J. McCormack: "Generative Modelling with Timed L-Systems", in J. S. Gero (ed) Design Computing and Cognition '04, [Proceedings of the First International Conference on Design, Computing and Cognition (DCC'04), Cambridge, USA, 19-21 July 2004], Kluwer Academic Publishers, Dordrecht, The Netherlands, ISBN: 1-4020-2393-6, pp. 157–175.

GENERATIVE MODELLING WITH TIMED L-SYSTEMS

JON McCORMACK
Centre for Electronic Media Art,
School of Computer Science and Software Engineering
Monash University, Clayton, Australia
Email: jonmc@csse.monash.edu.au

Abstract. This paper describes a generative design system based on timed, parametric Lindenmayer systems (L-systems), developed for the continuous modeling of dynamic phenomena such as morphogenesis. The specification of development functions gives the system the ability to continuously control temporal aspects of development in conjunction with the discrete rewriting for which L-systems are commonly associated. Incorporating advanced modeling extensions, such as generalized cylinders, into the interpretation of derived strings gives the system the ability to model complex shapes and forms. Examples in the design and simulation of mechanical models, plant morphogenesis and the animation of animal gaits are provide as an insight into the flexibility provided by the system.

1. Introduction

Generative design offers new modes of aesthetic experience based on the incorporation of system dynamics into the production of artifact and experience. In the terminology of Thomas Kuhn (Kuhn 1996) it offers a 'paradigm shift' for the process of design and the expression of that process. The traditional modes of representation become unnecessary, being replaced by a meta-design process of activities, relationships, events and processes. A key feature of generative processes is one of *database amplification*, where simple sets of interacting components generate information complexity, many orders of magnitude greater than the specification.

This paper looks at a developmental model suitable for generative design, based on Lindenmayer Systems (L-systems). A temporal, developmental model is described, and its application to generative design is illustrated with examples. This model is well suited to simulating the development of a wide variety of natural and artificial phenomena.

2. L-systems

L-systems are string-rewriting formalisms, originally developed in 1968 by the biologist Lindenmayer for the purposes of modeling multicellular development. Since the original formulation, L-systems have been widely studied from a variety of perspectives, including: biological simulation, mathematical formalisms, theory of computation, artificial life, and visualization. A number of variations have been introduced adapting the basic rewriting process to specific applications. Some of these applications include: cellular interaction (Lindenmayer and Rozenberg 1979); visual modeling of plants, flowers and trees (Prusinkiewicz and Lindenmayer 1990); music composition (McCormack 1996); data compression (Nevill-Manning and Witten 1997); modeling of biological organ growth (Durikovic, Kaneda and Yamashita 1998); design of neural networks (Kitano 1990); architectural design (Coates, Broughton and Jackson 1999; Hornby and Pollack 2001a); construction of artificial creatures (Hornby and Pollack 2001b); procedural design of cities (Parish and Müller 2001). This diverse oeuvre confirms the flexibility and adaptability of L-systems as a general paradigm in generative design and modeling.

Fundamental to all varieties of L-systems is the concept of *parallel rewriting*, whereby a set of symbols are rewritten (replaced, changed) according to some set of *rules*. This rewriting process occurs over the entire set of symbols simultaneously, simulating parallel development of components, similar to the way cells develop in parallel in an organism. Due to this parallelism, L-systems differ from other grammars such as Chomsky grammars, which are rewritten sequentially.

2.1. L-SYSTEM CLASSIFICATION: AN OVERVIEW

Key classification distinctions in L-systems are made between *context sensitive* and *context free* grammars. In the context sensitive case neighbor relations are considered in deciding which rewriting rule is to be applied (Salomaa 1973). Context sensitivity can be used, for example, to implement chemical signaling or hormone propagation in cellular simulations.

Deterministic L-systems are characterized by having only one rule or *production* for each symbol in the L-system alphabet. This means that all rewriting is deterministic and the strings produced by such L-systems will be identical provided they begin with the same initial string (known as the *axiom*). *Stochastic L-systems* permit rewriting on a probabilistic basis and allow the simulation of inter- and intra-species variation from the same L-system in the case of plant modeling. They can also be used to simulate Markov models, popular in musical applications (McCormack 1996).

In certain modeling contexts, the discrete nature of L-systems makes the simulation of irrational ratios or continuous properties difficult. *Parametric L-systems* solve this problem by associating real-valued parameters with symbols (collectively referred to as a *module*) and permit symbol rewriting to proceed by not only matching symbols but logical and arithmetic conditions involving parameters as well (Hanan 1992; Prusinkiewicz and Hanan 1990).

The key type of L-system that will be considered here is the context free, timed, parametric L-system (*tp0L-system*), which includes a temporal component suitable for simulation of continuous development. This is described in the next section.

3. Timed, Parametric 0L-systems

Timed L-systems were proposed by Prusinkiewicz and Lindenmayer (Prusinkiewicz and Lindenmayer 1990, Chapter 6) as an extension for achieving continuous development with D0L-systems[1]. The important developments described by the research presented in this paper are:

- the incorporation of parametric and stochastic components into timed modules;
- birth age parameters (defined below) may be expressions;
- a general *development function* is used to specify the relationship between module age and its realized properties (e.g. size, colour, shape, etc.).

3.1 DEFINITION OF TPD0L-SYSTEMS

We assume an *alphabet*, V, composed of a finite set of distinct *symbols* $s_1, s_2, \ldots, s_n$. tpD0L-systems operate on *timed, parametric modules,* which consist of timed symbols with associated parameters. For each module, the symbol also carries with it an *age* — a continuous, monotonically increasing, real variable, representing the amount of time the module has been active in the derivation string. Strings of modules form timed, parametric words, which can be interpreted to represent modeled structures (described in section 4). As with parametric L-systems, it is important to differentiate between formal modules used in production specification, and actual modules that contain real-valued parameters and a real-valued age.

Let V be an alphabet as defined above, $\Re$ the set real numbers and $\Re_+$ the set of positive real numbers, including 0. The triple

[1] The 'D' indicates deterministic, '0' represents the context level — in this case 0, which means context free.

