of the production typical for L+C are highlighted in boldface. The process of translation is based on the fact that productions are similar to functions in imperative programming languages. The similarities can be summarized into the following:

- A production is a piece of code to be executed,
- Its input is its predecessor and optionally, parameters of the predecessor's modules, and
- Its output is the successor.

The differences between productions and functions are as follows:

- L-system programs do not call productions explicitly. The general mechanism of matching productions determines which production should be applied and when.
- Productions do not return a value in the traditional sense. Instead, their output modifies the contents of the L-system string.

The first step in translating a production into a C++ function is to declare a function prototype, using the types declared in the relevant modules. For example, the following substitution is made:

Original code:	Translated code:
`A(dl, xl) < B(n, a)`	`void P1(data dl, float xl, int n, float a)`

Another element in the production code that needs to be translated is the produce statement. The code resulting from the translation of this statement must add the successor to the new string, and terminate the production. In our example, the produce statement is translated into code similar to this:

Original code:	Translated code:
`produce B(n+1, x);`	`{App(B_id); App(n+1); App(x); return;}`

It should be noted that the translation process, as so far described, does not retain all the necessary information. In particular, the modules in the strict predecessor and the context information are not present in the generated code. It is then necessary to add information that bridges the generator and the translated L+C code. However, as this is of a purely technical concern of program implementation, the additional code is not further discussed here.

4. Conclusions

We have described a modeling language L+C, which incorporates C++ into the framework of L-systems. We have also implemented a modeling system that uses L+C programs as input. To implement the L+C translator, we have introduced a methodology based on the separation of the constructs specific to L-systems from the C++ code. This methodology made it possible for a single person to implement the L+C translator in one month. The L-system-specific code is translated into C++ and combined with the C++ code taken verbatim from the L+C programs. The resulting code is translated into a DLL module using a standard C++ compiler. This module is linked with the generator that executes the L-systems. In practice, the DLL module is small in size compared to the generator and the graphical interpreter associated with it. Consequently, the DLL module compiles and links fast (of the order of one second on the current Windows and Linux workstations), which allows for interactive manipulation and modification of the models. The increased expressiveness of L+C, compared to the previous L-system based languages, makes it possible to create models of a relatively greater complexity. L+C is currently being used to model aspects of plant genetics, physiology, and biomechanics.

References

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

Aho 1986: Aho, A., Sethi, R. and Ullman, J. [1986], *Compilers: Principles, techniques and tools*. Addison-Wesley, Reading.

Baker R. and Herman G. T. [1970]: Simulation of organisms using a developmental model, parts I and II. *International Journal of Bio-Medical Computing* **3**, pp. 201-215 and 251-267.

Chien, T. and Jurgensen, H. [1992]: Parameterized L systems for modelling: Potential and limitations. In: G. Rozenberg and A. Salomaa (Eds.): *Lindenmayer systems: Im-*

pacts on theoretical computer science, computer graphics, and developmental biology. Springer, Berlin, pp. 213—229.

Erstad, K. [2002]: *L-systems, twining plants, Lisp*. M. Sc. thesis, University of Bergen.

Giavitto, J.-L. and Michel, O. [2001]: *MGS: A programming language for the transformation of topological collections*. Research Report, 61-2001, CNRS – Universite d'Evry Val d'Esonne.

Giavitto, J.-L. and Michel, O. [2002]: Data structures as topological spaces. Proceedings of the 3rd International Conference on Unconventional Models of Computation UMC02, *Lecture Notes in Computer Science* **2509**, pp. 137-150.

Hammel, M. [1996]: *Differential L-systems and their application to the simulation and visualization of plant development*. Ph. D. thesis, University of Calgary.

Hanan, J. [1992]: *Parametric L-systems*. Ph. D. thesis, University of Regina.

Karwowski, R. [2002]: *Improving the process of plant modeling: The L+C modeling language*. Ph. D. thesis, University of Calgary.

Knuth, D. [1968]: Semantics of context-free languages. *Mathematical Systems Theory* **2**, pp. 191-220.

