Modeling plant development with L-systems

Przemyslaw Prusinkiewicz, Mikolaj Cieslak, Pascal Ferraro, and Jim Hanan

Abstract Since their inception in 1968, L-systems have become a key conceptual, mathematical and software tool for modeling plant development at different levels of plant organization spanning molecular genetics, plant physiology, whole plant architecture, and plant communities. The models can be descriptive, directly recapitulating observations and measurements of plants; mechanistic, explaining higher-levels processes in terms of lower-level ones; or they may combine features of both classes. We present the basic idea of L-systems, motivate and outline some of their most useful extensions, and give a taste of current techniques for modeling with L-systems. The sample models progress in the scale of organization from a bacterium to a herbaceous plant to a tree, and simulate different forms of information transfer during the development, from communication between adjacent cells to bidirectional information exchange with the environment.

1 Genesis of the idea

The discovery of the structure and functioning of DNA placed molecular genetics in the center of modern biology, and opened the door for reducing diverse biological processes to their biochemical (and, ultimately, physical) basis. However, the sequencing of the genome of numerous organisms, including humans, has also

Przemyslaw Prusinkiewicz
Department of Computer Science, University of Calgary, e-mail: pwp@ucalgary.ca

Mikolaj Cieslak
Department of Computer Science, University of Calgary, e-mail: msciesla@ucalgary.ca

Pascal Ferraro
Department of Computer Science, University of Calgary, e-mail: pferraro@ucalgary.ca

Jim Hanan
Centre for Horticultural Science, University of Queensland, e-mail: j.hanan@uq.edu.au

highlighted the gap between knowing the genome and understanding how it regulates the development and form of an organism. This regulation is not direct; instead, genes and molecular-level processes establish local rules for the spatially distributed dynamic processes of morphogenesis, from which the developing forms arise. The emergence of global patterns, forms or behaviors through the local interaction of their components is the defining feature of self-organization and highlights the inherent difficulty in studying and understanding development: properties of self-organizing systems are often non-intuitive and difficult to analyze. Computational models and simulations facilitate studies of self-organizing phenomena by revealing the consequences of different hypothetical rules, explicitly specified by the modeler. A series of computational experiments, in which the model is modified and refined until the emerging developmental dynamics approximates reality with sufficient accuracy, leads to insights into the plausible processes underlying the analyzed processes, patterns and forms.

The use of computational models raises the question of how they should be conceptualized, specified and executed to effectively support the process of scientific discovery. Similar questions can be raised in any application of computing, but in the case of morphogenesis the answers are particularly elusive. This is due to the discrepancy between the standard view of computation that underlies commonly used programming languages on the one hand, and the nature of the problems of morphogenesis on the other.

The standard view of computation is related to the classical (von Neumann) model of computer architecture. According to this model, computation is organized around a central processing unit (CPU), which executes a sequence of operations on a set of numbers in a well-defined order. The geometric concept of space is not part of this model. In contrast, morphogenesis involves multiple processes taking place in parallel, and it is an inherently geometric phenomenon. Spatial relations between components of the developing organism play a key role in defining which specific processes take place in each component at each point of time. Further complicating the matter, the number of components and processes may change as the organism grows. Because of these changes, the standard mathematical framework of dynamical systems, in which temporal evolution of a system is described using a predefined set of variables and equations, becomes too limiting (Giavitto et al, 2002).

In view of these discrepancies, nonstandard models of computation, programming languages and computational devices have been proposed to specify morphogenetic processes and implement simulations effectively. A well-known example is that of *cellular automata* (also called cellular spaces), invented in the 1950s by von Neumann and Ulam. Their early applications included a biologically-inspired model of self-replication (von Neumann, 1966), and — of particular importance to the history of computational studies of morphogenesis — the first simulations of growing branching structures (Ulam, 1962, 1966).

With cellular automata, space is represented as an array of cells, each of which houses an automaton that indicates, depending on its state, whether the cell is empty or occupied by a component of the modeled structure. The automata change their states in parallel (synchronously) to simulate a uniform progress of time within the

entire structure. Transition functions governing state changes can be conveniently specified using declarative statements (rules), which characterize the next state of each automaton/cell as a function of its current state and the state of its neighbors. An example of such a rule for a one-dimensional cellular automaton defined over the set of states $\{0,1\}$ might be "a cell in state 1, situated between cells in state 0 to the left and state 1 to the right, will change its state to 0."

L-systems (Lindenmayer, 1968) were inspired by cellular automata, and in some simple cases are indistinguishable from them. Using the terminology of L-systems, the above rule would be termed a *production* that replaces the strict *predecessor*, symbol 1, occurring in the *context* of symbol 0 to the left and symbol 1 to the right, with the *successor* 0. Such a production can be written as $0 < 1 > 1 \rightarrow 0$ (here symbols $<$ and $>$ do not denote inequalities, but simply separate the strict predecessor from the left and right contexts). Four sample L-systems inspired by cellular automata are specified in Table 1, and the first 32 steps of their operation are shown in Figure 1.

Figure	**a**	**b**	**c**	**d**
Axiom	$0^{32}10^{32}$	$A00100B$		
Predecessor	Successor			
$0<0>0$	0	0	0	0
$0<0>1$	1	1	1	1
$0<1>0$	0	0	1	1
$0<1>1$	0	0	0	1
$1<0>0$	1	1	1	1
$1<0>1$	0	0	1	0
$1<1>0$	0	0	0	0
$1<1>1$	0	0	0	0
A	—	$A0$	$A0$	$A0$
B	—	$0B$	$0B$	$0B$

Table 1: Sample L-systems inspired by cellular automata. L-system **a** rewrites symbols of the initial string (axiom) without changing its length, as typical of cellular automata (the power notation 0^{32} denotes a string of 32 zeroes). In L-systems **b**–**d**, productions operating on symbols A and B gradually extend the generated strings.

The L-system in Figure 1a most closely mimics the operation of a cellular automaton, in the sense that the space in which it operates — a row of 65 cells — is given in advance. In the course of the simulation, the states of individual cells may change, but their number and spatial configuration are fixed. In contrast, the L-systems in Figures 1b–d employ productions $A \rightarrow A0$ and $B \rightarrow 0B$ operating on additional symbols A and B (colors used for emphasis) to gradually expand the strings of symbols that grow from the short L-system *axiom* (initial string) $A00100B$. The ability to expand (or contract) the modeled structure over time is a distinctive feature of L-systems.

There is also a more subtle distinction between cellular automata and L-systems. Cellular automata describe a quantity (state) associated with (discrete) points of

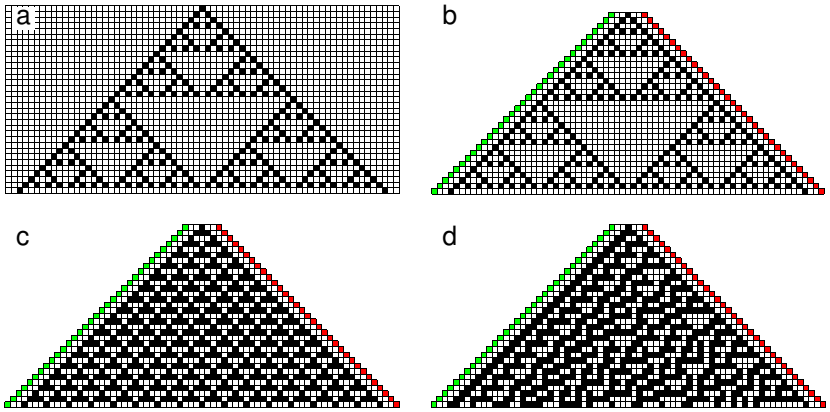

Fig. 1: Operation of L-systems from Table 1. White cells represent the symbol 0, black cells the symbol 1. In each simulation, the top row represents the initial state, and the subsequent rows represent the results of each simulation steps. The space in which simulations operate is fixed in model **a** and gradually expands in models **b**–**d**, in which colored cells produce new white cells. These examples also show that small differences in productions may result in qualitatively different patterns, including self-similar (**a**, **b**), repetitive (**c**) and chaotic (**d**) patterns (Wolfram, 1984, 2002). These differences highlight the need for the modeling and simulations in the studies of self-organization. The rules used in the above examples are commonly referred to as Rule 18 (**a**, **b**), Rule 54 (**c**) and Rule 30 (**d**). These names result from interpreting the list of successors in the respective columns of Table 1, read from the bottom up, as binary numbers, e.g. binary 00010010 equals 30 in common decimal notation.

space. This space-centered perspective is known as Eulerian. In contrast, L-systems focus on the developing structure itself, a perspective known as Lagrangian. The difference between these perspectives comes to light when the modeled structure grows over time, and is particularly evident when new components are added not only at the structure boundary, but in its interior as well. Some components then change their position and "flow" through space as they are pushed by other components that need room to fit. It makes a difference whether this process is described from the perspective of points of space or individual components (Prusinkiewicz and Runions, 2012). The latter perspective, afforded by L-systems, is often natural and convenient in the description of development.

2 Modeling cell division patterns

A model of vegetative segments of the filamentous blue-green bacterium *Anabaena catenula* provides a simple example illustrating the capability of L-systems to simulate growth and cell division patterns in one dimension. A vegetative *Anabaena* segment consist of sequences of longer and shorter cells that appear to be arranged at random, but in fact divide deterministically (Mitchison and Wilcox, 1972). Dur-

ing the development, short cells elongate, and long cells divide asymmetrically into a short and long cell. A short daughter cell is always produced on the side of the older wall separating the mother cell from its neighbours. The following L-system captures these principles, with symbols S and L denoting a short and a long cell, respectively, and symbols W denoting cell walls that delimit cells within the filament: Compared to the L-systems in Table 1, this L-system incorporates two extensions.

Axiom:		$W(0)LW(1)$
Productions:	p_1 :	$W(a) \rightarrow W(a+1)$
	p_2 :	$W(a_l) < L > W(a_r) : a_l \geq a_r \rightarrow SW(0)L$
	p_3 :	$W(a_l) < L > W(a_r) : a_l < a_r \rightarrow LW(0)S$
	p_4 :	$S \rightarrow L$

Table 2: L-system modeling the development of a vegetative segment of *Anabaena*.