$(s,\lambda,\tau) \in V \times \Re^* \times \Re_+$ is referred to as a *timed parametric module* (hereafter shortened to *module*). It consists of the symbol, $s \in V$, its associated parameter vector, $\lambda = a_1, a_2, \ldots, a_n \in \Re$ and the age of s, $\tau \in \Re_+$. A sequence of modules, $x = (s_1,\lambda_1,\tau_1)\cdots(s_n,\lambda_n,\tau_n) \in (V \times \Re^* \times \Re_+)^*$ is called a *timed, parametric word*. A module with symbol $S \in V$, parameters $a_1, a_2, \ldots, a_n \in \Re$ and age τ is denoted by $(S(a_1, a_2, \ldots a_n), \tau)$. It is important to differentiate the real-valued *actual* parameters of modules, from the *formal* parameters (notated by underlined symbols in this section) specified in productions. In practice, formal parameters are given unique[2] identifier names when specifying productions.

We assume the following definitions:

- Σ is the set of formal parameters, $C(\Sigma)$ is a logical expression using parameters from Σ, $E(\Sigma)$ is an arithmetic expression with parameters from the same set.
- C and E consist of formal parameters and numeric constants, combined using the standard operators +, –, /, *, ^ (exponentiation) $\sqrt[n]{\;}$ (nth root, defaulting to n=2 if n is not specified); relational operators $<, >, \leq, =, \neq$; logical operators ! (not), & (and), | (or); a number of trigonometric, stochastic and other functions; and parentheses '(', ')'. Rules for constructing expressions, operator precedence and associativity are the same as for the C programming language (Kernighan and Ritchie 1988).
- $\mathcal{C}(\Sigma)$ and $\mathcal{E}(\Sigma)$ are the sets of correctly constructed logical and arithmetic expressions with parameters from Σ. Logical expressions evaluate to Boolean values of TRUE or FALSE (equivalent to 1 or 0). Logical expressions evaluate to a real number in an arithmetic context.

A *timed, parametric 0L-system (tp0L-system)* is an ordered quadruplet $G = \langle V, \Sigma, \omega, P \rangle$ where:

- V is the non-empty set of symbols called the alphabet of the L-system;
- Σ is the *set of formal parameters*;
- $\omega \in (V \times \Re^* \times \Re_+)^+$ is a nonempty timed, parametric word over V, called the *axiom*, and

[2] Within the scope of the associated production.

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

A production $\left(\underline{a}, C, \underline{\chi}\right)$ is denoted $\underline{a} : C \rightarrow \underline{\chi}$, where the formal module $\underline{a} \in V \times \Sigma^* \times \Re_+$ is the *predecessor*, the logical expression $C \in \mathcal{C}(\Sigma)$ is the *condition*, and the formal timed parametric word $\underline{\chi} \in \left(V \times \mathcal{E}(\Sigma)^* \times \mathcal{E}(\Sigma)\right)^*$ is called the *successor*. Let $\left(s, \underline{\lambda}, \beta\right)$ be a predecessor module in a production $p_i \in P$ and $\left(s_1, \underline{\lambda}_1, \underline{\alpha}_1\right) \cdots \left(s_n, \underline{\lambda}_n, \underline{\alpha}_n\right)$ the successor word of the same production. The parameter $\beta \in \Re_+$ of the predecessor module represents the *terminal age* of s. The expressions, $\underline{\alpha}_i \in \mathcal{E}(\Sigma), i = 1..n$ sets the initial or *birth age*. Birth age expressions are evaluated when the module is created in the derivation string. This nomenclature is illustrated in the figure below. If the condition is empty, the production can be written $\underline{s} \rightarrow \underline{\chi}$. Formal and actual parameter counts must be the same for any given symbol.

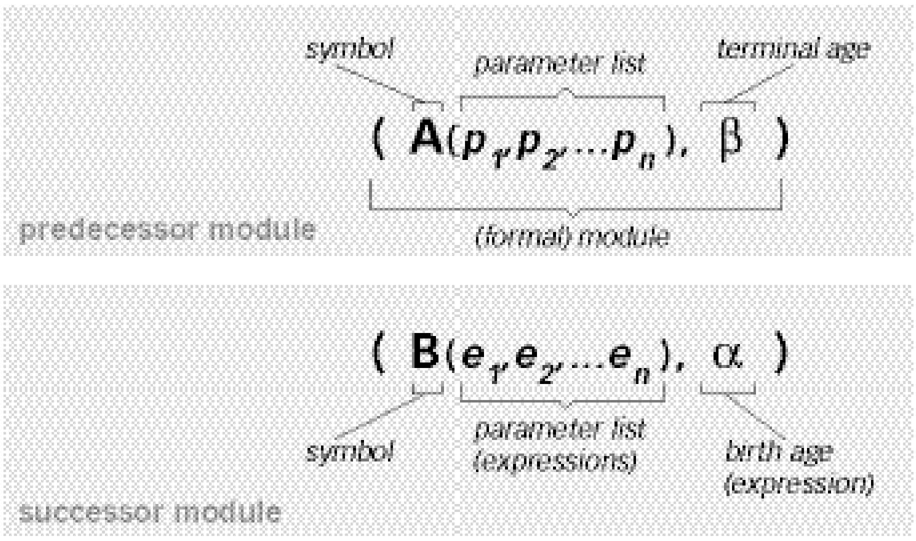


Figure 1. Nomenclature for predecessor and successor modules in a tpD0L-system

Here are some example productions:

$$\left(A\left(j,k\right),3.0\right) : j < k \rightarrow \left(B\left(j * k\right),0.0\right)\left(C\left(j+1,k-1\right),0.5\right) \tag{1}$$

$$\left(A\left(t\right),3.0\right) \rightarrow \left(A\left(t+1\right),3.0/t\right) \tag{2}$$

It is assumed:

- For each symbol $s \in V$ there exists at most one value of $\beta \in \Re_+$ for any production $p_i \in P$ where $(s, \underline{\lambda}, \beta)$ is the predecessor in p_i. If s does not appear in any production predecessor then the terminal age of s, $\beta_s = \infty$ is used (effectively the module never dies).
- If $(s, \underline{\lambda}, \beta)$ is a production predecessor in p_i and $(s, \underline{\lambda}_i, \underline{\alpha}_i)$ any module that appears in a successor word of *P* for s, then $\beta > \alpha_i$ when $\underline{\alpha_i}$ is evaluated and its value bound to α_i (i.e. the lifetime of the module, $\beta - \alpha_i > 0$).