Kurth, W. [1994]: *Growth grammar interpreter (GROGRA 2.4): A software tool for the 3-dimensional interpretation of stochastic, sensitive growth grammars in the context of plant modeling. Introduction and reference manual*. Forschungszentrum Waldokosysteme der Universitat Gottingen.

Lindenmayer, A. [1968]: Mathematical models for cellular interaction in development. *Journal of Theoretical Biology* **18**, pp. 280-315.

Lindenmayer, A. [1971]: Developmental systems without cellular interaction, their languages and grammars. *Journal of Theoretical Biology* **30**, pp. 455-494

Prusinkiewicz, P. [1986]: Graphical applications of L-systems. Proceedings of Graphics Interface '86 – Vision Interface '86, pp. 247-253.

Prusinkiewicz, P. and Hanan, J. [1989]: *Lindenmayer systems, fractals and plants*. Lecture Notes in Biomathematics **79**, Springer, Berlin.

Prusinkiewicz, P. and Hanan, J. [1990]: Visualization of botanical structures and processes using parametric L-systems. In: D. Thalmann (Ed.), *Scientific visualization and graphics simulation*, J. Wiley & Sons, Chichester, pp. 183-201.

Prusinkiewicz, P. and Hanan, J. [1992]: L-systems: From formalism to programming language*s*. In: G. Rozenberg and A. Salomaa (Eds.), *Lindenmayer systems: Impacts on theoretical computer science, computer graphics and developmental biology*. Springer, Berlin, pp. 193-211.

P. Prusinkiewicz, P., Hammel, M. and Mjolsness, E. [1993]: Animation of plant development. Proceedings of SIGGRAPH 93, pp. 351-360.

Prusinkiewicz, P. [1997]: A look at the visual modeling of plants using {{L}-systems. In R. Hofestadt and T. Lengauer and M. Loffler and D. Schomburg (Eds.): Bioinformatics. *Lecture Notes in Computer Science* **1278**, Springer, Berlin, pp.11-29.

Prusinkiewicz, P., Hanan, J., and Mech, R. [2000]: An L-system-based plant modeling language. In M. Nagl, A. Schuerr and M. Muench (Eds.): Applications of graph transformation with industrial relevance. *Lecture Notes in Computer Science* **1779**, Springer, Berlin, pp. 395-410.

Prusinkiewicz, P., Muendermann, L., Karwowski, R. and Lane, B. [2001]: The use of positional information in the modeling of plants. Proceedings of SIGGRAPH 2001, pp. 289-300.

Prusinkiewicz, P., Samavati, F., Smith, C. and Karwowski, R. [2003]: L-system description of subdivision curves. To appear in the *International Journal of Shape Modeling*.

Sievanen R., Perttunen J., Prusinkiewicz P., Karwowski R., Modeling language L, unpublished report.

Stroustroup, B. [1991]: *The C++ Programming Language*, Addison-Wesley, Reading.

Szilard, A. and Quinton, R. [1979]: An interpretation for D0L systems by computer graphics. *The Science Terrapin* **4**, pp. 8-13.

Woodger J. [1937]: *The axiomatic method in biology*, University Press, Cambridge.

Generating subdivision curves with L-systems on a GPU*

Radomír Měch†
SGI

Przemyslaw Prusinkiewicz†
University of Calgary

Abstract

The introduction of floating-point pixel shaders has initiated a trend of moving algorithms from CPUs to graphics cards. The first algorithms were in the rendering domain, but recently we have witnessed increased interest in modeling algorithms as well.

In this paper we present techniques for generating subdivision curves on a modern Graphics Processing Unit (GPU). We use an existing method for generating subdivision curves with L-systems, we extend these L-systems to implement adaptive subdivision, and we show how these L-systems can be implemented on a GPU.

We chose L-systems because they can express many modeling algorithms in a compact way and are parallel in nature, making them an attractive paradigm for programming a GPU.

1 Introduction

In recent years subdivision curves became an important alternative to parametric curves in computer aided design. For a modeler they are very attractive because a complex curve can be defined using a small number of control points.