The first one is the introduction of a numerical *parameter a* representing wall age. It is incremented in every derivation step by production p_1. The second extension is the introduction of *conditions* that guard production application according to the parameter values. Specifically, conditions $a_l \geq a_r$ and $a_l < a_r$ determine whether a long cell L will divide into a short cell S followed by a long cell L (production p_2) or into a long cell L followed by a short cell S (production p_3). The choice depends on the age of the walls surrounding L. The remaining production, p_4, represents the growth of a short cell S into a long cell L. The first five simulation steps, beginning with a single long cell bound by walls with the age set arbitrarily to 0 and 1 in the axiom, are shown in Figure 2. As this example illustrates, numerical parameters and conditional application of productions are very useful in modeling practice. They first appeared as a programming construct in the L-system-based simulator CELIA (Baker and Herman, 1970, 1972; Lindenmayer, 1974) and were subsequently formalized by Prusinkiewicz and Hanan (1990), Prusinkiewicz and Lindenmayer (1990), and Hanan (1992).

Experimenting with the L-system in Table 2 provides insights beyond the original objective of simulating the development of the vegetative segment of *Anabaena*. For instance, Figure 3a shows the developmental sequence obtained by removing production p_4. Cell L, present in the axiom, is then the only cell that divides. Cells S, created by L in consecutive derivation steps, appear on both of sides of L and "internalize" it near the filament center. A similar mechanism, operating in two dimensions, explains the distribution of stomata in the leaf epidermis. As in the linear case, a dividing cell — precursor of a stoma — produces new cells towards its older walls. This process results in the internalization of the precursor cell and, consequently, the separation of one stoma from another by non-stomatal cells: a fundamental feature of stomata distribution (Robinson et al, 2011).

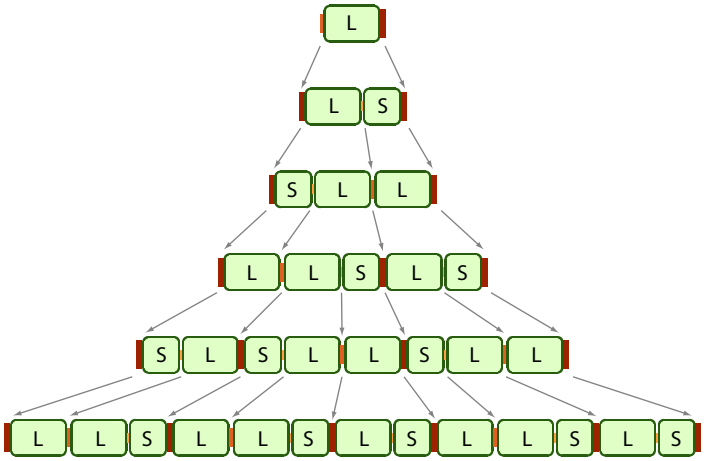

Fig. 2: The first five steps of the simulation of the development of a vegetative filament of *Anabaena*, simulated using the L-system in Table 2. The size and color of bars between the cells and on the outside of the filament indicates the age of the walls (0, 1, or ≥ 2). Arrows indicate how the walls propagate from one simulation step to the next.

A further modification reverts conditions in productions p_2 and p_3 as follows (color added for emphasis):

$$\begin{aligned} p_2' &: \quad W(a_l) < L > W(a_r) : a_l \leq a_r \rightarrow SW(0)L \\ p_3' &: \quad W(a_l) < L > W(a_r) : a_l > a_r \rightarrow LW(0)S \end{aligned}$$

The resulting L-system creates a drastically different developmental sequence, in which the dividing cell L is positioned at the end of a growing structure, extending it in one direction only (Figure 3b). A similar process is commonly observed in plants, from filamentous algae to mosses, ferns and higher plants, where an apical cell (or multicellular apical meristem) creates an axis of plant development. The contrast between the development of a filament with multiple dividing cells, the division of a single cell leading to its internalization, and the maintenance of an axis with the dividing cell situated at an apex shows that even very simple L-systems can shed light on nontrivial relations between the local rules of development and the global characteristics of the resulting structures.

3 Programming with L-systems

The L-system in Table 2 has been specified using a syntax inspired by the notion of rewriting, as studied in mathematics and theoretical computer science (in particular, formal language theory, where it underpins the concept of formal grammars). The relation of L-systems to rewriting systems was first recognized by Lindenmayer (1971). With the subsequent extensions to parametric L-systems, production such as those shown in Table 2 are written as:

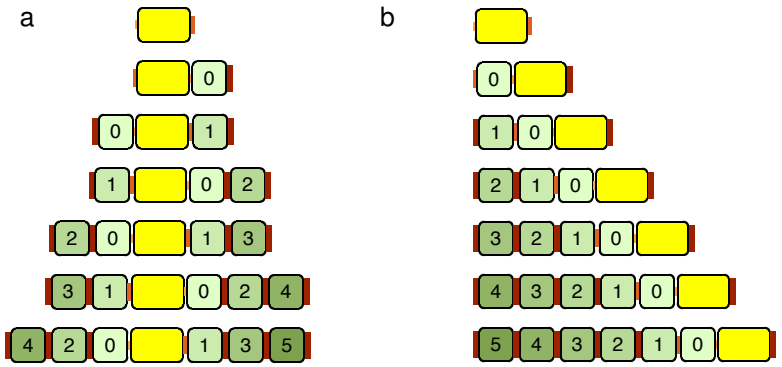

Fig. 3: Two developmental sequences generated by modifying the L-system for the vegetative segment of *Anabaena*. The size and color of bars between the cells and on the outside of the filament indicates the age of the walls (0, 1, or ≥ 2), as in Figure 2. As the long cell divides, the daughter short cell is positioned on the side of the older (**a**) or younger (**b**) wall of the mother cell. Numbers and colors of the short cell indicate their age.

$$left\ context < strict\ predecessor > right\ context : condition \rightarrow successor$$

for instance

$$A(u) < B(x,y) > C(v)D : u < v \rightarrow E(x+\sin(u), y+\cos(v))$$

For simple L-systems, this mathematically-inspired syntax works well; in particular, its conciseness makes it easy to specify and modify productions, an inherent component of developing models in an interactive computing setting. Unfortunately, it does not easily scale up to more complex L-systems (Prusinkiewicz, 2004). For instance, sequences of single-letter symbols associated with multiple parameters look cryptic, making model specifications difficult to read and maintain, and there are no constructs for evaluating conditions and parameter values programmatically. Extensions to the mathematical notation addressing some of these limitations have been proposed and are useful (Prusinkiewicz and Lindenmayer, 1990), but a fundamental solution is to combine L-system constructs with a programming language. Existing systems include binding L-systems with C/C++ (Karwowski and Prusinkiewicz, 2003; Prusinkiewicz et al, 2007), Java (Kniemeyer, 2004; Kniemeyer et al, 2007), and Python (Boudon et al, 2012). For instance, the basic format of productions in the L+C language (Karwowski and Prusinkiewicz, 2003; Prusinkiewicz et al, 2007) is:

$$left\ context < strict\ predecessor > right\ context\ :\ \{statement\ block\}.$$

The statement block is (relatively unrestricted) C++ code, allowing for the specification of arbitrarily complex computations involving module parameters, and extended with `produce` statements , which precede the specification of a successor. A single production may include several `produce` statements, which provides a

means of specifying alternative outcomes to production execution, depending on the evaluation of conditions. Furthermore, modules may have parameters of different types, including entire data structures. This use of structures improves the clarity of L-system specification if numerous parameters are involved, and implies that in L+C modules are declared.

4 Incorporating a genetic regulatory network into an L-system model

The next model illustrates L+C language constructs and provides an example of an L-system incorporating a genetic regulatory network (GRN). This example returns to *Anabaena*, but focuses on an additional element of its development: the differentiation of a special type of cell, the heterocyst.

In an *Anabaena* filament grown in an environment without nitrogenous compounds there is a division of functions: vegetative cells assimilate carbon from the atmosphere in the process of photosynthesis, while heterocysts fix nitrogen (Adams and Duggan, 1999; Herrero et al, 2016). These functions are performed by different cells because the enzyme involved in nitrogen fixation, the nitrogenase, disintegrates in the presence of the oxygen produced during photosynthesis. On the average, heterocysts are separated by approximately 10 vegetative cells (Haselkorn, 1998). New heterocysts differentiate as the distance between existing heterocysts increases due to the division of the vegetative cells in-between. The mechanism controlling this differentiation has been of interest for a long time, and has included the construction of L-system models both to address the biological problem itself and to illustrate model construction with L-systems (e.g. (Baker and Herman, 1970, 1972; Lindenmayer, 1974; de Koster and Lindenmayer, 1987; Prusinkiewicz and Lindenmayer, 1990; Giavitto et al, 2002; Coen et al, 2004; Prusinkiewicz and Lane, 2013)).

It was initially thought that heterocyst differentiation is triggered by low concentration of nitrogeneous compounds (Fogg, 1949), which are synthesized in the heterocyst and diffuse into vegetative segments where they are used. The concentration of these compounds would decrease as the length of a vegetative segments increases, and eventually would fall below a threshold near the centre of the vegetative segment, triggering the differentiation of a new heterocyst (Fritsch, 1951). Advances in molecular biology have shown that, while the general idea of a diffusing substance controlling the differentiation of heterocysts is correct, the morphogenetically active diffusing substance is a small protein, PatS, acting as a proxy for the nitrogenous compounds (Yoon and Golden, 1998). The production of PatS is regulated by another protein, HetR, which is present in high concentration in heterocysts and defines their identity (Buikema and Haselkorn, 1991). The interaction between PatS and HetR is qualitatively depicted in Figure 4.

To incorporate these interactions into an L-system model it is necessary to give them a mathematical form. Following (Wilcox et al, 1973; Hammel and Prusinkiewicz, 1996; Giavitto et al, 2002), we cast it as an activator-inhibitor model

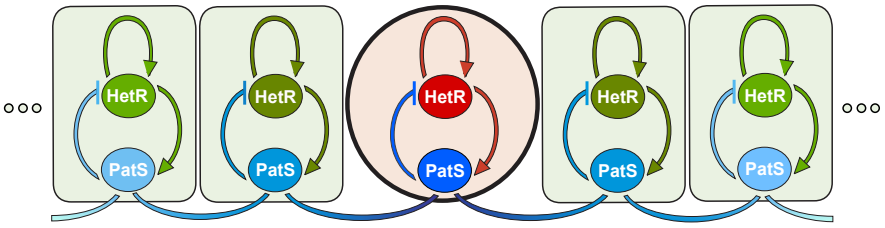

Fig. 4: Interaction between HetR and PatS in an *Anabaena* filament. Arcs with arrows denote upregulation, arcs with bars downregulation, and horizontal arcs diffusion of the respective proteins. Color intensity indicates concentration of HetR and PatS in the heterocyst (circular cell) and the neighbouring vegetative cells. Adapted from (Coen et al, 2004).