Development proceeds according to some global time, *t*, common to the entire word under consideration. Local times are maintained by each module's age variable, τ (providing the relative age of the module). Re-writing begins with the axiom, ω. Modules age according to *t*, and when a module reaches it's terminal age, it is replaced by a successor word, χ, from the production whose predecessor module *matches* the module that has reached its terminal age. The formal parameters of the successor modules bound to the corresponding actual values when placed into the derivation string.

The necessary conditions for matching are: if the module and production predecessor symbols *and* parameter counts match; the condition statement, *C,* evaluates to TRUE when the module's actual parameters are bound to the formal parameters as specified in the predecessor module.

3.2 INTERPRETATION OF MODULES

A *derivation word* is the list of modules generated by *G* at some time *t.* In order for such an L-system to generate (for example) geometric form, this word must be interpreted to generate actual geometry. This is traditionally carried out using a *turtle interpretation*, based on the idea of turtle geometry (Abelson and DiSessa 1982) from the LOGO programming language, whereby an imaginary turtle maintains a local coordinate frame which is modified by commands such as 'move forward', 'turn left', 'turn right' and so on. Commands to draw ('pen up' and 'pen down') permit 2D drawing. The system described in this paper offers considerably more sophisticated drawing commands (detailed in section 4.1), but the basic principle remains the same. With a turtle interpretation, the entire derivation word is interpreted sequentially from left to right, generating the geometry. By interpreting the derivation word at different times in its development, animated sequences can be generated.

Certain symbols in the alphabet, V, are designated as *turtle commands*. The interpretation of these commands involves all elements of the module: the symbol determines the actual command, the module's age and parameters control various properties of the particular command. A module's parameters and age are connected to the turtle interpretation via the *development function*, specified in the next section.

3.3 DEVELOPMENT FUNCTIONS

Let $m = \left(s,\lambda,\tau\right) \in V \times \Re \times \Re_+$ be an actual module composed of the symbol, s, its actual parameters, λ and its current relative age, τ. A timed symbol $s \in V$ may optionally have associated with it a *development function,* $g_s : \left(V \times \Re^* \times \Re_+\right) \to \Re$. This function may involve any of the parameters, the current age, τ, and the terminal age β of s (determined by the predecessor of the production acting on s). Thus g_s is a real valued function that can be composed of any arithmetic expression $E\left(\underline{\lambda_s},\underline{\tau_s},\underline{\beta_s}\right)$. In addition to the formal parameters supplied, expressions can include the operators, numeric constants, and functions defined in section 3.1. The development function returns a real value, which is then used as a scaling factor for the actual parameter vector λ. That is:

$$\lambda' = g_s \cdot \left[\lambda\right] \tag{3}$$

The development function is evaluated whenever a module requires turtle interpretation, with parameter vector λ' sent to the turtle, rather than λ as is the case with parametric L-systems. No constraints are placed on the range or continuity of g_s, however if continuity is required when a production is applied (such as the accurate modeling of cellular development) g_s must be monotonic and continuous. These constraints extend to the development functions for those symbols that are related as being part of the successor definition of s.

The development function is specified in the form:

$$g_s\left(\texttt{parameter\_list}\right) = \texttt{expression} \tag{4}$$

where `parameter_list` is drawn from the parameters, age and terminal age of s $\left\{\underline{\lambda_s},\underline{\tau_s},\underline{\beta_s}\right\}$ and `expression` is any valid arithmetic expression.

4. Turtle Commands

The system provides a wide variety of turtle commands. For the sake of brevity, only a subset will be detailed here. For more complete details the reader is referred to (McCormack 2003).

The turtle maintains its own local coordinate reference frame, consisting of three orthogonal unit vectors representing the current Heading, Left and Up directions (denoted **H**, **L**, and **U** respectively). A world coordinate position, **t**, is also maintained (see Figure 2 below).

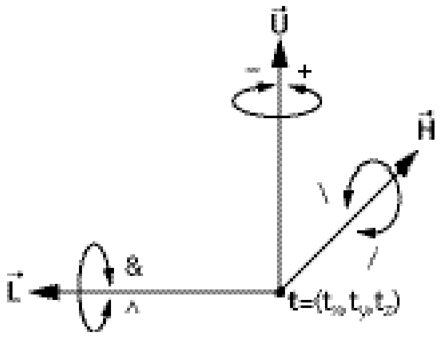


Figure 2. Turtle coordinate system and commands to change orientation

Basic turtle commands allow movement forward ('f'), turning left ('+') and right ('–'), pitch up ('^') and down ('&') and twist clockwise ('/') and anti-clockwise ('\'). Parameters to these commands describe the amount of that particular command, i.e. f(3.2) moves forward (in the direction of **H** by 3.2 units; +(45) turns the turtle left by 45 degrees.

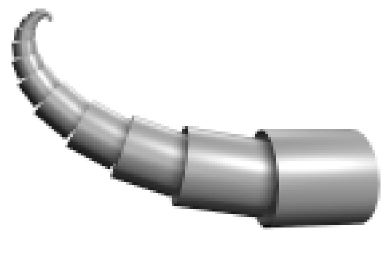
The turtle is a state-based system and the square brackets ('[' and ']') push and pop the turtle state (reference frame and associated drawing parameters) on and off a stack. This mechanism permits the creation of branching structures so important in plant and tree modeling applications (Prusinkiewicz and Hanan 1989).

Other commands instance geometry, for example the 'F' command draws a cylinder of length specified by its parameter (the radius is set using the '!' command). So for example the command sequence: !(3)F(10) draws a cylinder of radius 3 units and length 10, with the principle axis in the direction of **H**. In addition, the current turtle position is updated to the end of the cylinder after drawing.

