On the Evolution of Parametric L-systems

Roger Curry

research conducted under the supervision of Przemyslaw Prusinkiewicz

Department of Computer Science
University of Calgary, Alberta
CANADA
email: curry@cpsc.ucalgary.ca

9 November 1999

Abstract

L-systems have been shown to be a useful tool for modeling plants; however, the generation of realistic plant models requires an extensive knowledge of L-systems. People who are not L-system experts require a simpler way of creating plant models. This paper will describe my attempts at designing a user interface which allows the user to guide the evolution of plant models (generated from a parametric L-system) and the genetic algorithm employed to facilitate this evolution. The final goal of this research would be a design environment in which plant models could be easily and quickly generated without explicit knowledge of L-systems. The plant models would still be L-systems but the user would not ever need to see them. By exploring the concept of evolution as a method for developing plant models we may also gain insight into the development of plants themselves.

1 Introduction

Over the past decade L-systems have proven themselves a useful tool for the modeling of plant structures. Many life-like models can be found in *The Algorithmic Beauty of Plants* [1]. Generation of models such as these requires not only knowledge of the plant being modeled but also a clear understanding of L-systems (and corresponding modeling software such as *Cpfg* [2]). Creating an L-system model is not always intuitive; it requires that one think recursively about the development of a plant.

Employing genetic programming techniques [3, 4] in order to evolve L-systems may allow generation of realistic plant models in a shorter period of time and perhaps more intuitively than was previously possible. A long-term goal of this research is a design environment in which plant models can be generated (via user-guided evolution) without explicit knowledge of L-systems. Using this approach the user works only with visual models and not with their textual representations. This will allow both experts and non-experts alike to quickly generate realistic plant models in an intuitive fashion.

As a first step towards this goal, this paper describes an interface which allows the user to guide the evolution of plant models (generated from a parametric L-system). It also highlights the genetic operations which were employed in evolving these models.

The rest of this section contains some background information regarding genetic algorithms and parametric L-systems; however, some familiarity with these topics is assumed. In Section 2 the details of the model are examined, and in Section 3 the observations and results are presented.

1.1 Parametric L-systems

The concept of parametric L-systems [1, Section 1.10] was formulated in order to extend the modeling capabilities of L-systems and to simplify the representation of continuous phenomena within a model.

In a parametric L-system each symbol can be associated with one or more real-valued parameters. In addition, the application of a production may be dependent upon an associated condition. Parametric L-system productions have the form:

$$predecessor : condition \rightarrow successor.$$

In the simplest case the predecessor is a single symbol and its associated parameters. If the predecessor's parameters satisfy the condition then the predecessor is replaced by the successor. Consider the following parametric L-system:

$$\begin{array}{lll} \omega: & A(0.0, 0.2) & \\ p_1: & A(s,f) & : s < f \rightarrow F[+F][-F]A(s+0.1, f) \\ p_2: & A(s,f) & : s \geq f \rightarrow F@O(0.5) \end{array} \tag{1}$$

The derivation sequence is as follows:

$$\begin{array}{l} \mu_0 : A(0.0, 0.2) \\ \mu_1 : F[+F][-F]A(0.1, 0.2) \\ \mu_2 : F[+F][-F]F[+F][-F]A(0.2, 0.2) \\ \mu_3 : F[+F][-F]F[+F][-F]F@O(0.5) \end{array}$$

Start with μ_0 as the axiom. Production p_1 applies (since $0.0 < 0.2$), thus replace $A(0.0, 0.2)$ with $F[+F][-F]A(0.1, 0.2)$. Again production p_1 applies (because $0.1 < 0.2$) and we get μ_2. At this point in the derivation, production p_2 applies (since $0.2 = 0.2$), yielding μ_3. Note that there is no production with F as the predecessor, so $F \rightarrow F$ is assumed.

The graphic interpretation of parametric L-systems is similar to that of L-systems without parameters, except that symbols used for drawing may be parameterized. For example, in a non-parametric L-system the symbol F tells the "turtle" to move forward in the direction of the current heading by 1 world unit. A parametric L-system may contain F(c) which indicates that the movement should be by c world units. Other drawing commands behave similarly. Figure 1 is the graphical interpretation of the string μ_3 generated by L-system (1).

See [1, Section 1.10] for a mathematical description of parametric L-systems and further details on their "turtle" interpretations.

1.2 Genetic Programming

The general concept of genetic programming will now be presented. More information regarding the topic can be found in [3, 4, 5, 6]. Genetic algorithms are an optimization technique based on Charles Darwin's [7] theory of survival of the fittest and natural selection. Genetic programming is an approach to problem solving that employs algorithms functioning analogously to evolution.

In an evolutionary model we start with an initial population of genetically diverse individuals. Each of these individuals has an associated genetic code from which it can be constructed. Also required is a function which evaluates the fitness of the individuals. Fitness is defined as the ability of an individual to survive in its environment and reproduce. The last major requirement is a method of simulating sexual reproduction

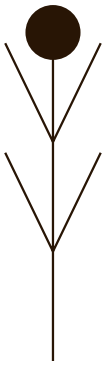


Figure 1: Graphic interpretation of L-system (1).

(*crossover*) and *mutation*. This can be accomplished by defining genetic operations which act upon the genetic codes of the individuals in our population. A *crossover* operation takes the genetic codes of two *parent* individuals and creates a new *child* exhibiting characteristics of both parents. A *mutation* operation takes a single *parent* and produces (via a pseudo-random operation) a new individual that is based on the original but modified in some aspect. Now we can begin to simulate evolution. A simple genetic algorithm is presented below.

1. (Randomly) generate the initial population.
2. Evaluate the fitness of each individual.
3. Take a portion of the individuals with superior fitness and apply the genetic operations to create some new individuals.
4. Remove those individuals who do not posses the minimum fitness required for survival.
5. Goto step 2.

This algorithm would be terminated after some fixed number of iterations. The individuals which comprise the population after this algorithm finishes should have a higher average fitness than those in the initial population. Over time the population will become more fit to survive in its environment.

2 The Model

In this section we examine the L-system itself, the genetic algorithm employed to evolve this L-system, and the user interface through which this evolution is guided.

2.1 The L-system

Honda described a simple parametric model which could be used to generate diverse tree-like structures. The following five assumptions [1, pages 51-52] form the basis of his model.