(Gierer and Meinhardt, 1972; Meinhardt, 1982): a type of reaction-diffusion model in which the interaction between the intervening substances is qualitatively consistent with the interaction between PatS and HetR (Adams and Duggan, 1999). The equations for an arbitrary cell in the filament then have the form:

$$\frac{d[\text{HetR}]}{dt} = \rho \frac{[\text{HetR}]^2}{[\text{PatS}]} + \rho_0 - \mu[\text{HetR}], \tag{1}$$

$$\frac{d[\text{PatS}]}{dt} = \rho[\text{HetR}]^2 + \rho_0 - \nu[\text{PatS}] + \Phi. \tag{2}$$

Symbols [HetR] and [PatS] denote the concentration of the corresponding proteins, μ and ν control their turnover (use and decay), ρ and ρ_0 control production rates, and Φ represents the flux of PatS into the cell under consideration from its neighbouring cells (for simplicity, we assume that all cells have a unit volume, and the walls between them have a unit area). Given the diffusive character of PatS transport, this flux is proportional to the difference in concentration of PatS in the cell under consideration and its left and right neighbours, here identified by subscripts l and r (Fick's law):

$$\Phi = D\left(([\text{PatS}]_l - [\text{PatS}]) + ([\text{PatS}]_r - [\text{PatS}])\right). \tag{3}$$

The coefficient of proportionality D controls the rate of diffusion through cell walls.

The specification of an L+C model corresponding to these equations begins with definitions of the simulation parameter values. These definitions are given in the standard C/C++ format:

```
/* Definition of constants used in simulation */
#define rho    1.0      // controls rate of the production of HetR and PatS
#define rho0   0.01     // controls basic production of HetR and PatS
#define mu     0.1      // turnover rate of HetR
#define nu     1.0      // turnover rate of PatS
#define D      2.0      // diffusion constant for PatS
#define thr    20       // threshold HetR concentration defining a heterocyst
#define dt     0.05     // time step
```

In addition to parameters controlling the biochemical aspects of the simulation, the parameter list includes geometric parameters that control the dimensions and growth rates of the cells:

```
#define gr1    0.004    // relative elementary growth rate of vegetative cells
#define MAX    1.0      // threshold length at which a vegetative cell divides
#define L      0.61804  // length of the longer daughter cell after division
#define S      0.38196  // length of the shorter daughter cell after division
#define TW     0.5      // target width of vegetative cells
#define TD     0.7      // target diameter of heterocysts
#define gr2    0.015    // rate of cell dimension adjustment towards target
```

The genetic regulatory network is then specified as a system of two functions that determine the rates of expression (production) of proteins HetR and PatS, given their concentrations in the cell:

```
/* Functions defining the genetic regulatory network */
float hetR(float HetR, float PatS)  // HetR production rate
{
   return rho*HetR*HetR/PatS + rho0;
}

float patS(float HetR)              // PatS production rate
{
   return rho*HetR*HetR + rho0;
}
```

The data structures involved in the simulation are specified next, following standard C/C++ syntax:

```
/* Declaration of data structures characterizing cells and walls */
struct CellData
{
   float HetR, PatS; // concentration of HetR and PatS
   float l, w;       // cell length and width
};

struct WallData
{
   float flux;       // PatS flux through the well
   float age;        // wall age
};
```

The L+C-specific code begins with the declaration of module types and the initial state (axiom) of the simulation:

```
/* Initial values */
CellData icd1 = {0.02, 100, S, TW};    // HetR, PatS, length, width
CellData icd2 = {0.01, 100, L, TW};    // HetR, PatS, length, width
WallData iwd  = {0, 0};                // flux, age

/* Declaration of module types used in the model */
module Cell(CellData);
module Wall(WallData);

/* Defintion of the initial structure and state of the simulation */
axiom: Right(90) Wall(iwd) Cell(icd1) Wall(iwd) Cell(icd2) Wall (iwd);
```

Module `Right` orients the filament horizontally on the screen, cf. Section 5. The key part of the model is specified using two type of rules. Ordinary productions are associated with the progress of time by interval `dt`. The production for walls simply advances the age of walls, as in the L-system in Table 2:

```
58 production:
59 Wall(wd) :  // advance wall's age in each simulation step
60 {
61    wd.age += dt;
62    produce Wall(wd);
63 }
```

The production for cells has two components. The first component (lines 67–76) represents a numerical solution to the initial value problem given by Equations 1 and 2 using the forward Euler method. For clarity, different components of this solution (protein diffusion, production, and turnover of proteins) are split (MacNamara and Strang, 2016) into separate statements. The second component (lines 78–86) uses ad-hoc rules to simulate cell growth. A vegetative cells elongates with the relative elementary growth rate `gr1`, while its width tends to target `TW`. A heterocyst asymptotically tends to a rounded shape with diameter `TD`.

```
65 Wall(wdl) < Cell(cd) > Wall(wdr) :     // update cell state
66 {
67 // Diffusion of patS
68    cd.PatS += (wdl.flux - wdr.flux) * dt;
69
70 // Gene expression
71    cd.HetR += hetR(cd.HetR,cd.PatS) * dt;
72    cd.PatS += patS(cd.HetR) * dt;
73
74 // Decay of proteins
75    cd.HetR -= mu*cd.HetR * dt;
76    cd.PatS -= nu*cd.PatS * dt;
77
78 // Growth
79    if (cd.HetR < thr) {                // vegetative cell...
80       cd.l += cd.l * gr1 * dt;         // grows in length;
81       cd.w += gr2 * (TW - cd.w) * dt;  // width is adjusted towards target
82    }
83    else {                              // heterocyst...
84       cd.l += gr2 * (TD - cd.l) * dt;  // grows towards target in length
85       cd.w += gr2 * (TD - cd.w) * dt;  // and in width
86    }
87    produce Cell(cd);
88 }
```

These productions are followed by decomposition rules (Prusinkiewicz et al, 2000; Karwowski and Prusinkiewicz, 2003), which are applied immediately after the productions (in the same derivation step) and perform computations that do not involve advancing time. The decomposition rule for walls computes the flux of PatS through each wall according to Equation 3:

```
93 /* Determine the flux of PatS through the wall */
94 Cell(cdl) < Wall(wd) > Cell(cdr) :
95 {
96    wd.flux = D * (cdl.PatS - cdr.PatS);   // Fick's law
97    produce Wall(wd);
98 }
```

The decomposition rule for cells specifies asymmetric division of vegetative cells according to the age of the incident walls in a manner similar to the L-system in Table 2, except that the rule is applied when the mother cell reaches the threshold length rather than threshold age. Cell division is assumed to be instantaneous,

because in the context of a continuous-time simulation it represents a somewhat arbitrarily chosen moment of the completion of the division process. Note that both patterns of division, with the shorter cell located on the left or on the right side of the mother cell, are specified within the same production, using alternative `produce` statements (lines 107 and 109).

```
100 /* Cell division (assumed to be instantaneous in the model) */
101 Wall(wdl) < Cell(cd) > Wall(wdr) :
102 {
103    if (cd.l > MAX) {                      // if cell length exceeds limit
104       CellData shorter = cd, longer = cd; // daughters inherit mother's state
105       shorter.l = S; longer.l = L;        // ... except for length
106       if (wdl.age > wdr.age)              // wall age sets division polarity
107          produce Cell(shorter) Wall(iwd) Cell(longer);
108       else
109          produce Cell(longer) Wall(iwd) Cell(shorter);
110    }
111 }
```

The model is completed by interpretation rules that specify the visual output of the simulations. Snapshots of the simulation are shown in Figure 5.

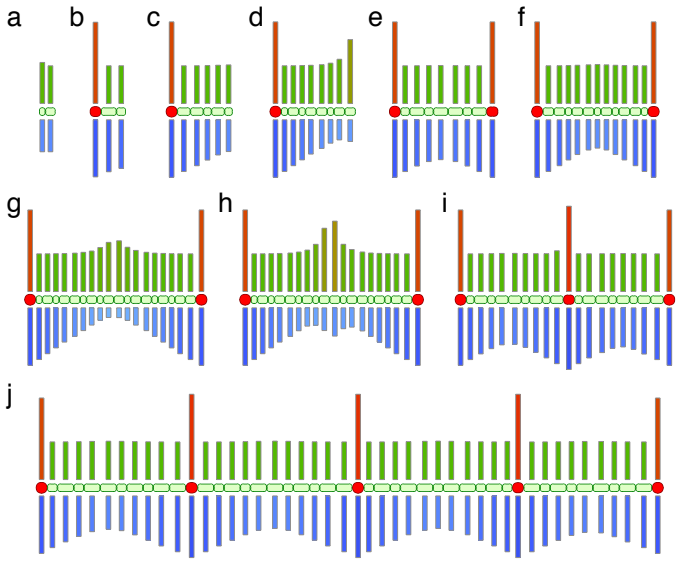

Fig. 5: Simulation of the development of an *Anabaena* filament with vegetative cells (green) and heterocysts (red). Bars above cells indicate the concentrations of HetR, and bars below indicate the concentrations of PatS, both on a logarithmic scale. The simulation begins with a short assembly of vegetative cells (**a**), in which a heterocyst quickly differentiates (**b**). As the filament grows and vegetative cells further divide (**c**), the concentration of PatS decreases away from this heterocyst (**d**), which eventually triggers the differentiation of the second heterocyst (**e**). This process repeats, producing a growing filament in which the average distance between heterocysts is approximately 10 cells (**f–j**). Note that vegetative cells distant to previously formed heterocysts may compete to become a heterocyst (**g**,**h**), until one of them eventually wins (**i**). This competitition increases the robustness of heterocyst patterning by reducing the likelihood of heterocysts differentiating next to each other (Wilcox et al, 1973).