4.1 GENERALIZED CYLINDERS

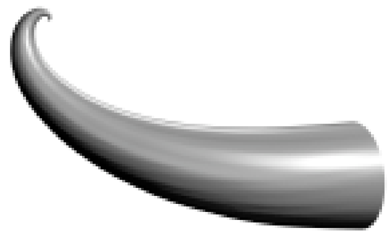
Wainwright suggests that the cylinder has found general application as a structural element in plants and animals (Wainwright 1988). He sees the cylinder as a logical consequence of the *functional morphology* of organisms, seeing the dynamic physiological processes (function) as dependent on form and structure over time. Wainwright distinguishes the cylinder as a natural consequence of evolutionary design based on the physical and mechanical properties of cylindrical structures.

This simple cylindrical method provided by the ‘F’ command is not sufficient for more complex geometric modeling. Consider the case of modeling a compound segment, such as a tentacle or horn. A relatively simple L-system can be used to describe such a shape, as illustrated in Figure 3. The problem in using the cylinder symbol, F, is that compound segments exhibit discontinuities between segments.



```
#define N   10
#define L   10
#define R   5
#define A   10
#define $k_l$  0.8
#define $k_r$  0.8
$\omega : horn(N,L,R,A)$
$p_1 : horn(n,l,r,a) \rightarrow seg(n,l,r,a)$
$p_2 : seg(n,l,r,a)$ : n > 0 : $\rightarrow$
    $!(r)\,F(l)\,\hat{}\,(a)$
    $seg(n-1, l*k_l, r*k_r, a*1.1)$
```

A



```
#define N   10
#define L   10
#define R   5
#define A   10
#define $k_l$  0.8
#define $k_r$  0.8
$\omega : horn(N,L,R,A)$
$p_1 : horn(n,l,r,a) \rightarrow c(1)\,cseg(n,l,r,a)$
$p_2 : cseg(n,l,r,a)$ : n > 0 : $\rightarrow$
    $!(r)\,C(l)\,\hat{}\,(a)$
    $cseg(n-1, l*k_l, r*k_r, a*1.1)$
```

B

Figure 3. A simple horn defined (**A**) using cylinders, which leaves noticeable gaps where the radius and angle of the cylinder changes. In **B**, this problem is fixed with the use of generalized cylinders. The parametric L-system generating each model is shown below the image (timed information is not shown)

This problem of describing more complex cylindrical shapes can be solved using *generalized cylinders*, originally developed by Agin for applications in computer vision (Agin 1972). Generalized cylinders have been used extensively by Bloomenthal to model tree limbs (Bloomenthal 1985) and a variety of natural objects (Bloomenthal 1995).

Similar systems to the one described here also make use of generalized cylinders. The *xfrog* system of Lintermann and Deussen makes basic use of cylindrical structures in stem and branch modelling (Lintermann and Deussen 1998; Lintermann and Deussen 1999). Prusinkiewicz et. al. describe an interactive plant modeling system that makes use of generalized cylinders and is based on L-systems (Prusinkiewicz et al. 2001). Both these systems rely on external curve editing software, whereas the system described in this paper uses an extended set of turtle commands to automate generalized cylinder construction.

The basic principle for creating a generalized cylinder is to define a series of *cross-sections*, possibly of varying shape and size, distributed over some continuous curve, known as the *carrier curve*. The cross-sections are connected to form a continuous surface.

Turtle commands include the selection and construction of cross-sections and the cubic interpolation of curves between cross-sections. Specific details on generalized cylinder generation can be found in (McCormack 2003).

5. Examples

Here some examples of the use of tpD0L-systems and their associated development functions are shown, highlighting their application in modeling animated mechanical and organic structures.

5.1 A SIMPLE PISTON SYSTEM

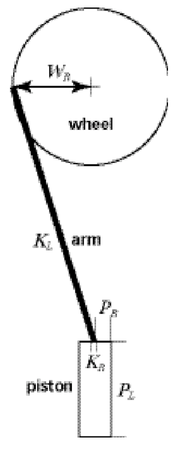


Figure 4: Piston schematic

This example simulates a simple piston and flywheel mechanism. The piston is connected to the flywheel via an arm of fixed length. The movement of the piston is constrained to the vertical, which drives the wheel in a circular motion. The schematic diagram (Figure 4, left) shows the principle features.

Figure 5 shows the L-system used to model the above system. The period of the system is determined by the constant ρ. In-built turtle geometry commands are sufficient to construct all the geometry. Production p_1 builds the system, productions $p_2 - p_4$ cycle the components in loops of period ρ. When a symbol reaches the end of its life, it begins again with age 0 and the cycle continues. The ‘equiv’ directives equate symbols with *different* names the *same* turtle function, for example the *bend* symbol is interpreted in the same way as the '+' command (counter-clockwise rotation about the **U** vector).

$$
\begin{array}{lll|lll|lll}
\#define & W_R & 50 & \#define & P_L & 60 & equiv & f & mov \\
\#define & K_L & 200 & \#define & \rho & 4 & equiv & + & bend \\
\#define & K_R & 2 & \#define & \mathrm{E} & 0.04 & equiv & - & turn \\
\#define & P_R & 10 & & & & & &
\end{array}
$$

$$\omega: \ (piston, 0)$$

$$p_1: \ (piston, \mathrm{E}) \to (mov(W_R), 0)\,!(P_R)\; cC(P_L, TRUE)(bend(1), 0)\,!(K_R)\; cC(K_L, TRUE)$$
$$(turn(1), 0)\; f(W_R) + (\pi/2)[\; disc(0, W_R)\,]$$

$$p_2: \ (mov(x), \rho) \to (mov(x), 0)$$

$$p_3: \ (bend(x), \rho) \to (bend(x+1), 0)$$

$$p_4: \ (turn(x), \rho) \to (turn(x), 0)$$

$$g_{mov}(\tau_{mov}, \beta_{mov}) = 1 - \cos\left(\frac{2\pi\tau_{mov}}{\beta_{mov}}\right)$$

$$g_{bend}(\tau_{bend}, \beta_{bend}) = \sin^{-1}\left(\frac{W_R \sin\left(\frac{2\pi\tau_{bend}}{\beta_{bend}}\right)}{K_L}\right)$$

$$g_{turn}(\tau_{turn}, \beta_{turn}) = g_{bend}(\tau_{turn}, \beta_{turn}) + \frac{2\pi\tau_{turn}}{\beta_{turn}}$$