The new programmable graphics hardware with capabilities of executing a set of instructions during the vertex or fragment processing has proven to be capable of solving difficult processing tasks. Various algorithms have been implemented on these Graphics Processing Units (GPUs), ranging from ray-tracing [6] to solving differential equations [2]. Considering the parallel nature of subdivision algorithms the new graphics hardware is a suitable candidate for implementing them.

In this note we review L-systems that capture different subdivision scheme, we extend the L-systems presented in [5] with support for adaptive subdivision of curves, and we show how to implement these L-systems on a GPU[1].

GPUs can be programmed using assembler level languages or higher level languages, such as Cg [3] or Direct X 9.0 HLSL[2]. We chose to implement L-systems using the assembler level language.

2 Generating subdivision curves

Subdivision curves can be described using context-sensitive parametric L-systems [5]. Control points of the curve are stored as symbols in the initial string, with parameters specifying point locations[3]. L-system productions are used to replace each point with new points according to a given subdivision scheme. For example, Chaikin subdivision of a closed curve is captured by a single production [5],

L-system 1:

$$P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\tfrac{1}{4}\mathbf{v}_l + \tfrac{3}{4}\mathbf{v})P(\tfrac{3}{4}\mathbf{v} + \tfrac{1}{4}\mathbf{v}_r),$$

which replaces one point, the strict predecessor, with two new points, forming the successor. The location of each new point is an affine combination of the locations v, v_l and v_r of the predecessor point and its context (neighbors).

It is easy to modify L-system 1 to express different subdivision schemes. Each scheme is using different affine combination of the neighbors. Some schemes presented in [5] are using more than one neighbor on each side of the point, but not more than two. Thus we can combine these L-systems in a single scheme:

L-system 2:

$$P(\mathbf{v}_0)P(\mathbf{v}_1) < P(\mathbf{v}_2) > P(\mathbf{v}_3)P(\mathbf{v}_4) \rightarrow P(\textstyle\sum_{i=0}^{4} a[i].\mathbf{v}_i)P(\sum_{i=0}^{4} b[i].\mathbf{v}_i),$$

where arrays a and b store parameters of the affine combination for each new symbol. L-system 2 can express Chaikin subdivision scheme using values $a = \{0, \frac{1}{4}, \frac{3}{4}, 0, 0\}$ and $b = \{0, 0, \frac{3}{4}, \frac{1}{4}, 0\}$, cubic B-spline subdivision using $a = \{0, \frac{1}{8}, \frac{3}{4}, \frac{1}{8}, 0\}$ and $b = \{0, 0, \frac{1}{2}, \frac{1}{2}, 0\}$, and Dyn-Levin-Gregory (4-point) subdivision using $a = \{0, 0, 1, 0, 0\}$ and $b = \{0, -\frac{1}{16}, \frac{9}{16}, \frac{9}{16}, -\frac{1}{16}\}$.

In the case of open subdivision curves, end points of the curve do not change location and the rules for creating new points in their neighborhood are different from those operating farther from the endpoints. If we denote the endpoints by symbol E, we can expand L-system 1 to open curves as follows [5]:

L-system 3:

$$p_1\text{: } E(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\tfrac{1}{2}\mathbf{v}_l + \tfrac{1}{2}\mathbf{v})P(\tfrac{3}{4}\mathbf{v} + \tfrac{1}{4}\mathbf{v}_r)$$
$$p_2\text{: } P(\mathbf{v}_l) < P(\mathbf{v}) > E(\mathbf{v}_r) \rightarrow P(\tfrac{1}{4}\mathbf{v}_l + \tfrac{3}{4}\mathbf{v})P(\tfrac{1}{2}\mathbf{v} + \tfrac{1}{2}\mathbf{v}_r)$$
$$p_3\text{: } P(\mathbf{v}_l) < P(\mathbf{v}) > P(\mathbf{v}_r) \rightarrow P(\tfrac{1}{4}\mathbf{v}_l + \tfrac{3}{4}\mathbf{v})P(\tfrac{3}{4}\mathbf{v} + \tfrac{1}{4}\mathbf{v}_r)$$
$$p_4\text{: } E(\mathbf{v}) \rightarrow E(\mathbf{v})$$

L-system 3 can be generalized in a similar manner to L-system 1. To this end, we extend L-system 3 with two new productions, in which the symbol E is two symbols away

*This is an extended version of a sketch to be presented at SIGGRAPH 2003.