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

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

L-system (2) (from [8, page 551]), partially satisfies Honda's criteria. The first three assumptions are completely satisfied, and the daughter segments do form constant branching angles with respect to the mother branch. However, L-system (2) does not guarantee that the branch plane is fixed with respect to gravity, nor that it contains both the mother segment and its two daughter segments. For aesthetic reasons we attempt to model the narrowing girth of branches as one proceeds from the main axis though to higher order branches, however the girth of branches does not affect development of the plant model in any other way. The method used is loosely based on Leonardo da Vinci's postulate [9, page 156] which claims that "all the branches of a tree at every stage of its height when put together are equal in thickness to the trunk below them."

$$\begin{array}{lll} \omega: & A(100, w_0) & \\ p_1: & A(s,w): & s >= min \rightarrow\ !(w)F(s) \\ & & [+(\alpha_1)/(\varphi_1)A(s*r_1, w*q\hat{}e)] \\ & & [+(\alpha_2)/(\varphi_2)A(s*r_2, w*(1-q)\hat{}e)] \end{array} \tag{2}$$

Symbol A represents an apex. Angle values $\alpha_1, \alpha_2, \varphi_1$, and φ_2 determine the trajectory of new apices. s and w specify the length and width of the current internode. r_1 and r_2 determine the factor by which the length of successive internodes is decreased. w_0, q and, e are constants which control the width of branches. Branches will only form if their length is at least min.

Prusinkiewicz extended the class of trees produced by L-system (2) by adding to it the notion of fruit. The basic structure of L-system (3) is similar to (2), except that a second production has been added which creates pieces of fruit on the tree whenever the parameter `s` falls below `MINSIZE×0.1`. Some constant multipliers have also been added but these are used only to scale the range of input and do not affect the descriptive power of the L-system. For example the value min in L-system (2) corresponds to `MINSIZE×0.1` in L-system (3). Note: L-system (3) corresponds to actual *Cpfg* [2] code.

```
 Axiom :    A(0.3, W0)
A(s,w) :    s >= MINSIZE*0.1  --> !(w)F(s)!(w*b);
           [+(ANG1*120)/(DIV1*360)A(s*(0.75+0.3*RATE1),w*q^e)]
           [+(ANG2*120)/(DIV2*360)A(s*(0.75+0.3*RATE2),w*(1-q)^e)]
A(s,w) :    s < MINSIZE*0.1  --> ;(15)@O(0.1*MINSIZE)
```

(3)

Again, Symbol `A` represents an apex. `ANG1, ANG2, DIV1,` and `DIV2` correspond to angle values $\alpha_1, \alpha_2, \varphi_1$, and φ_2 of L-system (2). As before, `s` and `w` specify the length and width of the current internode. `RATE1` and `RATE2` correspond to r_1 and r_2. Constants `W0, q` and, `e` control the width of branches. At a certain point, when parameter `s` drops below `MINSIZE×0.1` a branch will produce fruit instead of branching again.

Derivations proceed as follows. We begin with an apex `A(0.3, W0)`. The first production is then repeatedly applied. This production replaces every apex `A(s,w)` with an internode of length `s` (and width `w`) and two

Structure	ANG1	ANG2	RATE1	RATE2	DIV1	DIV2	MINSIZE
0	0.010	-0.328	0.547	-0.823	0.603	0.423	-0.350
1	0.064	-0.175	0.354	-0.367	-0.002	0.142	-0.364
2	0.135	-0.243	-0.036	-0.410	0.453	0.011	0.425
3	0.008	-0.309	0.145	-0.416	0.454	0.011	0.227
4	-0.197	0.168	0.029	-0.066	0.045	-0.012	-0.177
5	0.388	0.210	-0.157	0.189	0.738	-0.524	0.317
6	-0.217	-0.247	0.228	0.365	-0.096	-0.375	0.063
7	-0.358	0.201	-0.083	0.361	0.249	-0.019	-0.280
8	-0.234	0.313	-0.029	0.418	-0.151	-0.340	-0.163

Table 1: Parameter values for structures in Figure 2.

new apices which will form child branches whose trajectories (with respect to their mother segment) will be determined by angles `ANG1, ANG2, DIV1, DIV2`. The length of the child branches will depend on the values of `RATE1` and `RATE2`. The second production producing spherical fruit is applied when the parameter `s` falls below `MINSIZE`$\times$`0.1`.

Although we manipulate only seven parameters (`ANG1, ANG2, DIV1, DIV2, RATE1, RATE2, MINSIZE`), L-system (3) is capable of producing a wide variety of branching structures as shown in Figure 2. The parameters used to generate these images are listed in Table 1.

2.2 The Genetic Algorithm

Representation & Terminology

As is common in much literature on genetic algorithms, some terminology is borrowed from genetics. An individual parameter is referred to as a *gene*. Parameters are represented as floating point numbers in the range $(-\infty, +\infty)$. The initial population is generated using uniformly distributed pseudo-random numbers in the range $(-0.5, 0.5)$. An ordered set of seven parameters constitutes a *genotype*. We use a 7-dimensional real vector to represent this set of seven parameters. The image created using these parameters and L-system (3) is referred to as the *phenotype*.

Algorithm

The genetic algorithm employed in these experiments can be summarized as follows:

- We start with an initial population of 9 randomly created individuals.
- There are 5 different genetic operations (see below).
- The fitness of individuals is determined by the user according to aesthetic criteria.

This genetic algorithm is atypical in the following two ways:

1. We are using a small population.
2. The user manually determines the fitness function by selecting which genotypes will be combined. The user chooses parents from a set of phenotypes based on their aesthetic qualities. We can say that the fitness function is dependent on the aesthetic appeal of the phenotypes. This approach of user-guided simulation of evolution was used by both Richard Dawkins and Karl Sims [6, 10], and later applied specifically to L-systems by Jon McCormack [11].