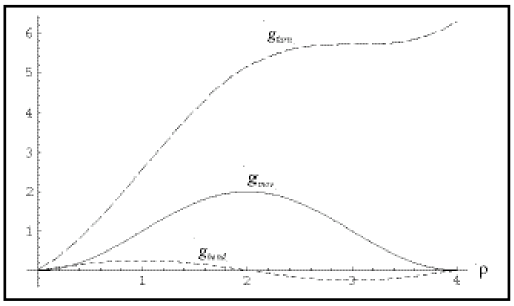


Figure 5. tp0L-system for the piston system and graph showing development functions over a single cycle

The development functions associated with each timed module provide the information necessary drive the animation. Thus, the productions provide structure and control, while the development functions control the animation properties of the model. The derivation word for the piston system is shown for various global time values in Table 1. The parameter values shown reflect the application of the development function.

TABLE 1: Derivation word at specific times for the L-system of Figure 5.

t	**Derivation String**
(global time)	(after application of development function)
0.0	$(piston, 0)$

0.04	$\left(mov(0),0\right)!(10)\,c\,C(60,\mathit{TRUE})\left(bend(0),0\right)!(2)\,c\,C(200,\mathit{TRUE})\left(turn(0),0\right)f(50)+(1.57)[\,disc(0,50)\,]$
1.04	$\left(mov(50),1\right)!(10)\,c\,C(60,\mathit{TRUE})\left(bend(0.253),1\right)!(2)\,c\,C(200,\mathit{TRUE})\left(turn(1.82),1\right)f(50)+(1.57)[\,disc(0,50)\,]$
4.04	$\left(mov(0),0\right)!(10)\,c\,C(60,\mathit{TRUE})\left(bend(0),0\right)!(2)\,c\,C(200,\mathit{TRUE})\left(turn(6.28),0\right)f(50)+(1.57)[\,disc(0,50)\,]$

5.2 MORPHOGENETIC DEVELOPMENT

L-systems are often associated with visual models of plants and plant ecosystems. This example shows the animated development of an imaginary species of plant from the interactive animation *Turbulence* developed by the author (McCormack 1994).

The full L-system describing the development is quite complex (approximately 35 productions), so in the interests of space and clarity an overview will be presented here. There are two main stages in the development of this sequence — *(i)* development of the stem, which after a certain time bifurcates into two segments; and *(ii)* the development of the flower head. The animated development of an individual model is shown below.

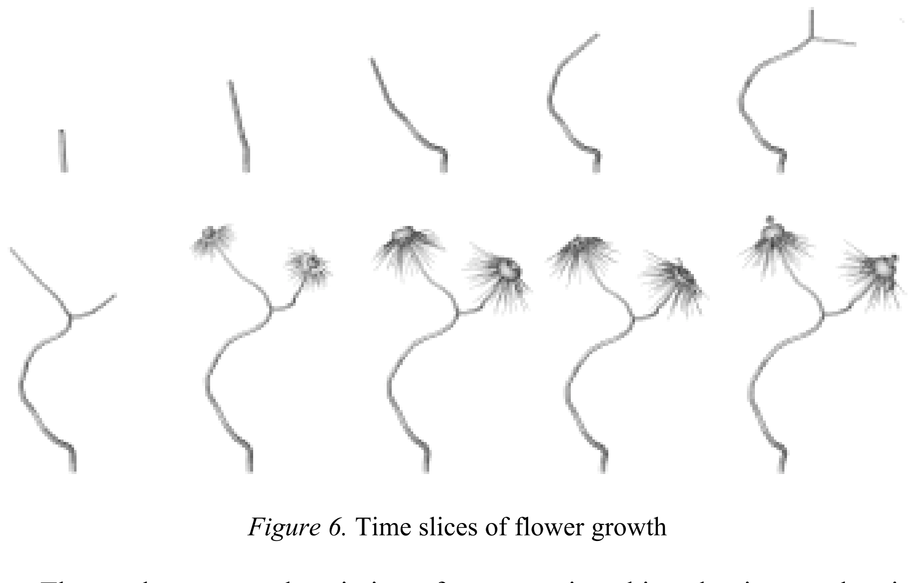


Figure 6. Time slices of flower growth

The randomness and variation of structure is achieved using stochastic functions that effect growth. The flower head consists of a number of animated components. The main head uses a set of three pre-defined surfaces, which are interpolated by development functions. Generalized cylinders are used to model the thorn and spike components. Note how these features animate in both shape and size.

A Bessel function of the first kind is used to model the scaling and animation of the flower head (Figure 7). Bessel functions are often used to solve motion equations for physical systems, and here the use of the function gives the animation a damped-spring-like quality (which the equation represents), as the head 'puffs' up rapidly in size and then pulsates in an oscillating rhythm, slowly damping down as the element ages. Visually similar behavior is observed in time-lapse sequences of real plants, as they respond to the rhythms of the day/night cycle.