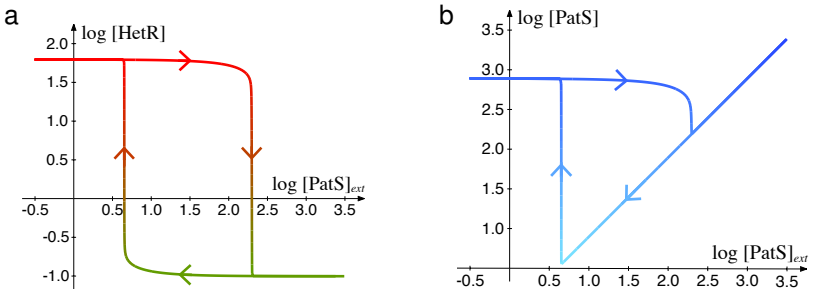

Fig. 6: Response of a single *Anabaena* cell to changing concentration of external PatS. Arrows indicate directions of changes. As the concentrations of external PatS decrease, the cell switches from the vegetative state characterized by low concentration of HetR (**a**) and PatS (**b**) to the heterocyst state characterized by high concentration of both proteins. It then remains in the heterocyst state in spite of the subsequent increases in $[\text{PatS}]_{ext}$.

An intriguing aspect of the development of an *Anabaena* filament, highlighted by the simulation, is that the concentration of PatS — an inhibitor of heterocyst differentiation — is highest in the heterocysts themselves. This observation leads to the question of why heterocysts do not inhibit themselves from being heterocysts: a problem known as the refractory behaviour of heterocysts (Adams and Duggan, 1999; Gerdtzen et al, 2009). An explanation is given in Figure 6, which shows how concentrations of HetR and PatS change within a single cell when the concentrations of PatS external to it ($[\text{PatS}]_{ext}$) decrease or increase. The plot reveals a bistable region in which the concentrations of HetR and PatS can both be low or high for the same value of $[\text{PatS}]_{ext}$, depending on the history of the system. This form of bistable behaviour (hysteresis) is the key to maintaining the heterocyst state: once the switch from the vegetative to the heterocyst state has occurred, the cell remains in the heterocyst state in spite of subsequent increases in $[\text{PatS}]_{ext}$.

5 Geometric interpretation of L-systems

In the examples considered so far we have not discussed the geometric aspects of generated structures. This follows the spirit of the original definition of L-systems, which was focused on the topology of organisms — the pattern of connections between cells or larger modules — rather than their size and shape. Nevertheless, the incorporation of geometry greatly extends the modeling power of L-systems. Different geometric interpretations of L-system-generated strings have been proposed. Among them, an interpretation based on the notion of turtle geometry (Abelson and diSessa, 1982; Papert, 1980) turned out to be particularly useful in diverse implications, including developmental modeling of plants (Prusinkiewicz, 1986; Prusinkiewicz and Lindenmayer, 1990). In this interpretation, a number of predefined L-system symbols act as commands that control a three-dimensional cursor

— conceptualized as a "turtle" — that moves in space and traces the skeleton of a branching structure. The key commands to which the turtle responds are listed in Table 3 and further illustrated in Figure 7. Many other operations are also useful in plant modeling, for examples see Algorithmic Botany (2018).

Turtle command	Symbol	L+C keyword
draw line segment	F	`F`
move without drawing a line	f	`f`
turn left | right	$+$ | $-$	`Left` | `Right`
bend up | down	$\wedge$ | &	`Up` | `Down`
roll to the left | right	\ | /	`RollL` | `RollR`
start | end branch	[|]	`SB` | `EB`
set line width	#	`SetWidth`
set line color	,	`SetColor`

Table 3: Basic turtle commands in a symbolic notation and in the L+C language.

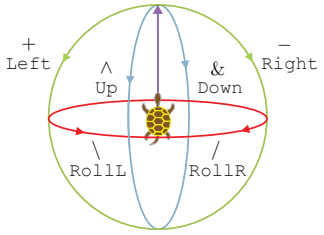

Fig. 7: Specification of turtle rotations in three dimensions.

For example, the following L-system with turtle interpretation generates the developmental sequence of branching structures shown in Figure 8.

Axiom:		$\#(0.4)A$
Productions:	p_1 :	$A \rightarrow I(1)[+A][-A]I(1)A$
	p_2 :	$I(s) \rightarrow I(2*s)$
Interpretation rules:	h_1 :	$A \rightarrow ,(1)f(0.2)F(0.8)$
	h_2 :	$I(s) \rightarrow ,(2)f(0.2)F(s-0.2)$

Table 4: L-system modeling the development of the branching structure in Figure 8.

In this example, all branching angles are assumed to have a globally specified magnitude of 45°. In contrast, the length of internode segments I is specified as a numerical parameter s. Production p_1 sets its initial value to 1, whereas production p_2 multiplies it in each simulation step by 2, thus simulating the growth (exponential elongation) of the internodes.

The L-system in Table 4 also includes two *interpretation* rules, h_1 and h_2. The interpretation rules (Kurth, 1994; Prusinkiewicz et al, 2000; Karwowski and Prusinkiewicz, 2003) are similar to decomposition rules, discussed previously, in that they are applied immediately after ordinary productions, in the same simulation step. In contrast to decomposition, however, the successor of an interpretation rule is substituted for the predecessor only temporarily, for the purpose of calculating the geometric, graphical representation of the predecessor symbol. In Table 4, the interpretation rules state that both the apices A and internodes I are visualized as line segments F preceded by short invisible segments f. The invisible segments highlight the composition of the branching structure by separating its individual components. The modules A and I continue to exist as the predecessor of productions p_1 and p_2 in the subsequent simulation steps.

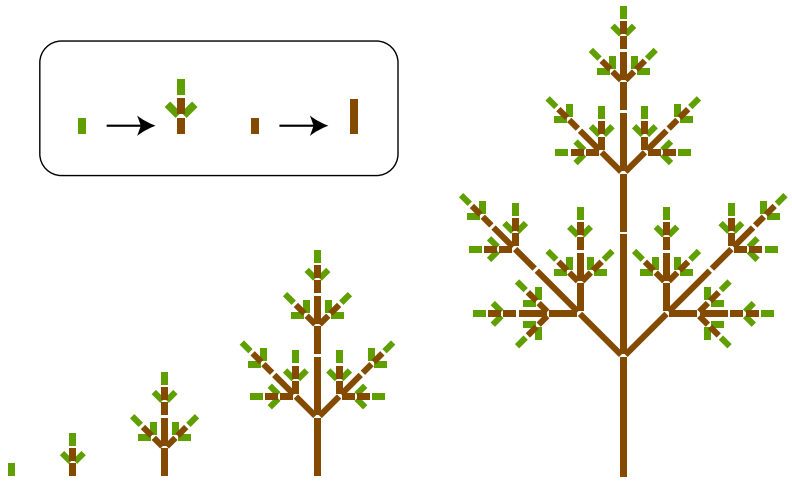

Fig. 8: The first five stages of the developmental of a simple branching structure modeled using the L-system in Table 4. The inset shows a graphical representation of the productions.

An L+C program equivalent to the symbolic notation in Table 4 is shown below:

```
module A;          // apex
module I(float);  // internode (length)

Axiom: SetWidth(0.4) A;

A :    produce I(1) SB Left(45) A EB SB Right(45) A EB I(1) A;

I(s) : produce I(2*s);

interpretation:
A    : produce SetColor(1) f(0.2) F(0.8);
I(s) : produce SetColor(2) f(0.2) F(s-0.2);
```

6 Descriptive modeling of plant architecture

Bracketed parametric L-systems with turtle interpretation are well suited to model plant development at the architectural level. In this case, L-system modules do not correspond to individual cells, but represent higher-level plant components, such as apices (apical meristems), internodes, leaves, flowers and fruits (Room et al, 1994). Architectural models include both descriptive and mechanistic models. Descriptive models recreate plant development according to direct observations and measurements of growth, whereas mechanistic models aim at simulating and explaining development in terms of the underlying lower-level biological processes. In both cases, hypothetical parameter values can be used to test predictions, fill gaps in data, or explore the range of forms that the model can theoretically generate. Below we

Fig. 9: A photograph of a mature *Lychnis coronaria* plant. The inset shows the characteristic sympodial branching pattern. The architecture of the whole plant results from iterating this motif.

present a model of the perennial herbaceous plant *Lychnis coronaria* as an example of a descriptive model. The appeal of *Lychnis* stems from the contrast between the apparent complexity of the mature plant (Figure 9) and the simplicity of its architectural development. Its key feature is repetitive sympodial branching, in which each axis is terminated by a flower, and the main thrust of development is transferred to a pair of lateral branches (Figure 9, inset).

The entire model has approximately 160 lines of L+C code, thus we limit the model description to the key constructs. The main modules are declared as follows:

```
1 module A(float);              // apex (age)
2 module I(float,float,float);  // internode (target length, width, age)
3 module L(float);              // leaf (age)
4 module K(float);              // bud/flower/fruit (age)
5 module B(float);              // branching point (age)
```

The first four modules represent components of the plant. The fifth one, `B`, represents branching points, and is introduced to specify time-varying branching angles. All modules are parametrized by age `t`, measured with respect to the time of their creation.

In each simulation step the age of each module is advanced by increment `dt`, which can be made arbitrarily small (as in the model of *Anabaena* with heterocysts) to simulate development in continuous time. The age of apices, leaves, flowers, and branching points is advanced by simple productions:

```
82 A(t) : produce A(t+dt);
83 L(t) : produce L(t+dt);
84 K(t) : produce K(t+dt);
85 B(t) : produce B(t+dt);
```

The age of internodes is advanced by more complex productions that also update the width of internodes and are discussed later.