Genetic operations

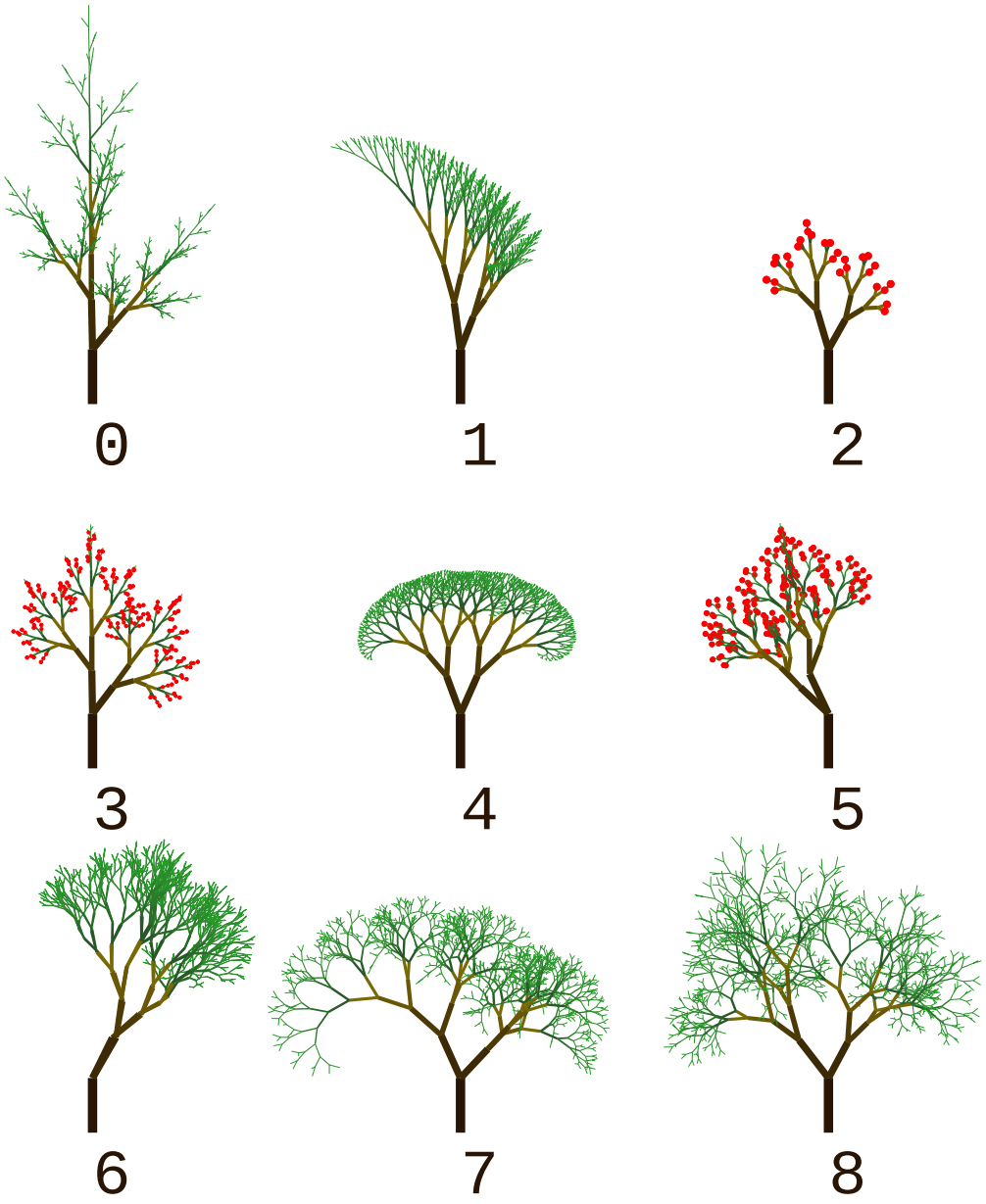


Figure 2: Sample structures produced using L-system (3).

We now describe the various genetic operations that can be applied to the genotypes in our model. We have considered both asexual and sexual operations, and both stochastic and deterministic operations. These operations are detailed below.

- *Single-parent mutation*

  Input: a single genotype (parent), a real mutation factor $x \in (0, 1)$
  Output: a single genotype
  Description: An individual genotype can be mutated by randomly varying the genes which it contains. The amount by which each gene will vary is proportional to the mutation factor.

  Process:

  1. create a vector of random numbers in the range (-0.5, 0.5)
  2. multiply this vector by the mutation factor x
  3. add the result to the parent genotype (vector)

  Example (mutation factor = 0.33):
  Random number (vector)

0.125	-0.031	-0.384	-0.235	-0.196	-0.020	-0.135

  Random number (vector) × mutation factor = 0.33

0.041	-0.010	-0.127	-0.077	-0.065	-0.007	-0.044

  +
  Parent genotype

0.049	-0.319	0.019	-0.494	0.389	0.004	0.183

  =
  Child genotype

0.008	-0.309	0.145	-0.416	0.454	0.011	0.227

  See Figure 3 for the corresponding phenotypes.

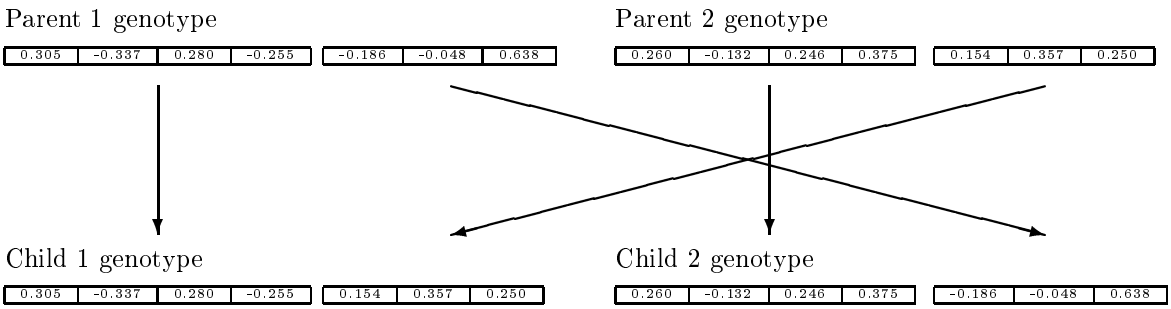
- *Single-point crossover*

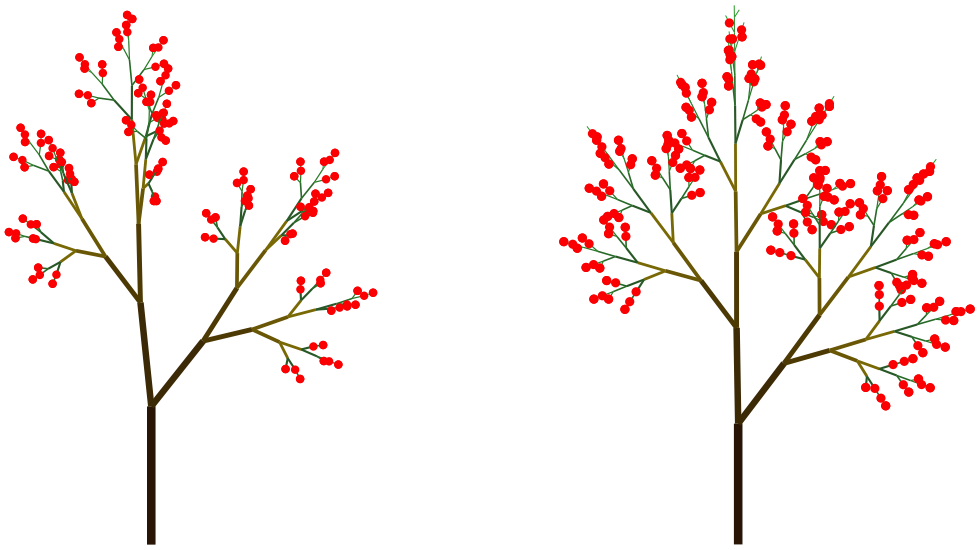
  Input: two distinct genotypes (parents), an integer crossover point $x \in [1, 6]$
  Output: a set of two genotypes (children)
  Description: Two genotypes can be sexually recombined to create two new individuals based on a given crossover point.