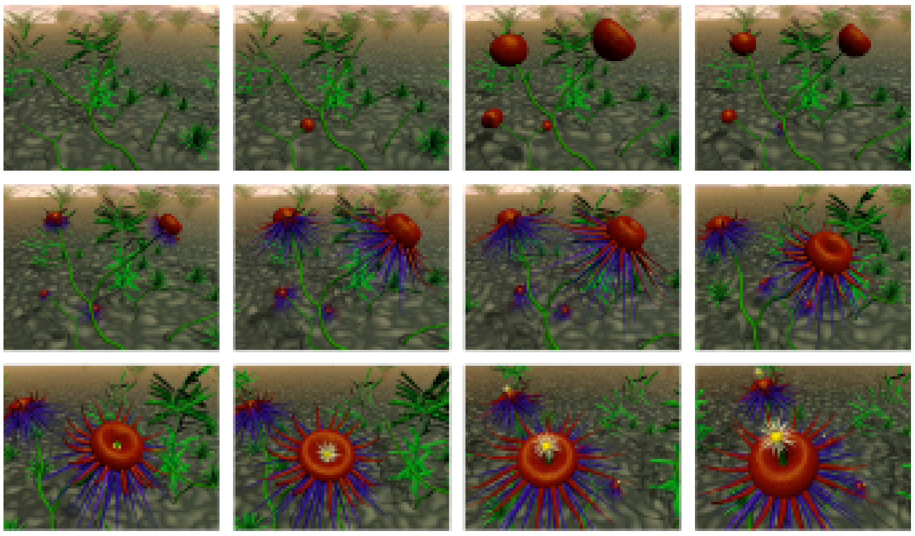


Figure 7. Time sequence showing development of the flower head

5.3 LEGGED GAITS OF ANIMALS

Timed L-systems may also be used as controllers in the simulation of animal gait cycles. Here the advantage of L-systems is that they can provide both motor control and structural definition within the same grammar.

For the example detailed in this section, we will consider the representation of an 'animal' with multiple rigid body segments, each connected with a 2-DOF[3] articulation as shown in Figure 8 below.

[3] Degree of Freedom

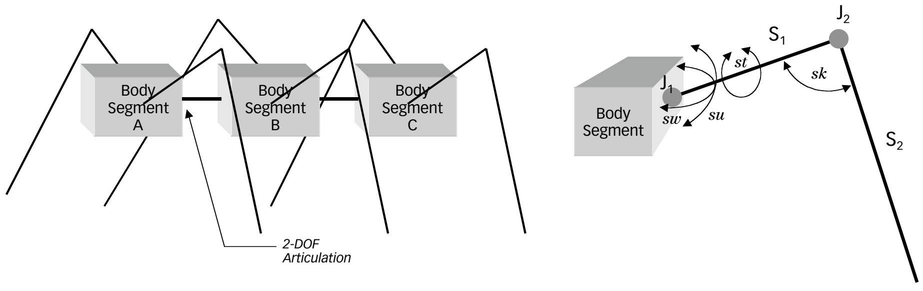


Figure 8: Legged animal composed of multiple articulated body segments (left) and detail of an individual segment's joint configuration (right)

Each body segment has two multi-jointed legs, at opposite sides of the body segment. The leg detail is also shown in the figure. A *leg* is composed of two rigid limb segments, S_1 and S_2. S_1 is attached to the body segment by a 3-DOF joint, J_1. The joint J_2 between S_1 and S_2 has 1-DOF.

The key advantage of an L-system specification is in the flexibility of body and limb design and specification. Similar techniques have been used as a general system to evolve novel designs of articulated figures (Sims 1994a; Sims 1994b). Arbitrary joint, limb, and body segment configurations can be achieved by modifying the productions that generate these elements. The generalized cylinder techniques (discussed in section 4) are used to model the complex limb and body geometries of the creature.

The motor control structure is also specified by a tp0L-system, forming a simple finite state machine that drives the movement of the body segments and legs. Constraints on the movement of individual limbs are set within the development functions for each joint.

$$\begin{array}{lll|lll|lll}
\#define & \beta_{ls} & \varphi & equiv & + & sw & \#define & \beta_{st} & \frac{\varphi}{5} \\
\#define & \beta_{sw} & \varphi & equiv & \wedge & su & \#define & \beta_{sk} & \varphi \\
\#define & \beta_{su} & \varphi & equiv & / & st & \#define & R_{S12} & \frac{3}{2} \\
 & & & equiv & \& & sk & & &
\end{array}$$

$$p_1 : \left(leg(\phi,l),\beta_{ls}\right) \rightarrow \left(sw(1,\phi),0\right)\left(su(1,\phi),\frac{3\beta_{su}}{4}\right)\left(st(1,\phi),0\right) ls(l)\left(sk(1),0\right) ls\left(R_{S12}l\right)$$

$$p_2 : \left(sw(n,\phi),\beta_{sw}\right) \rightarrow \left(sw(n,\phi),0\right)$$

$$p_3 : \left(su(n,\phi),\beta_{su}\right) \rightarrow \left(su(n,\phi),0\right)$$

$$p_4 : \left(st(n,\phi),\beta_{st}\right) \rightarrow \left(st2(n,\phi),\beta_{st}\right)$$

$$p_5 : \left(sk(n,\phi),\beta_{sk}\right) \rightarrow \left(sk(n,\phi),0\right)$$

$$p_6 : \left(st2(n,\phi),3\beta_{st}\right) \rightarrow \left(st(-n,\phi),0\right)$$

Figure 9. tp0L-system fragment for the gait control mechanism to specify a single leg configuration. The cycle time of a single step is specified by the variable φ

To illustrate how this scheme works, we will focus on the construction and animation of a single leg. The L-system shown in Figure 9 captures the essential components. The modules *sw*, *su* and *st* represent the motor control of S_1 (3-DOF articulation), while *sk* represents the joint angle between S_1 and S_2. The module *ls* (leg segment) calls a complex set of productions to create the geometry of the leg using generalized cylinders (these productions are not show for the sake of clarity). This module's parameter specifies the overall length of the leg segment and the turtle is placed at the position of the next joint upon completing of the geometric construction of the leg segment. The *ls* module is not timed due to the fixed structure of the leg segment itself. This constraint means that the geometric data can be cached to avoid regeneration across multiple locations and time steps. The constant R_{S12} is the ratio of size between the upper (S_1) and lower (S_2) leg segments.

An individual *walk cycle* represents the movement of the leg system over a single gait. The total time for this cycle is represented by the variable φ, with individual controllers using this time or a rational ratio of it to ensure cyclic animation. For example, the twisting motion of the leg (module *st*) is separated into three distinct components which sum to the gait cycle time.

The parameter ϕ represents the phase of the animation cycle. As each body segment is added the phase of the walk cycle is shifted to ensure correct motion relative to the position of the segment. The parameters *n* and *l* control the magnitude of the gait and the leg respectively.