The sympodial branching structure results from the activity of shoot apices, which give rise to a flower in the terminal position and create a pair of new lateral apices. This process is modeled by the following decomposition rule:

```
95  decomposition:
96  A(t):
97  {
98     if (t > 0 && T < BRANCHING_ENDS) produce
99        I(LEN1,0,t)
100       SB RollR(PHI) B(t) SB L(t) EB A(D1+t) EB
101       SB RollL(PHI) B(t) SB L(t) EB A(D2+t) EB
102       I(LEN2,0,t) K(t);
103 }
```

According to this production, an apex reaching the threshold age of zero creates an internode `I`, a pair of lateral apices `A` in the axils of leaves `L`, and a flower bud `K` subtended by another internode `I`. The age values t associated with the active apices are always negative: they indicate the time leading to the production of lateral branches. The branches are initiated after unequal delays `D2<D1<0`, resulting in an asymmetric branchning structure (cf. Figure 9, inset). The pattern of repetitive production of lateral apices and branches repeats until the overall plant age represented by a global variable `T` (also incremented by `dt`, statement not shown) reaches the threshold age `BRANCHING_ENDS`.

The apex also creates two modules `B`, which specify the magnitude of the branching angles via the interpretation rule:

```
105 interpretation:
106 B(t) : produce Down(br_angle(t));   // branching angle changes over time
```

According to this rule, in each simulation step module `B` will turn a lateral branch down by the modeler-specified function of time `br_angle(t)`. Meanwhile, module `B` will continue to exist as the object of the time-advancing production in line 85. The development of leaves, flowers and fruits (pods) is modeled in a similar way. For example, leaves are modeled using the production:

```
120 L(t) :
121 {
122    produce Down(leaf_angle(t)) RollToVert() CurrentTexture(LEAF_TEXTURE)
123            SetColor(leaf_color(t)) Surface(LEAF_SURFACE, leaf_length(t));
124 }
```

The turtle command `Surface(LEAF_SURFACE, leaf_length(t))` visualizes a leaf as the surface identified by its name. The shape of this surface is predefined by the modeler using an interactive editor, and the size is determined by the scaling function `leaf_length(t)`, which allows for a coarse approximation of growth over time. (More powerful techniques for modeling organ growth include anisotropic scaling capable of capturing allometric dependencies of organ proportions on size (Huxley, 1924, 1932; Niklas, 1994, 2004), and interpolation between shapes modeled at key developmental stages (Hanan, 1992; Prusinkiewicz et al, 1993; Owens et al, 2016).) The remaining modules specify the orientation, texture and color of the leaf.

The above description highlights the need for defining numerous constants and functions as a part of the model. In principle, the constants could be defined directly

in the L-system code, as in lines 1–18 of the *Anabaena* example, and the functions could be given by mathematical expressions, for example resulting from statistical data analysis. For the sake of model conceptualization, presentation and manipulation it is useful, however, to organize and describe its input graphically. A timeline editor — a concept borrowed from computer animation — can be applied for this purpose. A screenshot of a timeline editor open on the *Lychnis* model is shown in Figure 10. A modeler can use it to interactively modify all time points and function domains, and invoke the graphical editor of specific functions.

The last element of the model is a set of the productions characterizing the internodes:

```
89 I(l,w,t) >> SB I(l1,w1,t1) EB SB I(l2,w2,t2) EB I(l3,w3,t3) :
90        produce I(l, w1+w2+w3, t+dt);
91 I(l,w,t) >> SB I(l1,w1,t1) EB I(l2,w2,t2) : produce I(l, w1+w2, t+dt);
92 I(l,w,t) >> I(l1,w1,t1) : produce I(l, w1, t+dt);
93 I(l,w,t) >> K(t1) : produce I(l, width(t), t+dt);
```

The complicated form of these productions, compared to those describing the aging of the other modules (lines 82-85), stems from the need to determine the internode diameter, which is not simply a function of age. We follow MacDonald (1983) by assuming that, at every branching point, the diameter d_0 of the parent internode is related to the diameters d_1, d_2 and d_3 of the child internodes by equation

$$d_0^\eta = d_1^\eta + d_2^\eta + d_3^\eta. \tag{4}$$

To satisfy this equation, we assume that the diameter of the terminal internodes (supporting a bud, flower or fruit `K`) is determined directly as a function of their age (line 93), whereas the diameter of the remaining internodes is calculated by setting the width measure $w = d^\eta$ in the parent internode to the sum of measures w_i in the supported internodes. The productions in lines 89–92 capture this propagation in the case of three, two and one supported internodes. The symbol `>>` in the production predecessors indicates fast information transfer, a variant of context sensitivity that makes it possible to propagate information in one direction (from the extremities of a branching structure towards the base or vice versa) in a single derivation step (Karwowski and Prusinkiewicz, 2003). The basipetal propagation of width information is a version of the pipe model (Shinozaki et al, 1964), which relates the diameter of each branch in a tree to the number of leaves it supports. The specification of internodes is complemented by the interpretation rule:

```
111 I(l,w,t) :
112 {
113    nproduce SetColor(br_color(T)) SetWidth(pow(w, 1/eta));
114    for (int i=0; i < N_SEG; i++)
115       nproduce F(l*int_length(t) / N_SEG);
116    produce;
117 }
```

The loop in lines 114 and 115 divides the internode into `N_SEG` segments of length `l*int_length(t)`. This division makes it possible to simulate gravitropism by slightly reorienting the turtle towards the vertical at each junction between consecutive segments (Prusinkiewicz and Lindenmayer, 1990). (A more advanced model of

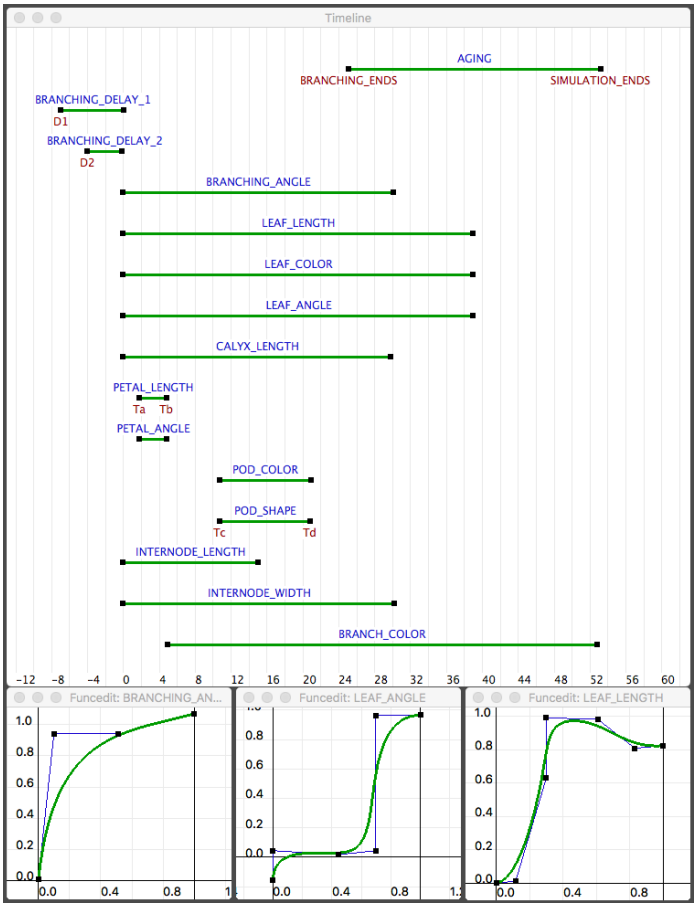

Fig. 10: A snapshot of the timeline editor controlling the *Lychnis* model. In addition to the function domains (green lines with blue labels above) and time points (black squares with red labels below) related to branching and leaf development discussed in the text, the editor controls the development of the flowers, fruits (pods) and internodes. The timeline editor also makes it possible to invoke a function editor, here open on three sample functions shown at the bottom. The default domain $[0,1]$ of each function is mapped onto the domain indicated by its timeline, e.g. $[0,30]$ in the case of `BRANCHING_ANGLE`. Outside their default domains the functions are assumed to be constant: $f(x) = f(0)$ for $x < 0$, and $f(x) = f(1)$ for $x > 1$. Note that, within the *Lychnis* L-system, the `AGING` and `BRANCH_COLOR` timelines and their associated time points are defined over the whole lifetime `T` of the plant, whereas the remaining timelines and points are defined relative to the age `t` of the modules they describe. This difference is not visualized by the editor.

tropisms is presented by Bastien et al (2013), commented on by Dumais (2013) and expanded by Bastien et al (2015); Chelakkot and Mahadevan (2017).) The `nproduce` statement used in lines 113 and 115 differs from the `produce` statement introduced previously in that it does not terminate the production application, making it possible to successively append modules to the successor (Karwowski and Prusinkiewicz, 2003). The productions in line 113 sets the internode color ac-

cording to plant age `T` (a global variable) and determines the internode diameter as a function of the width measure `w` using the equation $d = w^{1/\eta}$.

The operation of the complete *Lychnis* model is illustrated in Figure 11. A comparison with Figure 9 shows that the model correctly captures the architecture of *Lychnis coronaria* plants, although it does not reach branch density of a plant grown over many seasons. Modeling of dense plant structures requires incorporating collisions between plant organs and is a topic of current research (Owens et al, 2016).

Descriptive models have many applications. For example, they can be used as a synthetic representation of our knowledge of plant form and development (Prusinkiewicz et al, 1994b; Fournier and Andrieu, 1998; Mündermann et al, 2005), as a vehicle for exploring the range of forms that plants in a given class can potentially attain (McGhee, 1999) (such explorations may find practical applications in plant breeding), or as the source of ground truth for training artificial intelligence programs intended to automatically recognize plant traits from images (Ubbens et al, 2018). A distinctive feature of plant development is, however, their phenotypic plasticity: the ability of plants with the same genotype to assume different forms depending on the environment in which they grow. We describe the incorporation of environment into L-system plant models next.