†rmech@sgi.com, pwp@cpsc.ucalgary.ca

[1]Our implementation and description are based on the ATI Radeon 9700 card (http://www.ati.com/developer).

[2]http://msdn.microsoft.com/directx/

[3]We make here a distinction between the *location* of a point (three coordinates) and its *position* in the string (an index value).

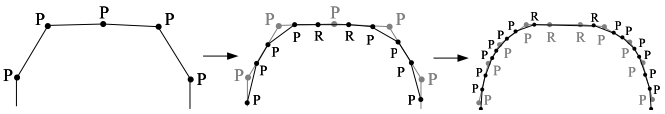


Figure 1: Operation of L-system 5. Points of type 0 are marked as R.

from the predecessor, and we define arrays a and b for each production:

L-system 4:

p_1: $E(\mathbf{v}_0)P(\mathbf{v}_1) < P(\mathbf{v}_2) > P(\mathbf{v}_3)P(\mathbf{v}_4)$
$\rightarrow P(\sum_{i=0}^{4} a[0][i].\mathbf{v}_i)P(\sum_{i=0}^{4} b[0][i].\mathbf{v}_i)$

p_2: $E(\mathbf{v}_1) < P(\mathbf{v}_2) > P(\mathbf{v}_3)P(\mathbf{v}_4)$
$\rightarrow P(\sum_{i=1}^{4} a[1][i].\mathbf{v}_i)P(\sum_{i=1}^{4} b[1][i].\mathbf{v}_i)$

p_3: $P(\mathbf{v}_0)P(\mathbf{v}_1) < P(\mathbf{v}_2) > P(\mathbf{v}_3)E(\mathbf{v}_4)$
$\rightarrow P(\sum_{i=0}^{4} a[2][i].\mathbf{v}_i)P(\sum_{i=0}^{4} b[2][i].\mathbf{v}_i)$

p_4: $P(\mathbf{v}_0)P(\mathbf{v}_1) < P(\mathbf{v}_2) > E(\mathbf{v}_3)$
$\rightarrow P(\sum_{i=0}^{3} a[3][i].\mathbf{v}_i)P(\sum_{i=0}^{3} b[3][i].\mathbf{v}_i)$

p_5: $P(\mathbf{v}_0)P(\mathbf{v}_1) < P(\mathbf{v}_2) > P(\mathbf{v}_3)P(\mathbf{v}_4)$
$\rightarrow P(\sum_{i=0}^{4} a[4][i].\mathbf{v}_i)P(\sum_{i=0}^{4} b[4][i].\mathbf{v}_i)$

p_6: $E(\mathbf{v}) \rightarrow E(\mathbf{v})$

L-systems provide a compact way of defining subdivision curves and they are easy to modify. For example, let us expand L-system 3 to support adaptive subdivision of open curves (see L-system 5). For this purpose, we add a second parameter t specifying the type of a point to each symbol P. This parameter is equal to 1 (the default) if the point is to be subdivided and 0 if it should not be subdivided any further. We also extend L-system 3 with three new productions. Productions p_1 and p_2 make sure that the point of type 0 next to a point of type 1 creates only one new point and not two. Production p_3 tests whether the point is close to the midpoint between its neighbors, in which case the newly created points are of type 0. Our approach is similar to the one described by Xu *et al.* [7]. Here is the resulting L-system:

L-system 5:

p_1: $P(\mathbf{v}_l, t_l) < P(\mathbf{v}, t)$: $t = 0 \;\&\; t_l = 1 \rightarrow P(\frac{1}{4}\mathbf{v}_l + \frac{3}{4}\mathbf{v}, 0)$

p_2: $P(\mathbf{v}, t) > P(\mathbf{v}_r, t_r)$: $t_r = 1 \;\&\; t = 0 \rightarrow P(\frac{3}{4}\mathbf{v} + \frac{1}{4}\mathbf{v}_r, 0)$