# *define*	R_{sw}	0	# *define*	A_{su}	0.139π	# *define*	R_{sk}	0.389π
# *define*	A_{sw}	0.378π	# *define*	R_{st}	0	# *define*	A_{sk}	0.194π
# *define*	R_{su}	0.056π	# *define*	A_{st}	0.056π			

$$g_{sw}\left(\tau_{sw},\beta_{sw},\phi\right) = R_{sw} + A_{sw}\sin\left(\frac{2\pi\tau_{sw}}{\beta_{sw}} + \phi\right)$$

$$g_{su}\left(\tau_{su},\beta_{su},\phi\right) = \mathrm{if}\left(\tau_{su} < \frac{\beta_{su}}{2}, 0, R_{su} - A_{su}\left(\cos\left(2\pi\left(\frac{\tau_{su}}{\beta_{su}} - 0.5\right) + \phi\right) + 1\right)\right)$$

$$g_{st}\left(\tau_{st},\beta_{st},\phi,n\right) = R_{st} + \mathrm{sign}\left(n\right)A_{st}\sin^{2}\left(\frac{\pi\tau_{sw}}{\beta_{sw}} + \phi\right)$$

$$g_{sk}\left(\tau_{sk},\beta_{sk},\phi\right) = \mathrm{if}\left(\tau_{sk} < \frac{\beta_{sk}}{2}, 0, R_{sk} - A_{sk}\left(\cos\left(4\pi\left(\frac{\tau_{sk}}{\beta_{sk}} - 0.5\right) + \phi\right) + 1\right)\right)$$

Figure 10. Key development functions for the gait of a single leg

The productions of Figure 9 specify a simple state machine that encodes the geometric and temporal structure of the leg gait. The remaining information required is the associated development function for each module, detailed in Figure 10.

These functions approximate the inverse kinematical solution of the system for a walking gait. The use of periodic functions ensures that phase and perfect cycling are easily accommodated, as required for a system that must coordinate a large number of legs in a coherent fashion. In the actual generated sequences, noise perturbations based on the age of the creature are used to introduce variation and a ‘natural’ feel into the walk cycle.

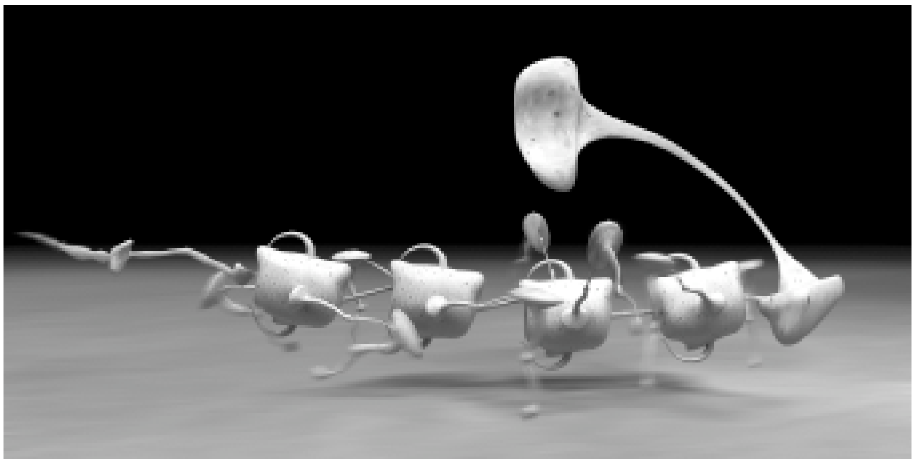


Figure 11. Still frame showing the legged creature running. The geometry and animation is generated using the timed L-system techniques described in this paper

6. The Design Environment

This section briefly describes how, in a practical sense, the formal systems described in this paper can be used in a design environment. Design using generative methods involves the creation and modification of rules or systems that interact to generate the finished design. Hence, the designer does not directly manipulate the produced artifact, rather the rules and systems involved in the artifact's production. The design process becomes one of *meta-design* where a finished design is the result of the *emergent* properties of the interacting system (McCormack and Dorin 2001). The 'art' of designing in this mode is in mastering the relation between process specification, environment, and generated artifact. Since this is an art, there is no formalized or instruction-based method that can be used to guide this relationship. The role of the human designer remains, as with conventional design, central to the design process.

In the case of using the generative methods described in this paper, fundamentally the meta-design process must produce a tpD0L-system axiom and set of productions. To this end, the author has used three different meta-design methods: *(i)* specification of L-systems by hand, similar to the way a programmer writes a computer program; *(ii)* using a visual programming metaphor, where modules and productions are represented visually and may be manipulated topologically to create new productions; *(iii)* using artificial evolutionary techniques based on aesthetic selection (McCormack 1993).

Of these three methods, the first (direct manipulation) gives the most flexibility and control, but is the most difficult for people without a strong programming background to understand. The second gives less control, but is much more intuitive from a design perspective, and is used in commercial implementations (Lintermann and Deussen 1999). The third method, interactive evolution, is excellent for generating novel designs without needing to understand the productions involved. However, exact control is extremely difficult. The method favored by the author is a combination of explicit editing of L-system productions combined with interactive evolution. This is the methodology used to produce the examples described in Section 5.2 and 5.3. Addressing the deficiencies of all these techniques remains an open research problem, if generative design is to achieve widespread adoption by the design community.

7. Conclusions

The integration of timed and parametric components to L-systems permits a new degree of flexibility and possibility for generative modeling, some of which has been illustrated in this paper. In the area of visual simulation, the use of compound structures such as generalized cylinders can be integrated into a turtle interpretation of modules generated by L-systems. The system described here enables the procedural generation of complex, time-dependent geometric structures, in ways that would be difficult or impossible to design using other methods.

Important extensions to this model are the use of hierarchical specification of grammars and the incorporation of context into developmental systems (McCormack 2003). These extensions provide even further flexibility, particularly when modeling organic structures and their morphogenesis.