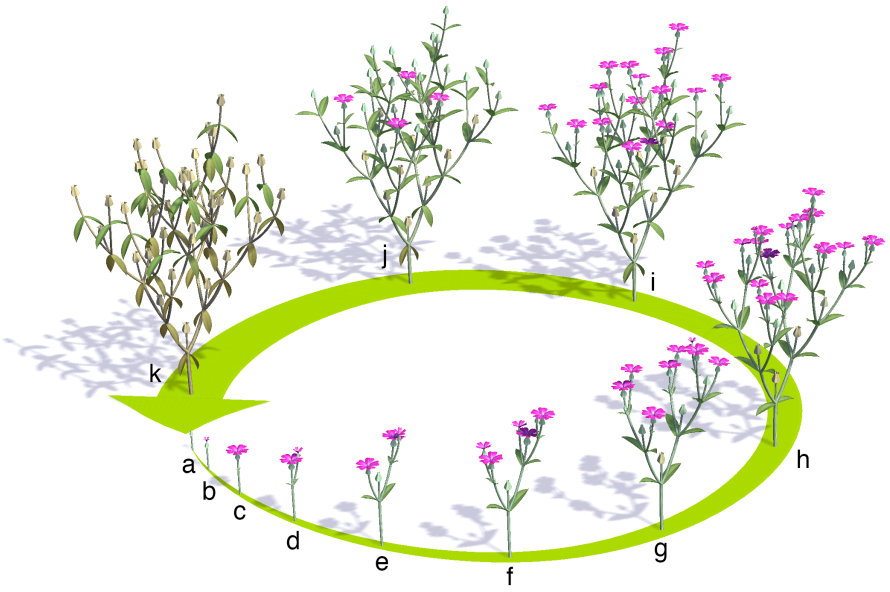

Fig. 11: Selected steps in the development of a *Lychnis coronaria* shoot. In the initial shoot axis (**a**,**b**,**c**), lateral apices supported by leaves initiate first one (**d**), then another (**e**) lateral branch. In the meantime, the apex terminating the main axis undergoes a transition from the vegetative to the flowering stage, producing a flowering bud (**b**) that gives rise to an open flower (**c**–**e**) and, eventually, matures into a seed pod (**f**–**k**). This branching pattern repeats, producing branches of increasingly high order (**e**–**h**). Upon reaching a threshold age (simulating the end of the growing season), further branching stops (**i**), and the plant gradually reaches maturity (**j**,**k**). Its seeds give rise to the next generation of plants (**a**).

7 Modeling phenotypic plasticity

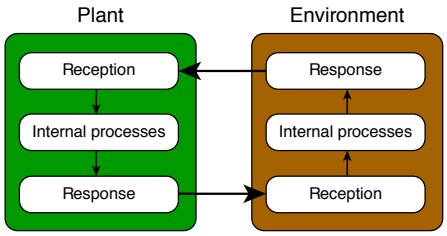

Fig. 12: Information flow in a simulation of a plant interacting with its environment. The plant and the environment are modeled by separate communicating programs.

Irrespective of the diversity of environmental factors that affect plant development, including space, light, water, nutrients, and pathogens, a unified method for modeling plants in the context of their environment is available. The idea is to model the plant and its environment as separate programs that run concurrently and communicate using standardized programming constructs (Měch and Prusinkiewicz, 1996) (Figure 12). In general, this communication is bilateral — the environment affects the plant while the plant reciprocally affects the environment — but simple models may focus on the unilateral influence of the environment on the plant.

An example of such influence is the dependence of the growth rate on temperature. Within some range, this rate is known to be proportional to the difference between the actual temperature τ and a base temperature τ_{base} characteristic of a given plant species, below which growth does not occur (Hanan, 1997; Wardlaw, 1999). For instance, Figure 13 shows forms created by the *Lychnis* model extended to include the influence of temperature, under the assumption that the temperature of each organ is determined by its position within the canopy. The plant shape is affected by temperature distribution in the environment in which it grows.

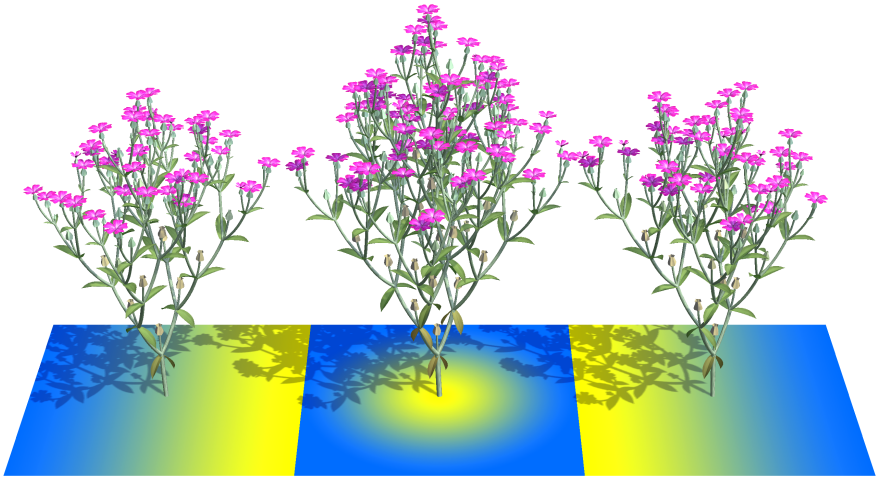

Fig. 13: A model of phenotypic plasticity. The simulated *Lychnis* plant grows relatively faster in regions with higher temperatures. Temperature distribution in the plant canopy is defined by the heat map laid on the ground, where yellow indicates warmer regions and blue indicates colder.

Specifically, the environmental program returns temperature as defined by an image (texture map): points above yellow regions are warmer than points above blue regions (Figure 13). Information between the plant and its environment is exchanged using predefined communication modules (Prusinkiewicz et al, 1994a; Měch and Prusinkiewicz, 1996). In the L+C language, these modules are named `E1`, `E2`, `E3`, etc., depending on the number of parameters they carry. In the *Lychnis* model only the `E1` module is used. First, it is inserted into the L-system string at the locations at which the temperature will be queried. This is accomplished by modifying the decomposition rule that controls *Lychnis* branching (lines 95–103 in the original *Lychnis* model) as follows:

```
95  decomposition:
96  A(t):
97  {
98     if (t > 0 && T < BRANCHING_ENDS) produce
99        E1(0) I(LEN1,0,t)
100       SB RollR(PHI) E1(0) B(t) SB E1(0) L(t) EB E1(0) A(D1+t) EB
101       SB RollL(PHI) E1(0) B(t) SB E1(0) L(t) EB E1(0) A(D2+t) EB
102       E1(0) I(LEN2,0,t) E1(0) K(t);
103 }
```

The value (`0`) of the parameter passed to the communication modules `E1` is here irrelevant: there is no information transferred from the plant to the environment other than the location of each module `E1`, which is passed on automatically. This parameter is needed, however, as a placeholder for the temperature information that the environmental program returns to the plant model. With the local temperatures known, the dependence of plant growth on temperature is captured by replacing constant time increments `dt` with the increments `DD(temp)*dt`, a product of the temperature above the threshold measured in `D`egree `D`ays, and the time increment. For instance, for apices and leaves this dependence is simulated be replacing lines 82 and 83 in the original *Lychnis* model with productions:

```
82  E1(temp) < A(t) : produce A(t+DD(temp)*dt);
83  E1(temp) < L(t) : produce L(t+DD(temp)*dt);
```

The advancement of time in the remaining modules is controlled in a similar manner, resulting in the simulation of plastic behavior shown in Figure 13.

8 Modeling the development of trees

In a recursive branching pattern with all internodes of approximately the same length, the number of branches increases exponentially, while the canopy radius — the reach of branches — grows only linearly with time. Eventually, there is not enough room in the canopy for all these branches (Borchert and Slade, 1981; Prusinkiewicz and de Reuille, 2010). The shortage of space is a critical factor in the development of trees due to the high number of branches they could potentially produce during their long life. Borchert and Honda (1984), and Sachs and Novoplansky (1995) proposed that the necessary limitation of branch proliferation in trees results

from their competition for space or light. This competition is not merely a manifestation of tree plasticity, but an essential component of the development in most trees (we exclude here non-branching palms and tree ferns, for example) (Palubicki et al, 2009). Below we show how such competition can be implemented with L-systems, by simulating bilateral communication between the plant and its environment.

The key components of the model and the initial state of the simulation are specified as follows:

```
struct internode_data {
   float l;     // internode length
   float w;     // internode width
   int age;     // needed to know when to shed
};

module A(int);              // apex (0: dead, 1: alive)
module I(internode_data);   // internode

internode_data trunk_init = {1.0, w_init, 0};

Axiom: I(trunk_init) A(1) E1(1) GetHead(0,0,0);
```

The axiom describes the initial configuration of the simulated structure as an apex `A` supported by an internode `I`. The apex is followed by two modules, `E1` and `GetHead`. The predefined module `GetHead`, inserted with arbitrary parameter values — here $(0,0,0)$ — returns the current direction of the turtle. Its role in the model is described further down. The communication module `E1` provides an interface between the plant model and the environmental program, which in this case tests whether the apex is in proximity of other apices. (Obviously, there is no other apex at the beginning of simulation, but the same sequence of modules is produced as the plant develops and new apices are formed.) The parameter passed to module `E1` — in the axiom, it is a 1 — represents apex vigor, and is an input to the environmental program. If the distance of a given apex to all other apices (or environmental modules indicating the presence of branches) is greater than a predefined threshold d, the same module `E1` will return 1, indicating that this apex is not dominated by any other tree component. On the other hand, if this distance is less than d, the environmental program will return 1 or 0, depending on whether the given apex has the highest vigour among all the nearby apices or not. A given apex is thus not dominated if there is enough free space around it, or it has higher vigor than all the modules with which it competes for space (if two or more colliding apices have the same high vigor, all of them are considered dominated). This environmental information affects the development of the simulated tree via the following production:

```
I(sl) < A(alive) E1(not_colliding) GetHead(x,y,z) : {
   internode_data s1 = {sl.l*r1, w_init, 0};
   internode_data s2 = {sl.l*r2, w_init, 0};
   if (alive && not_colliding && y > GrowDown) {
      produce E1(1)
      SB Left(a1)  RollL(phi)     I(s1) A(1) E1(v1) GetHead(0,0,0) EB
      SB Right(a2) RollL(phi-180) I(s2) A(1) E1(v2) GetHead(0,0,0) EB;
   }
   else
      produce A(0);
}
```

According to Line 57, the production predecessor consists of three modules, `A`, `E1` and `GetHead`, in the context of a supporting internode `I`. Line 60 states that the fate of an apex depends qualitatively on three factors: whether it is "alive" (i.e., was never dominated), whether it is dominated now, and what orientation it has. An apex that has never been dominated and is not oriented too steeply down gives rise to two branches (lines 62 and 63). Their length is reduced with respect to that of the supporting internode (`sl.l`) by predefined factors `r1` and `r2` (lines 58 and 59). The apices of the new branches are followed by modules `E1` and `GetHead`, allowing for the application of the same production and repetition of the same decision process in the next simulation step. The vigor values $0 <$ `v1` $<$ `v2` < 1 are predefined constants. The additional environmental module with argument 1 (maximum vigor), introduced in line 61, marks the branching point itself to assure that no apex will ever grow into a previously formed branch. The modules specified in lines 61–63 are not produced, however, if the condition in line 60 is not satisfied. In that case, the apex changes its state to "dead" (`A(0)` in line 66), and it will produce no further branches.