p_3: $P(\mathbf{v}_l, t_l) < P(\mathbf{v}, t) > P(\mathbf{v}_r, t_r)$: $|\mathbf{v} - \frac{\mathbf{v}_l + \mathbf{v}_r}{2}| < T$
$\rightarrow P(\frac{1}{4}\mathbf{v}_l + \frac{3}{4}\mathbf{v}, 0)P(\frac{3}{4}\mathbf{v} + \frac{1}{4}\mathbf{v}_r, 0)$

p_4: $E(\mathbf{v}_l) < P(\mathbf{v}, t) > P(\mathbf{v}_r, t_r)$
$\rightarrow P(\frac{1}{2}\mathbf{v}_l + \frac{1}{2}\mathbf{v}, t)P(\frac{3}{4}\mathbf{v} + \frac{1}{4}\mathbf{v}_r, t)$

p_5: $P(\mathbf{v}_l, t + l) < P(\mathbf{v}, t) > E(\mathbf{v}_r)$
$\rightarrow P(\frac{1}{4}\mathbf{v}_l + \frac{3}{4}\mathbf{v}, t)P(\frac{1}{2}\mathbf{v} + \frac{1}{2}\mathbf{v}_r, t)$

p_6: $P(\mathbf{v}_l, t_l) < P(\mathbf{v}, t) > P(\mathbf{v}_r, t_r)$
$\rightarrow P(\frac{1}{4}\mathbf{v}_l + \frac{3}{4}\mathbf{v}, 1)P(\frac{3}{4}\mathbf{v} + \frac{1}{4}\mathbf{v}_r, 1)$

p_7: $E(\mathbf{v}) \rightarrow E(\mathbf{v})$

In this L-system we are taking advantage of the assumption that if more than one production can be used to rewrite the predecessor, the one that appears first in the production list is chosen. For example, the third production is applied only to symbols to which the first or second production cannot be applied. Figure 1 illustrates the operation of L-system 5.

In the next section we implement these L-systems directly on a GPU.

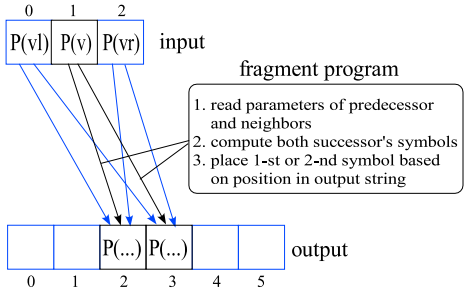


Figure 2: An L-system on GPU, algorithm 1: each symbol is replaced by two new symbols.

3 L-systems on a GPU

Algorithm 1. An L-system in which each symbol is replaced by a constant number of k symbols (for example, L-system 1 or L-system 2) is easy to implement on graphics hardware that supports floating-point fragment programs (also known as pixel shaders) (Figure 2). We store the initial string in one line of a texture[4]. The letter symbol of each point is in the alpha channel, and the coordinates are in the RGB channels. Given an input string of length n, we draw a line of length kn into a P-buffer, off-screen memory located on the graphics card. A pixel of the line at position i represents the $i\%k$-th point of the successor of the i/k-th symbol in the input string. As the line is rendered, the fragment program reads texel values at positions $(i/k - 1)\%n$, $(i/k)\%n$ and $(i/k + 1)\%n$ (the left context, the strict predecessor, and the right context), and sets the value of pixel i as defined for the $i\%k$-th point of the production successor. The positions of the predecessor and neighbors are deduced from three sets of texture coordinates. The texture coordinates of neighbors are shifted to the left and right from the predecessor coordinates. The value of i used to determine the symbol of the successor is set using a 1D texture coordinate with values of 0 and kn assigned with the two vertices of the line.

Once the symbol of the successor is identified, the fragment program has to compute symbol's parameters. If the computations for all successor's symbols are similar, such as in case of L-system 2, they can be performed by a single fragment program. This program uses a set of local fragment program parameters or an input texture to specify different parameters for each computation (equivalent to arrays a and b in L-system 2). The correct set of parameters is selected based on the symbol's position i in the final string. If the computations vary significantly, they cannot be expressed by a single formula that uses different parameters for different symbols of the successor. In this case we can apply a fragment program that computes all symbols of the successor

[4]If one line is not enough, we modify the neighbor selection process in order to store the string in a 2D texture.

and selects the one identified by the position i. If these computations do not fit into a single fragment program, we can use a set of fragment programs applied one after another, each setting only a particular symbol of the successor. This will be less of an issue in the future, because the maximum length of a fragment program will be significantly larger.