Acknowledgements

This work was produced with the assistance of a fellowship grant from the Australia Council for the Arts New Media Arts fund, and a creative development grant from the Australian Film Commission. The author acknowledges the generous support of these cultural organizations in funding this research. The paper reviewers also provided useful comments for the final version of this paper.

References

Abelson, H and DiSessa, AA: 1982, *Turtle geometry: the computer as a medium for exploring mathematics,* MIT Press, Cambridge, Mass.

Agin, GJ: 1972, *Representation and Description of Curved Objects,* Technical Memo, No. AIM-173, October 1972. Stanford Artificial Intelligence Report, Stanford, California.

Bloomenthal, J: 1985, Modeling the Mighty Maple. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985, Barsky, BA, ed). In *Computer Graphics* **19**(3) ACM SIGGRAPH, New York**,** pp. 305-311.

Bloomenthal, J: 1995, *Skeletal design of natural forms,* Ph.D. thesis, Department of Computer Science, University of Calgary, Calgary, Alberta.

Coates, P, Broughton, T and Jackson, H: 1999, Exploring Three-dimensional Design Worlds using Lindenmayer systems and Genetic Programming, *in* Bentley, PJ (ed), *Evolutionary Design by Computers*, Morgan Kaufmann, London, UK, pp. Chapter 14.

Durikovic, R, Kaneda, K and Yamashita, H: 1998, Animation of Biological Organ Growth Based on L-systems, *Computer Graphics Forum (EUROGRAPHICS '98)* **17**(3)**:** 1-14.

Hanan, J: 1992, *Parametric L-Systems and their Application to the Modelling and Visualization of Plants,* Ph.D. thesis, Computer Science, University of Regina, Saskatchewan.

Hornby, GS and Pollack, JB: 2001a, The Advantages of Generative Grammatical Encodings for Physical Design, *in*, *Proceedings of the 2001 Congress on Evolutionary Computation*, IEEE Press, pp. 600-607.

Hornby, GS and Pollack, JB: 2001b, Evolving L-systems to generate virtual creatures, *Computers & Graphics* **26**(6)**:** 1041-1048.

Kernighan, BW and Ritchie, DM: 1988, *The C Programming Language* (Second Edition), Prentice Hall, Englewood Cliffs, New Jersey.

Kitano, H: 1990, Designing neural networks using genetic algorithms with graph generation system, *Complex Systems* **4**(4)**:** 461-476.

Kuhn, TS: 1996, *The structure of scientific revolutions* (Third Edition), University of Chicago Press, Chicago, Ill.

Lindenmayer, A and Rozenberg, G: 1979, Parallel generation of maps: Developmental systems for cell layers, *in* Claus, V, Ehrig, H and Rozenberg, G (eds), *Graph grammars and their application to computer science; First International Workshop*, Vol. Lecture Notes in Computer Science 73, Springer-Verlag, Berlin, pp. 301-316.

Lintermann, B and Deussen, O: 1999, Interactive modeling of plants, *IEEE Computer Graphics & Applications* **19**(1)**:** 2-11.

Lintermann, B and Deussen, O: 1998, A modelling method and interface for creating plants, *Computer Graphics Forum* **17**(1)**:** 73-82.

McCormack, J: 2003, *The Application of L-systems and Developmental Models to Computer Art, Animation, and Music Synthesis,* Ph.D. thesis, School of Computer Science and Software Engineering, Monash University, Clayton.

McCormack, J: 1996, Grammar-Based Music Composition, *in* Stocker, R, et al. (eds), *Complex Systems 96: from Local Interactions to Global Phenomena*, ISO Press, Amsterdam, pp. 321-336.

McCormack, J: 1993, Interactive Evolution of L-System Grammars for Computer Graphics Modelling, *in* Green, D and Bossomaier, T (eds), *Complex Systems: from Biology to Computation*, ISO Press, Amsterdam, pp. 118-130.

McCormack, J: 1994, Turbulence: An Interactive Installation Exploring Artificial Life. Visual Proceedings: The Art and Interdisciplinary Programs of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics* Annual Conference Series, ACM SIGGRAPH, New York, pp. 182-183.

McCormack, J and Dorin, A: 2001, Art, Emergence and the Computational Sublime *in* Dorin, A (ed) *Second Iteration: a conference on generative systems in the electronic arts*, CEMA, Melbourne, Australia, pp. 67-81.

Nevill-Manning, CG and Witten, IH: 1997, Compression and explanation using hierarchical grammars, *The Computer Journal* **40**(2/3)**:** 103-116.

Parish, YIH and Müller, P: 2001, Procedural Modeling of Cities. Proceedings of SIGGRAPH 2001 (Los Angeles, California, August 12-17). In *Computer Graphics Proceedings* Annual Conference Series, ACM SIGGRAPH, pp. 301-308.

Prusinkiewicz, P and Hanan, J: 1989, *Lindenmayer Systems, Fractals and Plants,* Springer-Verlag, Berlin.

Prusinkiewicz, P and Hanan, J: 1990, Visualization of Botanical Structures and Processes using Parametric L-systems, *in* Thalmann, D (ed), *Scientific Visualization and Graphics Simulation*, John Wiley & Sons, Chichester, pp. 183-201.

Prusinkiewicz, P and Lindenmayer, A: 1990, *The algorithmic beauty of plants,* Springer-Verlag, New York.

Prusinkiewicz, P, et al.: 2001, The use of positional information in the modeling of plants. Proceedings of SIGGRAPH 2001 (Los Angeles, California, August 12-17). In *Computer Graphics Proceedings* Annual Conference Series, ACM SIGGRAPH, pp. 289-300.

Salomaa, A: 1973, *Formal Languages,* Academic Press, New York, NY.

Sims, K: 1994a, Evolving 3D Morphology and Behavior by Competition *in* Brooks, R and Maes, P (eds), *Proceedings of Artificial Life IV*, MIT Press, pp. 28-39.

Sims, K: 1994b, Evolving Virtual Creatures. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 15-22.

Wainwright, SA: 1988, *Axis and circumference: the cylindrical shape of plants and animals,* Harvard University Press, Cambridge, Mass.