  Process:

  1. create the first new individual by copying the first x genes from parent 1 and the last $7 - x$ genes from parent 2.
  2. create the second new individual by copying the first x genes from parent 2 and the last $7 - x$ genes from parent 1.

  Example (crossover point = 4):





Parent phenotype Child phenotype

Figure 3: Sample *Single-parent mutation* operation.

See Figure 4 for the corresponding phenotypes.

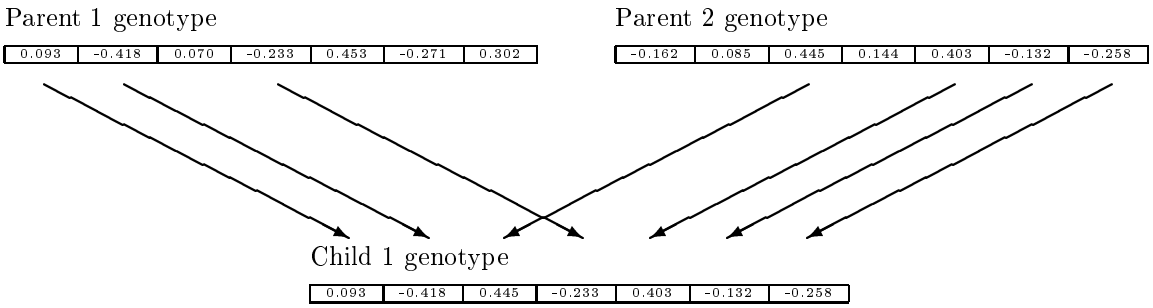
- *Multi-point crossover*

  Input: two distinct genotypes (parents), an integer contribution factor $x \in [1, 6]$
  Output: a single genotype (child)
  Description: Two genotypes can be sexually recombined to create a new individual which inherits a certain portion of each parent's genetic information. Which genes will be inherited from which parent is determined randomly; the number of genes inherited from each parent is determined by the contribution factor.

  Process:

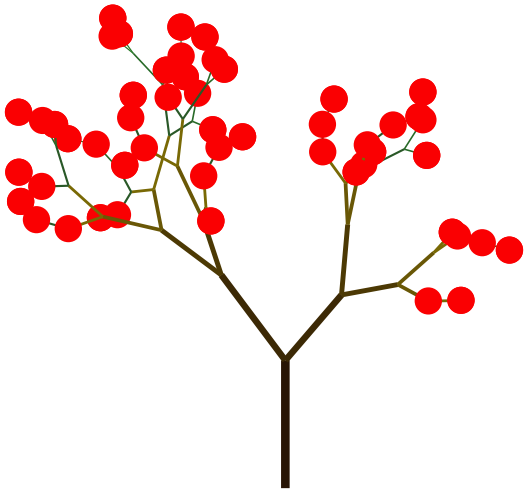
  1. randomly choose x distinct genes to copy from parent 1
  2. copy these genes into the child genotype
  3. copy the remaining $7 - x$ genes from parent 2

  Example (contribution factor = 3):

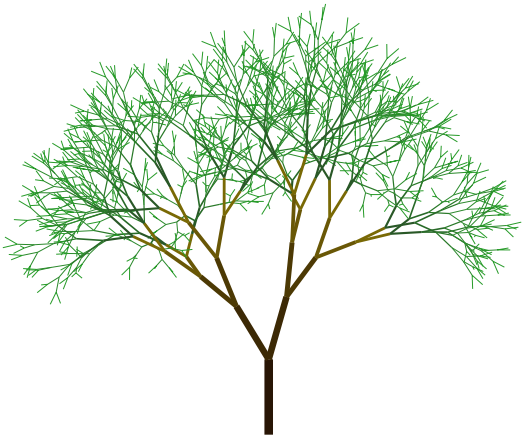


See Figure 5 for the corresponding phenotypes.