In each subsequent iteration of the algorithm, we bind the P-buffer as the input texture and use another P-buffer as the output. Finally, we read the final string using glReadPixels, and render the vertices. In the near future, the drivers will support rendering into a vertex array, which will make it possible to avoid the readback.

Algorithm 2. If an L-system has more than one production, and they have successors of different length (for example, L-system 3) there are two issues: to find a production for each symbol, and to position the successor in the output string. There are two approaches to finding the production. If the productions are of a similar form and the coefficients used to compute the successor's parameters can be tabulated, such as in L-system 3, 4 or 5, two fragment programs can be used, one to find an applicable production and one to apply it. These programs use textures that specify the correspondence between a specific predecessor and its successor, given the predecessor's context (see below for more details). If L-system productions vary significantly, it is necessary to represent each production or a group of similar productions using a separate fragment program.

The first approach is more desirable because it is easy for a user to modify the L-system by changing texture data without any changes to fragment programs. All productions are specified using two textures, the *predecessor texture* and the *successor texture*. Each row of the predecessor texture stores information on the context of all productions with the same strict predecessor. The productions are specified one after another, each production is specified by its four neighbors, the successor length and the index of the first symbol of the successor in the successor texture (see Figure 3). Optionally, for each production, the row can also store coefficients used to evaluate the production's condition. Each column of the successor texture stores the symbols and affine combination coefficients for one successor symbol of one production.

Figure 3 illustrates the operation of an L-system using textures organized as described above. Fragment program 1 finds the matching production for each point in the predecessor string, and outputs the successor length l and the index s of the first symbol of the successor, stored in the successor texture. Since the program tests one set of neighbors at a time, this takes up to M passes, where M is the maximum number of productions with the same strict predecessor.

To determine the position of each successor in the output string, we simulate the scan-add operation defined as foolows [4]: if y =scan-add(x), then $y[i] = \sum_{j=0}^{i-1} x[j]$ (and $y[0] = 0$)[5]. Before the productions are applied we run fragment program 2, which sums the lengths of all successors to

[5]We use the version of the scan-add operation, in which we do not add the value at the given position to the sum.

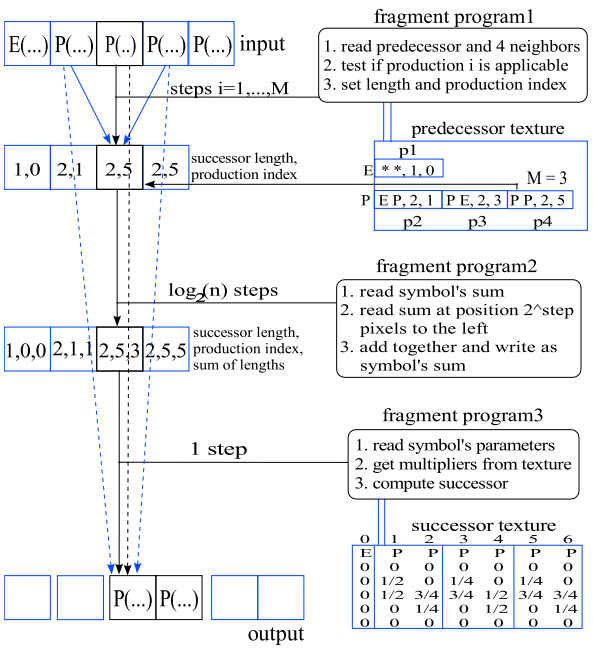


Figure 3: An L-system on GPU, algorithm 2: productions specified using textures, successor lengths vary. Texture data correspond to L-system 3.

the left of a given symbol. This can be done in $\lfloor log_2(n) \rfloor$ passes. These sums are read with glReadPixels and used to create a set of line segments on a CPU, each starting at the pixel given by a sum. Again, the readback can be avoided once rendering into vertex arrays is supported in drivers.