The effect of the detection and response to overcrowding is illustrated by Figure 14. The branching structure in Figure 14a was generated by a model in which detection of overcrowding was disabled (the condition in line 60 was replaced by the always-true statement `if(1)`). The produced structure is then exceedingly dense, with apices and branches running into each other and overlapping. In contrast, the structure in Figure 14b was generated with the condition in line 60 present. Predictably, it is much more sparse: overcrowding has been prevented.

In nature, branches that have ceased growing are often shed by the tree. A simple instance of this process is illustrated in Figure 14c. The shedding criterion used here is the age of a non-growing branch: the time since the last apex supported by it has become dominated. Shedding is implemented by the following production and decomposition rules:

```
77 production:
78 I(s) >> A(alive) : // advance branch age if apex is dead
79 {
80    if (!alive)
81       s.age += 1 ;
82    produce I(s);
83 }
84
85 I(s) >> SB I(sr) EB SB I(srr) EB :  // propagate girth and age
86 {
87    s.w = sr.w + srr.w;
88    s.age = min(sr.age, srr.age);
89    produce I(s);
90 }
91
92 decomposition:
93 I(s) :   // tag branch for shedding if too old
94 {
95    if(s.age >= ShedAge)
96       produce I(s) Cut();
97 }
```

The first production (lines 78–83) increments the age of an internode followed by a dominated apex. The second production (lines 85–90) propagates the age informa-

tion basipetally. When branches that meet at a given branching point have different age values, the supporting internode is assigned the smaller value of the two (line 88); consequently, each internode "knows" the age of the youngest internode it supports. The actual shedding is implemented by the decomposition rule in lines 93–97. When the age of an internode exceeds threshold `ShedAge`, this rule inserts a predefined module `Cut`, which (in the next simulation step) removes the entire branch that follows it.

The production in lines 85–90 also includes the assignment in line 87 that propagates a measure of the internode diameter according to the pipe model as discussed for *Lychnis*. Proper modeling of the girth of branches is an important element of the visually realistic modeling of trees. Remarkably, the inclusion of these two principles: the competition for space combined with shedding (extended from two to three dimensions by changing the phyllotaxis-defining parameter `phi` from 0 to 90° in lines 62 and 63) and the assignment of branch width according to the pipe model suffice to generate a visually plausible tree architecture (Figure 14d,e) resembling *Dracaena draco* (the dragon tree).

The use of separate programs to model the plant and its environment (Figure 12) facilitates simulation of different plants in the same environment or, conversely, the same plant in different environments. For instance, Figure 14f shows the result of substituting competition for light for competition for space in the simulation of tree development. The required modification of the tree model itself was limited to a single production in lines 57–67 (compare the original listing with the code below):

```
57 I(sl) < A(alive) E1(light) GetHead(x,y,z) : {
58    internode_data s1 = {sl.l*r1, w_init, 0};
59    internode_data s2 = {sl.l*r2, w_init, 0};
60    if (alive && light>Th && y > GrowDown) {
61       produce
62       SB Left(a1)  RollL(phi)     I(s1) A(1) E1(R) GetHead(0,0,0) EB
63       SB Right(a2) RollL(phi-180) I(s2) A(1) E1(R) GetHead(0,0,0) EB;
64    }
65    else
66       produce A(0);
67 }
```

The module `E1` passes the size of leaf clusters to the environment (predefined constant `R` in lines 63 and 64) and receives information about the intensity of light reaching each cluster in return (variable `light` in line 57). An apex is not dominated and can produce new branches (provided that the remaining conditions in line 60 are satisfied) if this intensity is greater than a predefined threshold `Th` (condition `light>Th`). Note the absence of module `E1` at the branching point of the light-driven model (compare lines 61 in both listings): this module is now not needed, because there are no leaf clusters at the branching points. A comparison of Figures 14d and f shows that the tree forms generated assuming competition for space or competition for light may substantially differ, even though the underlying tree model is basically the same. Not surprisingly, the environment has a significant impact on tree development and must be modeled carefully when a faithful simulation of develpmental processes in nature is sought.

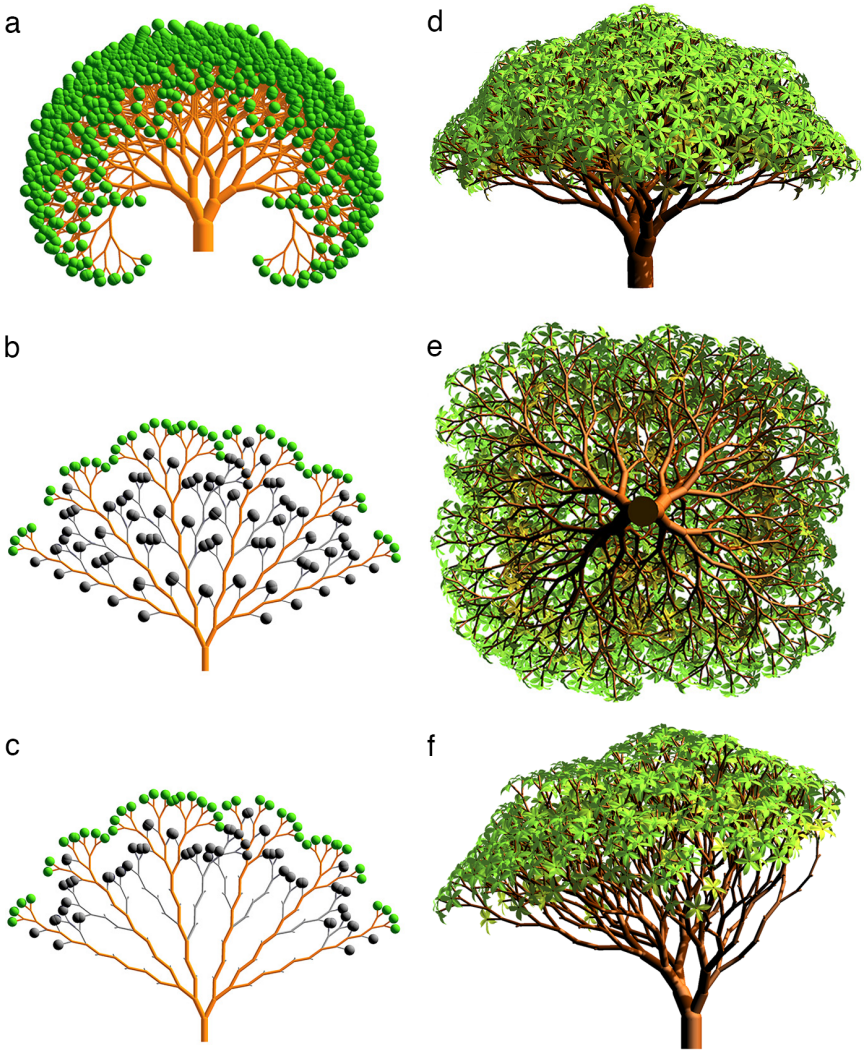

Fig. 14: Elements of tree modeling. **a**) A planar structure developing with branch proliferation unchecked. The "tree" crown is clearly overcrowded. **b**) A modification of the tree from Figure **a**, in which dominated branches die. Dead apices and internodes are shown in grey. The structure is no longer overcrowded. **c**) A modification of the model from Figure **b**, in which older dead branches are shed. **d**) A three-dimensional modification of the model from Figure **d**, in which phyllotaxis has been changed from distichous ($\varphi = 0$) to decussate ($\varphi = 90^\circ$), and leaves grow at the tips of branches. The difference in the diameter of branches results from the application of the pipe model. **e**) The tree from panel d seen from the bottom. Note the plausible architecture resulting from the competition of branches for space. **f**) A variant of the model from Figure **d**, in which the environmental program simulates competition for light rather than for space. A light source is positioned above and to the left of the tree, resulting in an asymmetric crown, leaning to the left.

9 Conclusion

In the 50 years since its inception, the formalism of L-systems has become a powerful research tool in developmental biology. Remarkably, it was not imported from another discipline, but created specifically for developmental biology. With their many extensions, L-systems can be and have been applied to an ever-increasing variety of problems spanning a wide range of scales, from sub-cellular to whole plants and plant ecosystems (Lane and Prusinkiewicz, 2002), as well as modeling styles, from descriptive to mechanistic. The key limitation of L-systems is their restriction to linear (filamentous) and branching structures. Nevertheless, the topological approach to modeling development, inherent in L-systems, has also inspired research and the establishment of practical methods for modeling growing cellular layers and volumetric structures (Prusinkiewicz and Lindenmayer, 1990; Smith, 2006; Lane, 2015). L-systems, their extensions, and applications continue to be an active and fascinating area of research.

Acknowledgements We thank John Hall for the prototype version of the timeline editor featured in Figure 10, Andrew Owens for help with Figure 11, and Lynn Mercer for insightful discussions. The authors' research on the L-system-based modeling methods, specific models, and the Virtual Laboratory (`vlab`) software used in the preparation of this paper was supported by the Natural Sciences and Engineering Research Council and the Plant Phenotyping Imaging and Research Centre / Canada First Research Excellence Fund. This support is gratefully acknowledged.