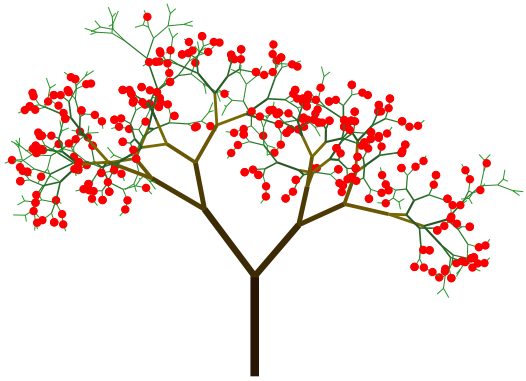
- *Weighted average*



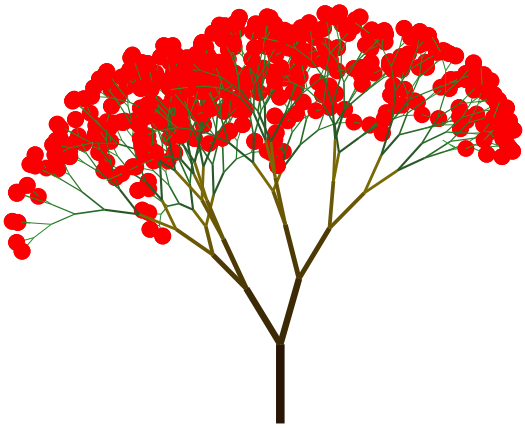
Parent 1 phenotype



Parent 2 phenotype

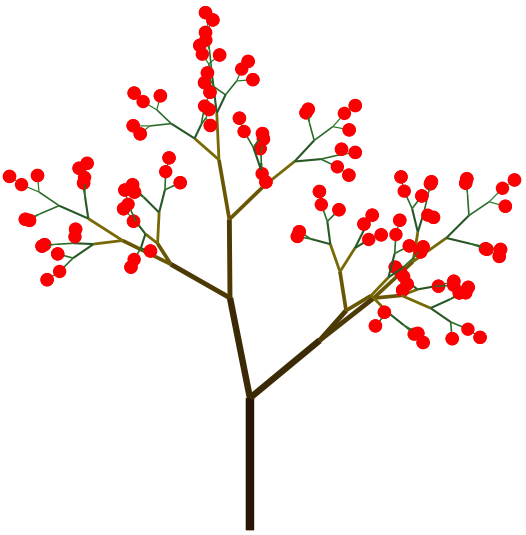


Child 1 phenotype

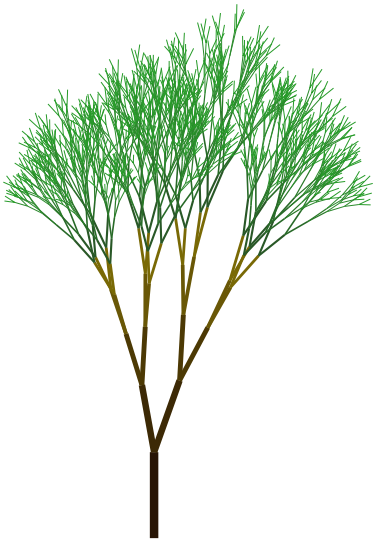


Child 2 phenotype

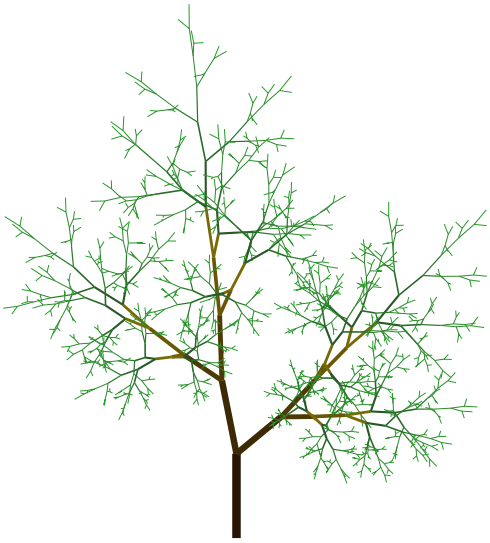
Figure 4: Sample *Single-point crossover* operation.



Parent 1 phenotype

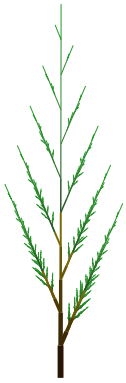


Parent 2 phenotype



Child 1 phenotype

Figure 5: Sample *Multi-point crossover* operation.



Parent 1 phenotype

Parent 2 phenotype

Child phenotype

Figure 6: Sample *Weighted average* operation.

Input: two distinct genotypes (parents), a real contribution ratio $x \in (0, 1)$
Output: a single genotype (child)
Description: Two genotypes can be averaged mathematically to obtain a new individual based on a given contribution ratio. The contribution ratio determines how much each parent's genes affect the offspring.

Process:

1. multiple the vector which represents parent 1's genotype by x
2. multiple the vector which represents parent 2's genotype by $(1 - x)$
3. add the resultant vectors to obtain the child genotype (vector)

Example (contribution ratio = 0.70):
Parent 1 genotype

-0.196	-0.001	-0.974	0.864	-0.238	-0.507	0.007

Parent 2 genotype

0.027	0.135	-0.749	1.05	-0.265	-0.355	-0.209

Parent 1 (vector) × contribution ratio = 0.70

-0.137	-0.001	-0.682	0.605	-0.167	-0.355	0.005

+

Parent 2 (vector) × (1 - contribution ratio) = 0.30

0.008	0.041	-0.225	0.315	-0.080	-0.107	-0.063

=

Child genotype

-0.129	0.040	-0.907	0.920	-0.247	-0.462	-0.058

See Figure 6 for the corresponding phenotypes.

- *Multi-parent mutation*

Input: two or more distinct genotypes (parents)
Output: a single genotype (child)
Description: To each parameter in the child genotype, we assign a random number which fits a gaussian distribution whose mean and standard deviation are determined by the corresponding parameters of the parent genotypes.

Process:

1. calculate the average value of the first gene (based on all of the parents).
2. calculate the standard deviation between all of the parents' values for the first gene.
3. generate a random number which fits a gaussian distribution with the mean and standard deviation calculated in steps 1 and 2.
4. assign this number to the first gene of the child genotype
5. repeat this process for each of the seven parameters

Example (3 parents):
Parent 1 genotype

0.111	-0.549	0.453	-0.645	0.493	-0.083	-0.052

Parent 2 genotype

0.049	-0.319	0.019	-0.494	0.389	0.004	0.183

Parent 3 genotype

0.018	0.563	0.342	-0.318	-0.354	0.181	-0.322

Parent mean

0.059	-0.102	0.271	-0.485	0.176	0.034	-0.064

Parent standard deviation

0.039	0.479	0.184	0.134	0.377	0.110	0.206

Child genotype

-0.007	-0.238	0.521	-0.449	0.567	0.059	0.068

See Figure 7 for the corresponding phenotypes.

2.3 The User Interface

The user interface consists of two *Cpfg* windows and the *evolve* application. The **current population** window (Figure 8, right) displays the nine individuals which comprise the current population. The **enlarge** window (Figure 8, top-left) can be used to display an enlarged copy of any individual in the **current population** window (both *Cpfg* windows can be resized and the individual in the **enlarge** window can be rotated and viewed in 3D). For details on the operation of *Cpfg* see [2]. Specifics regarding the L-system and view files used to implement the **enlarge** and **current population** windows are presented in Appendix A.

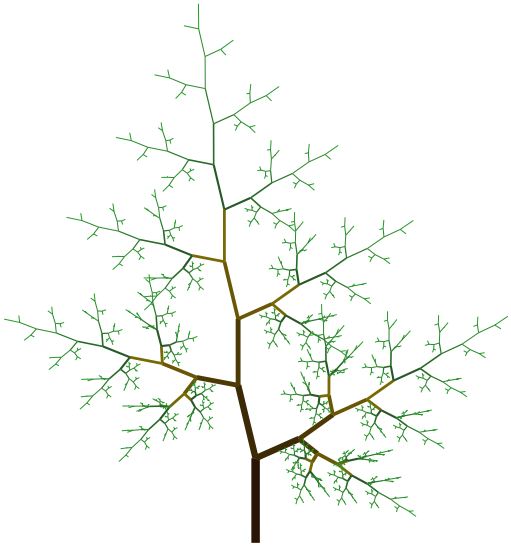
Simulated evolution proceeds as follows: the user chooses one or more individuals from the **current population** window by clicking on the corresponding keypad buttons. Then the user selects a genetic operation (choice of operation is restricted to only those which are applicable given the number of parents currently selected). The Mutation factor and parental contribution can be adjusted if desired. Finally, the selected genetic operation is applied to the currently selected individuals resulting in a new population. For a more detailed description of the *evolve* application controls see Appendix B.