The 1D texture coordinates at vertices of each line segment are set to s and $s + l$. Fragment program 3, executed for each pixel of each line segment, accesses the successor texture column identified by the 1D texture coordinate. It retrieves the symbol and its affine combination coefficients from the texture, computes the affine combination of the predecessor point and its neighbors, and sets the new symbol and the computed value.

If we have a set of productions whose successors have the same length, the scan-add step can be skipped. A single line of length kn is drawn as in algorithm 1 and the position i is used to determine the symbol of the successor in fragment program 3. Sometimes we can determine the successor from the position i even if the productions have successors of different length. In L-system 3, for example, only the first and last symbol in the string produce one new symbol, all other symbols produce two, and therefore the position of each successor can be determined in advance.

In the subsequent iteration of the subdivision process, the P-buffer is used as a input texture for the fragment programs. The final string is read with glReadPixels, and the vertices are rendered as in the closed curve case.

4 Results

Figures 4 and 5 show sample subdivision curves generated using L-systems 2, 3 and 4 implemented on the ATI's

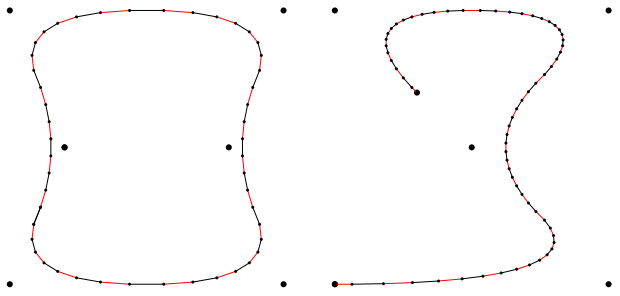


Figure 4: Closed and open subdivision curves generated in 3 steps using L-system 2 and L-system 3 implemented on a GPU.

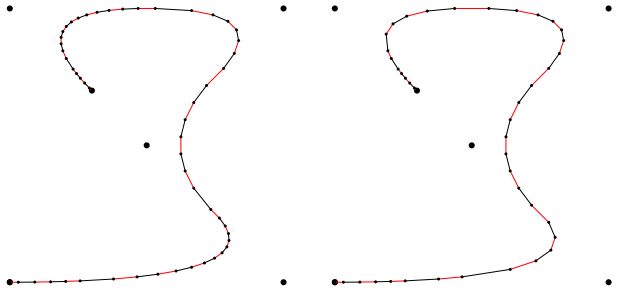


Figure 5: Adaptive subdivision curves generated in 5 steps using L-system 5 implemented on a GPU. $T = 0.025$ and 0.05.

Radeon 9700.

In the case of the closed curve in Figure 4, we used algorithm 1. A single fragment program generates both new points of the successor in a single rendering pass. The arrays a and b (L-system 2) are set using local parameters of the fragment program. The program has 15 instructions (12 arithmetic instructions and 3 texture reads). It took 0.4 ms to generate the closed curve in Figure 4, out of which 0.3 ms were spent in switching the rendering context from one P-buffer to another[6]. One context switch took about 0.1 ms, but the future drivers should significantly reduce this unnecessary overhead [1]. The overhead of context switches is also reduced if several curves are evaluated at once. Subdividing a curve defined by 4 control points 8 times (subdivision level 8) resulted in 1024 points and took (8*0.1 + 0.2) ms. These times do not include the final readback, which for 1024 points takes about 0.17ms.

Using a software implementation on a 2.4 GHz Pentium 4 CPU to generate three levels of subdivision took about the same time (0.1 ms), but at higher subdivision levels the GPU implementation became faster (if we ignore the context switch overhead). At subdivision level 8, the GPU was about twice as fast as the CPU.