References

Abelson H, diSessa AA (1982) Turtle Geometry. M.I.T. Press, Cambridge

Adams D, Duggan P (1999) Heterocyst and akinete differentiation in cyanobacteria. New Phytologist 144(1):3–33

Algorithmic Botany (2018) The Virtual Laboratory / L-studio software distribution. http://algorithmicbotany.org/virtual_laboratory

Baker R, Herman GT (1970) CELIA — a cellular linear iterative array simulator. In: Proceedings of the Fourth Conference on Applications of Simulation (9–11 December 1970), pp 64–73

Baker R, Herman GT (1972) Simulation of organisms using a developmental model, Parts I and II. International Journal of Bio-Medical Computing 3:201–215 and 251–267

Bastien R, Bohr T, Moulia B, Douady S (2013) Unifying model of shoot gravitropism reveals proprioception as a central feature of posture control in plants. Proceedings of the National Academy of Sciences 110(2):755–760

Bastien R, Douady S, Moulia B (2015) A unified model of shoot tropism in plants: photo-, gravi-and propio-ception. PLOS Computational Biology 11(2):e1004,037

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

Borchert R, Slade N (1981) Bifurcation ratios and the adaptive geometry of trees. Botanical Gazette 142(3):394–401

Boudon F, Pradal C, Cokelaer T, Prusinkiewicz P, Godin C (2012) L-Py: an L-system simulation framework for modeling plant architecture development based on a dynamic language. Frontiers in Plant Science 3:76

Buikema W, Haselkorn R (1991) Characterization of a gene controlling heterocyst differentiation in the cyanobacterium *Anabaena* 7120. Genes & Development 5(2):321–330

Chelakkot R, Mahadevan L (2017) On the growth and form of shoots. Journal of The Royal Society Interface 14(128):1–6

Coen E, Rolland-Lagan AG, Matthews M, Bangham A, Prusinkiewicz P (2004) The genetics of geometry. Proceedings of the National Academy of Sciences 101:4728–4735

de Koster CG, Lindenmayer A (1987) Discrete and continuous models for heterocyst differentiation in growing filaments of blue-green bacteria. Acta Biotheoretica 36:249–273

Dumais J (2013) Beyond the sine law of plant gravitropism. Proceedings of the National Academy of Sciences 110(2):391–392

Fogg G (1949) Growth and heterocyst production in *Anabaena cylindrica* Lemm. in relation to carbon and nitrogen metabolism. Annals of Botany 13(51):241–259

Fournier C, Andrieu B (1998) A 3D architectural and process-based model of maize development. Annals of Botany 81:233–250

Fritsch F (1951) The heterocyst: A botanical enigma. Proceedings of the Linnean Society of London 162(2):194–211

Gerdtzen Z, Salgado J, Osses A, Asenjo J, Rapaport I, Andrews B (2009) Modeling heterocyst pattern formation in cyanobacteria. BMC Bioinformatics 10(6):S16

Giavitto JL, Godin C, Michel O, Prusinkiewicz P (2002) Computational models for integrative and developmental biology. LaMI Rapport de Recherche 72-2002, CNRS — Université d'Evry val d'Essonne

Gierer A, Meinhardt H (1972) A theory of biological pattern formation. Kybernetik 12:30–39

Hammel M, Prusinkiewicz P (1996) Visualization of developmental processes by extrusion in space-time. In: Proceedings of Graphics Interface '96, pp 246–258

Hanan JS (1992) Parametric L-systems and their application to the modelling and visualization of plants. PhD thesis, University of Regina

Hanan JS (1997) Virtual plants — integrating architectural and physiological models. Environmental Modeling & Software 12:35–42

Haselkorn R (1998) How cyanobacteria count to 10. Science 282:891–892

Herrero A, Stavans J, Flores E (2016) The multicellular nature of filamentous heterocyst-forming cyanobacteria. FEMS Microbiology Reviews 40(6):831–854

Huxley JS (1924) Constant differential growth ratios and their significance. Nature 114:895–896

Huxley JS (1932) Problems of Relative Growth. MacVeagh, London

Karwowski R, Prusinkiewicz P (2003) Design and implementation of the L+C modeling language. Electronic Notes in Theoretical Computer Science 86(2):134–152

Kniemeyer O (2004) Rule-based modelling with the XL/GroIMP software. In: The Logic of Artificial Life: Abstracting and Synthesizing the Principles of Living Systems; Proceedings of the 6th German Workshop on Artificial Life, April 14-16, 2004, Bamberg, Germany, AKA Akademische Verlagsgesellschaft, Berlin, pp 56–65

Kniemeyer O, Buck-Sorlin G, Kurth W (2007) GroIMP as a platform for functional-structural modelling of plants. In: Vos J, et al (eds) Functional-Structural Modeling in Crop Production, Springer, Dordrecht, pp 43–52

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 Waldökosysteme der Universität Göttingen, Göttingen

Lane B (2015) Cell complexes: The structure of space and the mathematics of modularity. PhD thesis, University of Calgary

Lane B, Prusinkiewicz P (2002) Generating spatial distributions for multilevel models of plant communities. In: Proceedings of Graphics Interface 2002, pp 69–80

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

Lindenmayer A (1971) Developmental systems without cellular interaction, their languages and grammars. Journal of Theoretical Biology 30:455–484

Lindenmayer A (1974) Adding continuous components to L-systems. In: Rozenberg G, Salomaa A (eds) L Systems, Lecture Notes in Computer Science 15, Springer-Verlag, Berlin, pp 53–68

MacDonald N (1983) Trees and Networks in Biological Models. J. Wiley & Sons, New York

MacNamara S, Strang G (2016) Operator splitting. In: Glowinski R, Osher S, Yin W (eds) Splitting Methods in Communication, Imaging, Science, and Engineering, Springer, Berlin, pp 95–114

McGhee G (1999) Theoretical Morphology: The Concept and its Applications. Columbia University Press

Meinhardt H (1982) Models of Biological Pattern Formation. Academic Press, London

Mitchison G, Wilcox M (1972) Rules governing cell division in *Anabaena*. Nature 239:110–111

Mündermann L, Erasmus Y, Lane B, Coen E, Prusinkiewicz P (2005) Quantitative modeling of Arabidopsis development. Plant Physiology 139:960–968

Měch R, Prusinkiewicz P (1996) Visual models of plants interacting with their environment. In: Proceedings of SIGGRAPH 1996, pp 397–410

von Neumann J (1966) Theory of Self-reproducing Automata. University of Illinois Press, Urbana, edited by A. W. Burks

Niklas KJ (1994) Plant Allometry: The Scaling of Form and Process. The University of Chicago Press, Chicago

Niklas KJ (2004) Plant allometry: Is there a grand unifying theory? Biological Reviews 79(4):871–889

Owens A, Cieslak M, Hart J, Classen-Bockhoff R, Prusinkiewicz P (2016) Modeling dense inflorescences. ACM Transactions on Graphics 35(4):136

Palubicki W, Horel K, Longay S, Runions A, Lane B, Měch R, Prusinkiewicz P (2009) Self-organizing tree models for image synthesis. ACM Transactions on Graphics 28:58

Papert S (1980) Mindstorms: Children, Computers and Powerful Ideas. Basic Books, New York

Prusinkiewicz P (1986) Graphical applications of L-systems. In: Proceedings of Graphics Interface '86 — Vision Interface '86, pp 247–253

Prusinkiewicz P (2004) Art and science for life: Designing and growing virtual plants with L-systems. Acta Horticulturae 630:15–28

Prusinkiewicz P, Hanan J (1990) Visualization of botanical structures and processes using parametric L-systems. In: Thalmann D (ed) Scientific Visualization and Graphics Simulation, J. Wiley & Sons, Chichester, pp 183–201

Prusinkiewicz P, Lane B (2013) Modeling morphogenesis in multicellular structures with cell complexes and L-systems. In: Capasso V, et al (eds) Pattern Formation in Morphogenesis, Springer, Berlin, pp 137–151

Prusinkiewicz P, Lindenmayer A (1990) The Algorithmic Beauty of Plants. Springer-Verlag, New York, with J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer

Prusinkiewicz P, de Reuille PB (2010) Constraints of space in plant development. Journal of Experimental Botany 61:2117–2129

Prusinkiewicz P, Runions A (2012) Computational models of plant development and form. New Phytologist 193:549–569

Prusinkiewicz P, Hammel M, Mjolsness E (1993) Animation of plant development. In: Proceedings of SIGGRAPH 1993, pp 351–360

Prusinkiewicz P, James M, Měch R (1994a) Synthetic topiary. In: Proceedings of SIGGRAPH 1994, pp 351–358

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

Prusinkiewicz P, Hanan J, Měch R (2000) An L-system-based plant modeling language. In: Nagl M, Schürr A, Münch M (eds) Applications of Graph Transformations with Industrial Relevance, Lecture Notes in Computer Science 1779, Springer-Verlag, Berlin, pp 395–410

Prusinkiewicz P, Karwowski R, Lane B (2007) The L+C plant-modeling language. In: Vos J, et al (eds) Functional-Structural Modeling in Crop Production, Springer, Dordrecht, pp 27–42

Robinson S, de Reuille PB, Chan J, Bergmann D, Prusinkiewicz P, Coen E (2011) Generation of spatial patterns through cell polarity switching. Science pp 1436–1440

Room PM, Maillette L, Hanan J (1994) Module and metamer dynamics and virtual plants. Advances in Ecological Research 25:105–157

Sachs T, Novoplansky A (1995) Tree form: Architectural models do not suffice. Israel Journal of Plant Sciences 43:203–212

Shinozaki K, Yoda K, Hozumi K, Kira T (1964) A quantitative analysis of plant form — the pipe model theory. I. Basic analyses. Japanese Journal of Ecology 14(3):97–104

Smith C (2006) On vertex-vertex systems and their use in geometric and biological modeling. PhD thesis, University of Calgary

Ubbens J, Cieslak M, Prusinkiewicz P, Stavness I (2018) The use of plant models in deep learning: an application to leaf counting in rosette plants. Plant Methods 14:6:1–10

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

Ulam S (1966) Patterns of growth of figures: Mathematical aspects. In: Kepes G (ed) Module, Proportion, Symmetry, Rhythm, Braziller, New York, pp 64–74

Wardlaw I (1999) Thermal time. In: Atwell B, Kriedemann P, Turnbull (eds) Plants in Action: Adaptation in Nature, Performance in Cultivation, Macmillan Education Australia, Melbourne

Wilcox M, Mitchison GJ, Smith RJ (1973) Pattern formation in the blue-green alga, *Anabaena*. I. Basic mechanisms. Journal of Cell Science 12:707–723

Wolfram S (1984) Universality and complexity in cellular automata. Physica D: Nonlinear Phenomena 10(1-2):1–35

Wolfram S (2002) A New Kind of Science. Wolfram Media / Cambridge University Press, Champaign, IL

Yoon HS, Golden JW (1998) Heterocyst pattern formation controlled by a diffusible peptide. Science 282:935–938