3 The Results

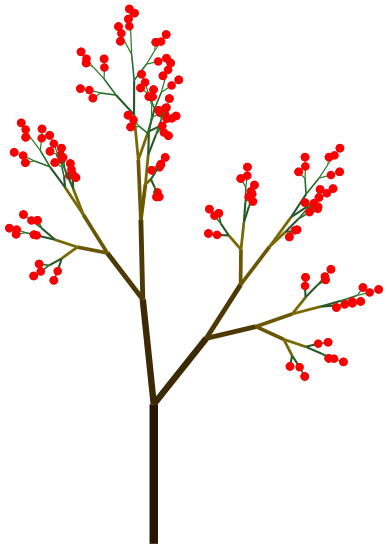
This section will highlight some of the observations made while experimenting with the model described in Section 2. A sample evolution will be examined, observations presented, and future work considered.

3.1 Sample Evolution

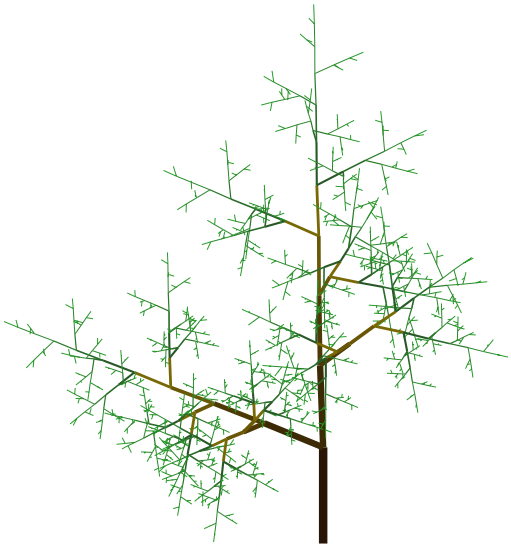
Figure 3.1 illustrates the process of user-guided evolution within the context of the model presented in Section 2. The goal of this particular evolution was to generate a monopodial structure with fruit in its branches.



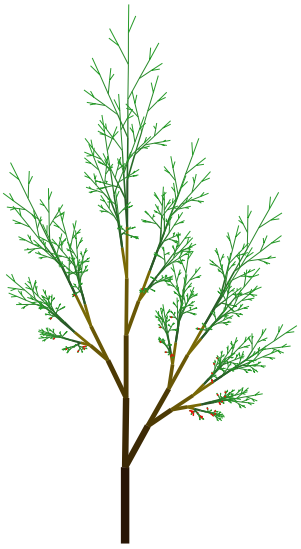
Parent 1 phenotype



Parent 2 phenotype



Parent 3 phenotype



Child phenotype

Figure 7: Sample *Multi-parent mutation* operation.

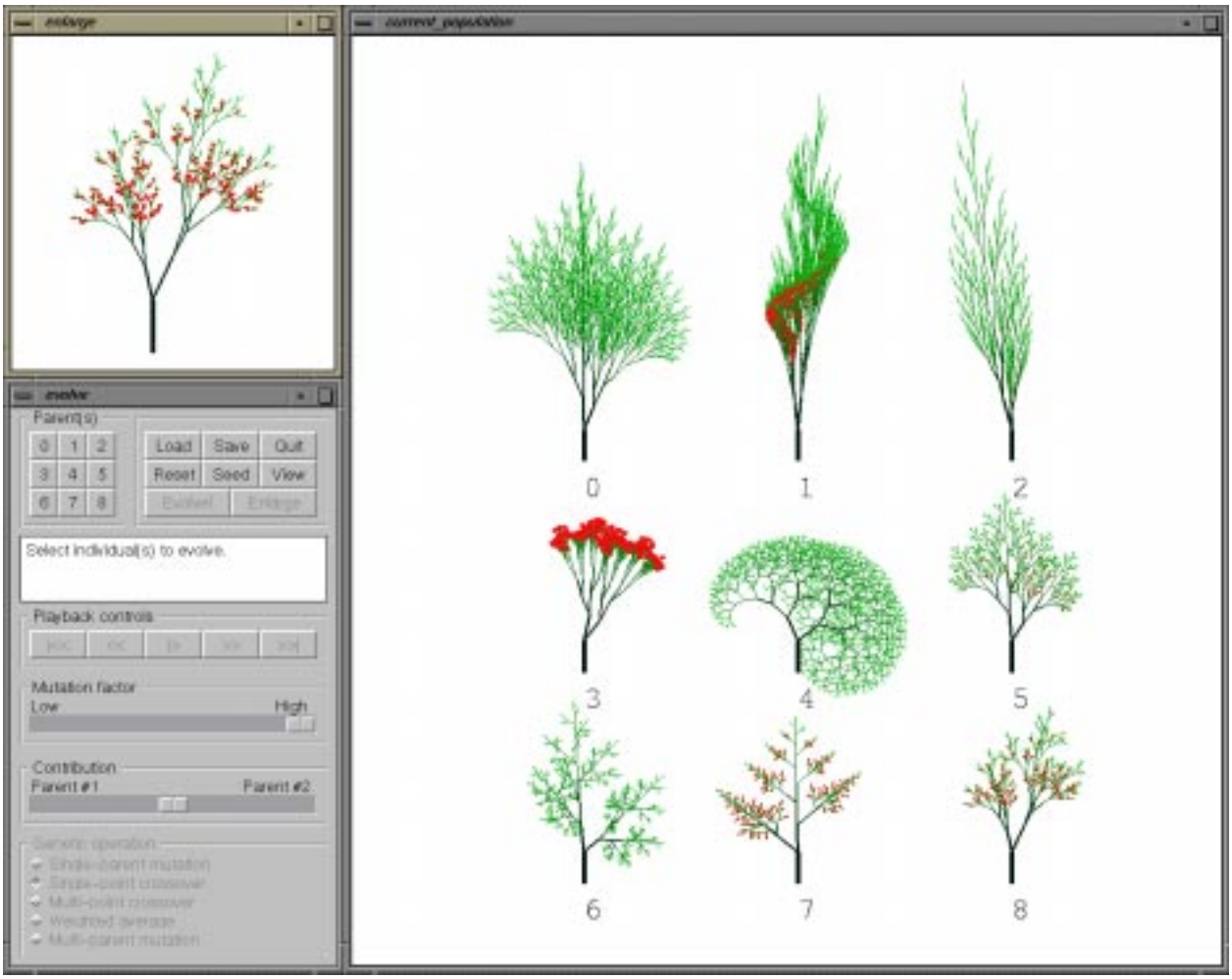
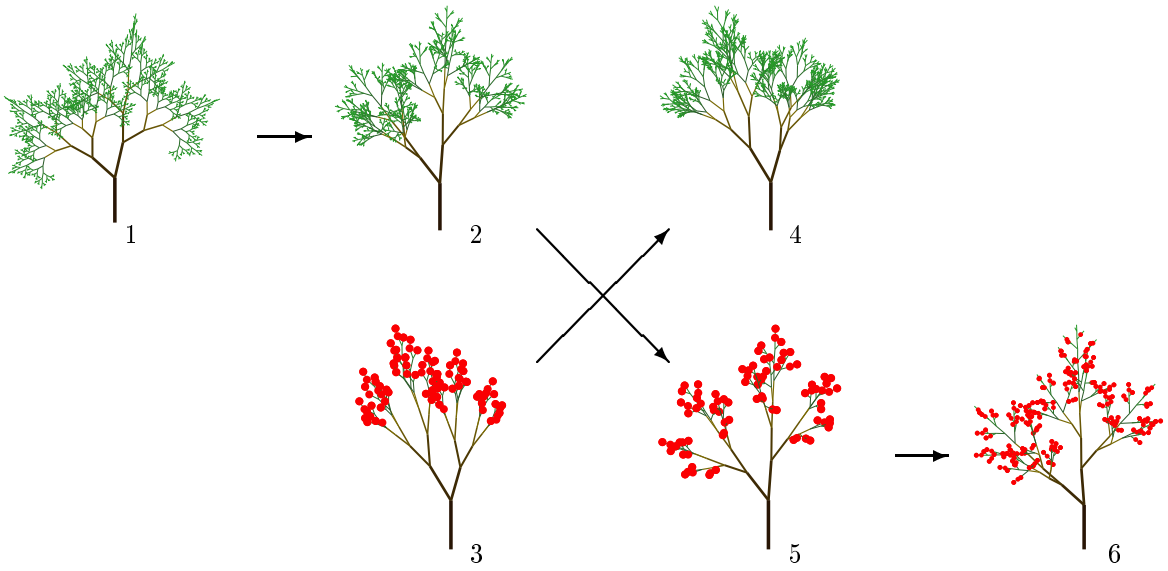