In the case of open curves we used algorithm 2. The L-system is automatically converted into the predecessor texture and successor texture. Fragment program 1 has 45 instructions (35 arithmetic instructions + 10 texture reads), fragment program 2 has 24 instructions (16+8)[7], and fragment program 3 has 18 instructions (15+3). It took 2.1 ms to generate the open curve in Figure 4, out of which 1.35 ms were spent on 11 context switches and 0.3 ms on 3 readbacks after each scan-add operation. The overall time of 2.1 ms can be reduced by 0.9 ms (5 context switches + 0.4 ms) by skipping the scan-add operation, because in L-system 3 the position of each production successor can easily be determined (see Section 3). The timings for the two adaptive curves in Figure 5 are 3.8 ms (2.4 ms for 20 context switches, 0.5 ms for 5 scan-add readbacks) and 3.7 ms (2.3 ms for 19 context switches, 0.5 ms for 5 scan-add readbacks). In this case we cannot skip the scan-add step.

The software implementation of open subdivision curves is faster than the GPU implementation for a small number of control points. Subdividing a non-adaptive open curve from Figure 4 up to level 8 was 4 times faster in software (ignoring the cost of context switches). The GPU disadvantage is caused by having to perform several rendering passes to find a production, and several passes to perform scan-add operation, while dealing with a relatively small number of pixels. Once we increase the number of pixels by evaluating several curves in parallel the GPU algorithm becomes faster. Evaluating 16 non-adaptive open curves (8 subdivision levels) took about the same time on the CPU and the GPU, and for 32 curves the GPU was about 50% faster. Consequently, using the GPU for evaluating subdivision curves is better only if one needs to evaluate many of them at once.

5 Conclusions

We created a set of fragment programs on ATI's Radeon 9700 that implement L-systems capable of generating subdivision curves. We implemented not only selected basic schemes, but also an adaptive scheme.

We chose L-systems as a conceptual basis for our implementation because they compactly express many subdivision schemes and they can easily be modified by changing few parameters or adding a few new productions. Our approach is similar to the implementation of L-systems on the Connection Machine by Ortiz *et al.* [4]. In contrast to Ortiz *et al.*, however, our implementation supports parametric L-systems and uses different data structures, more suitable for fast access by a GPU.

As the results indicate, if we have to perform more than one rendering pass for a single subdivision step the GPU implementation becomes faster compared to a CPU implementation when many curves are evaluated at once. An additional problem is the fact that current drivers do not implement switches of rendering context very efficiently and that the API for rendering to vertex arrays has not been finalized yet and thus the drivers lack the support for this functionality.

[6] We have to alternate between two P-buffers because a single P-buffer cannot be used both as an input and output.

[7] We observed that it is faster to use a longer fragment program 2 that sums 8 values at once and reduce the number of passes needed for the scan-add operation than to use a shorter program in more passes (even if we ignore the cost of context switches).

An intriguing problem for further research is an extension of this work to subdivision surfaces, where the advantage of a GPU implementation is likely to be more significant, because we are dealing with larger numbers of points.

Acknowledgemens

We would like to thank Sylvain Lefebvre for helpful comments on this note.

References

[1] J. Bolz and P. Schröder. Evaluation of subdivision surfaces on programmable graphics hardware. http://www.multires.caltech.edu/pubs/GPUSubD.pdf. Submitted for publication.

[2] N. Goodnight, G. Lewin, D. Luebke, and K. Skadron. A multigrid solver for boundary-value problems using programmable graphics hardware. Technical Report CS-2003-02, Univ. of Virginia Dept. of Computer Science., January 2003.

[3] W. R. Mark, S. Glanville, and K. Akeley. Cg: A System for Programming Graphics Hardware in a C-like Lanuage. *ACM Transactions on Graphics*, 22(3), July 2003. To appear.

[4] L.F. Ortiz, R.Y. Pinter, and S.S. Pinter. An array language for data parallelism: Definition, compilation, and applications. *The Journal of Supercomputing*, (5):7–29, 1991.

[5] P. Prusinkiewicz, F. Samavati, C. Smith, and R. Karwowski. L-system description of subdivision curves. *International Journal of Shape Modeling*. To appear..

[6] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002.

[7] Z. Xu and K. Kondo. Local Subdivision Process with Doo-Sabin Subdivision Surfaces. *SMI 2002:International Conference on Shape Modelling and Applications*, May 2002.