Figure 8: User interface for evolution experiments.



Structure	ANG1	ANG2	RATE1	RATE2	DIV1	DIV2	MINSIZE
1	-0.107	0.399	0.195	-0.271	0.462	-0.488	-0.389
2	-0.034	0.315	0.203	-0.235	0.570	-0.190	-0.596
3	0.260	-0.132	0.246	-0.141	-0.408	0.085	0.490
4	0.260	-0.132	0.246	-0.235	0.570	-0.190	-0.596
5	-0.034	0.315	0.203	-0.141	-0.408	0.085	0.490
6	0.036	0.398	0.256	-0.250	-0.574	0.211	0.328

Figure 9: Sample evolution of a monopodial fruit bearing structure

Individual 1 was chosen from nine random phenotypes because it exhibited the most pronounced main axis. Several mutations of individual 1 were created and individual 2 was then selected. Individual 3 was introduced (from an independent evolution) because it was a fruit bearing structure. Individuals 2 and 3 were then crossed in order to obtain offspring that might produce fruit and exhibit monopodial structure. The second child (individual 5) captured both of these characteristics. Mutations of individual 5 were created in search of a structure with greater branch density. Individual 6 was selected from these mutations; thus concluding the evolution.

In terms of the genetic operations, individual 2 was obtained from individual 1 by applying the *Single-parent mutation* operation using a mutation factor of 0.68. Individuals 4 and 5 were the results of a *Single-point crossover* operation (with crossover point, $x = 3$) applied to individuals 2 and 3. *Single-parent mutation* was used again with a mutation factor of 0.22 to obtain individual 6 from individual 5.

3.2 Observations

Positive Aspects of model

L-system (3) is a relatively simple L-system; nevertheless, it is capable of generating aesthetically pleasing plant models which exhibit realistic branching patterns. Using the model described in Section 2 it is possible to evolve both monopodial and sympodial structures.

The user interface is successful in allowing interactive user-guided evolution of simple plant models.

Possible improvements

Organs such as leaves and flowers play an important role in determining a plant's appearance. The current

model focuses primarily on the branching structure of the plant rather than its organs. In order to improve the realism of phenotypes it will be necessary to model plant organs as well.

The current model does not capture any developmental properties of plants. For example, once branch segments are formed their lengths do not change. Producing developmental models will require taking into account additional aspects which have been neglected in this simple model.

Genetic operations

Sometimes mutations may result in offspring bearing no resemblance to their ancestors; this may indicate that the mutation factor is too large. In general, greater success is obtained when the mutation factor is kept small. When the mutation factor is too large it is difficult to control the direction of evolution (since each mutation of the genotype results in such large changes of the phenotype). There is a tradeoff between the rate of evolution and the amount of control that can be exerted over it. For instance, with a high mutation factor it is possible to generate new plants which are very different from the parent, but there is little control over what characteristics these plants will exhibit. Using a small mutation factor, the offspring generated are only slight variations of the parent. Over several generations this allows more opportunity to direct the evolution of the plants.

On occasion a genetic operation will yield offspring identical to the parent phenotype(s). Usually this results from a *Single-parent mutation* where the mutation factor is too low to produce any visible changes in the phenotype. While experimenting with this model it was discovered that a *Single-point crossover* can produce the same phenomena. How does this come about? Let capital letters $A, B, C, \ldots$ denote genotypes. Each genotype is an ordered set of seven genes. For example $A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$ For simplicity the brackets and commas are omitted. Consider the following situation.

A	B
$a_1a_2a_3a_4a_5a_6a_7$	$b_1b_2b_3b_4b_5b_6b_7$
C	D
$a_1a_2a_3b_4b_5b_6b_7$	$b_1b_2b_3a_4a_5a_6a_7$

A	C
$a_1a_2a_3a_4a_5a_6a_7$	$a_1a_2a_3b_4b_5b_6b_7$
E	F
$a_1a_2a_3b_4b_5b_6b_7$	$a_1a_2a_3a_4a_5a_6a_7$

The *Single-point crossover* operation (crossover point $x = 3$) is applied to the two parents A and B resulting in two children C and D. At some point individual A and C are crossed (again using the *Single-point crossover* operation) using the same crossover point $x = 3$. This will result in two offspring identical to the parents `(E = C, F = A)`. Incestual crossovers are those in which a child is crossed with one of it parents. The results of such crossovers will not yield new offspring unless a different crossover point is used.

The mutation operation is necessary and sufficient for the genetic algorithm to function; however, crossover operations provide shortcuts which allow the user to combine characteristics of several phenotypes into one.

Although the results of crossover operations are not always predictable, in most cases it is possible to produce offspring which exhibit characteristics of both parents. However, it is sometimes necessary to try different combinations of crossover operation and contribution ratio, to obtain the desired result.

3.3 Extension to the Evolution of Surfaces

Bicubic surfaces (patches) are often used in modeling plant organs such as petals and leaves. This section briefly describes how the genetic algorithm (from Section 2.2) and the user interface (from Section 2.3) can be used to evolve surfaces. Some preliminary experiments have been performed; the current model and some results will now be presented.

The model

Figure 10: A "leaf" generated using an unrestricted model where the genotype consists of 16 control points.

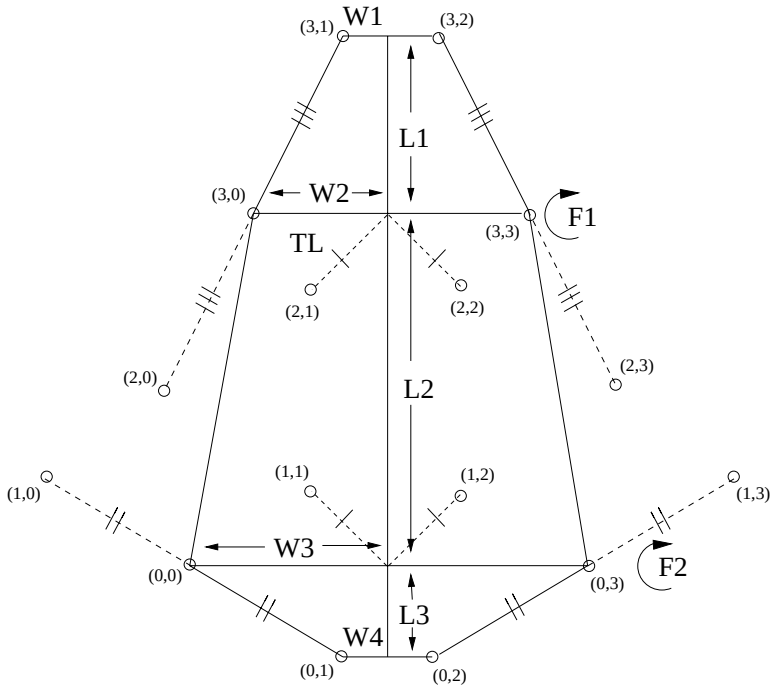


Figure 11: Current model of the surface control structure.

There are several types of Bicubic surfaces; this particular model uses Bézier patches [12, Section 13.6] to model leaves and petals. The use of developmental bicubic patches in L-systems is detailed in [13, Section 4.3]. Bézier patches are defined by 16 control points. Each of these control points is specified by an x, y, and z coordinate. The mathematics behind Bézier surfaces is not covered here but this information can be found in [12, Section 13.6].

One possible approach to creating a parametric organ model would be to treat each coordinate of each control point as a "gene". This would yield a genotype of 48 parameters. Each phenotype would then be the surface defined by the 16 control points. This approach was tested but the results were disappointing. Without any restriction on the genes, the phenotypes produced did not at all resemble leaves or petals (see Figure 10). One of the major problems with this approach is that the organs have no symmetry. In nature it is common to find leaves and petals which are symmetrical. The model can be simplified and possibly improved by forcing the surfaces to be symmetrical about some axis. The model described by Jim Hanan in [13, section 4.3] satisfies the criterion of symmetry. Using Hanan's model as a starting point the following model (see Figure 11) was developed.

The genotype contains ten genes which determine the shape of a surface control structure (Figure 11). The surface control structure is responsible for placement of each of the 16 control points. These control points

Surface	CC	L1	L2	L3	W1	W2	W3	W4	F1	F2
0	-0.380	0.165	-0.377	-0.203	0.013	-0.042	0.148	0.104	0.251	-0.373
1	0.448	0.485	0.441	0.392	0.104	-0.319	0.335	-0.166	0.140	-0.213
2	0.747	-0.637	-0.564	-0.537	0.012	0.046	0.082	0.064	0.195	-0.140
3	-0.080	-0.183	-0.169	0.831	-0.413	-0.172	0.170	0.040	-0.387	0.256
4	0.018	0.535	-0.132	0.389	-0.023	-0.025	0.375	0.356	-0.301	-0.522
5	-0.127	0.456	0.377	-0.414	0.147	0.304	-0.087	-0.054	0.457	-0.472
6	-0.352	0.101	-0.429	-0.290	-0.046	-0.005	0.061	0.043	0.337	-0.454
7	0.077	0.246	-0.259	-0.388	-0.439	0.136	0.324	-0.003	0.338	-0.505
8	0.060	0.403	0.438	0.093	0.013	0.192	-0.090	0.297	-0.197	-0.002

Table 2: Genotypes corresponding to surfaces in Figure 12.

will then specify a Bézier surface, hopefully resembling a petal or leaf. Appendix A contains the L-system and view files (enlarged_surface.l, surfaces.l and surfaces.v) used to implement this model. The parameters can be summarized as follows. Parameter CC is used to control how convex the surface will be. L1, L2, L3 specify the lengths of the base, center, and tip segments of the control structure. W1-W4 affect the width of the control structure at various locations. F1 and F2 will determine the amount of bending in the lower and upper portions of the surface. TL is not a parameter but a value calculated within the L-system. It is dependent on the convexity constant CC, and related to the parameters L1-L1, W1-W4.

The results

Figure 12 shows several of the phenotypes which were evolved. Table 2 contains the corresponding numerical genotypes. Overall, the model developed in Section 2 extends well to surfaces. It is possible to evolve quite a variety of petal and leaf shapes in a reasonably intuitive manner. If we know exactly what we are looking for (in terms of petal or leaf shape), it is more efficient to use a 3D surface editor to design our surfaces. Using the *evolve* application tends to be more a of creative process then a mechanical one.

3.4 Future Work

This section describes some of the possible directions this research may take. The current goal is to improve the realism and variety of plants which can be generated, without sacrificing the ideology of the user interface.

The arrangement of organs within a plant seem to follow some type of pattern or obey some kind of symmetry. Take for example the leaves on a stem or the seeds of a sunflower; both of these exhibit pattern and regularity. The arrangement of plant organs is clearly not random. Observation of these patterns suggests the existence of a mathematical model which might describe the apparent regularity. This regular arrangement of lateral organs is known as *phyllotaxis*, and there are in fact several models of this phenomenon (see [1, Chapter 4]). Since phyllotactic models are mathematical in nature they are also good candidates for parameterization and hence a logical extension of the work discussed in this paper.

Once the techniques in this chapter have been applied to phyllotactic models, it would be nice to see if L-System branching structures, Bézier surfaces, and phyllotactic patterns could be combined into a single model. The results of which could be quite interesting.

The next step is to look beyond parametric models. Focusing on the development of plant form from a more biological perspective may allow the creation of more realistic models. Coen [14] presents a model describing the development and differentiation of organs within a plant. As a basis for future models this theory will ultimately result in more biologically accurate simulations of plant development. Developing these new models will require that new genetic operations be implemented; operations that act not only on parameters but on the L-system productions themselves.

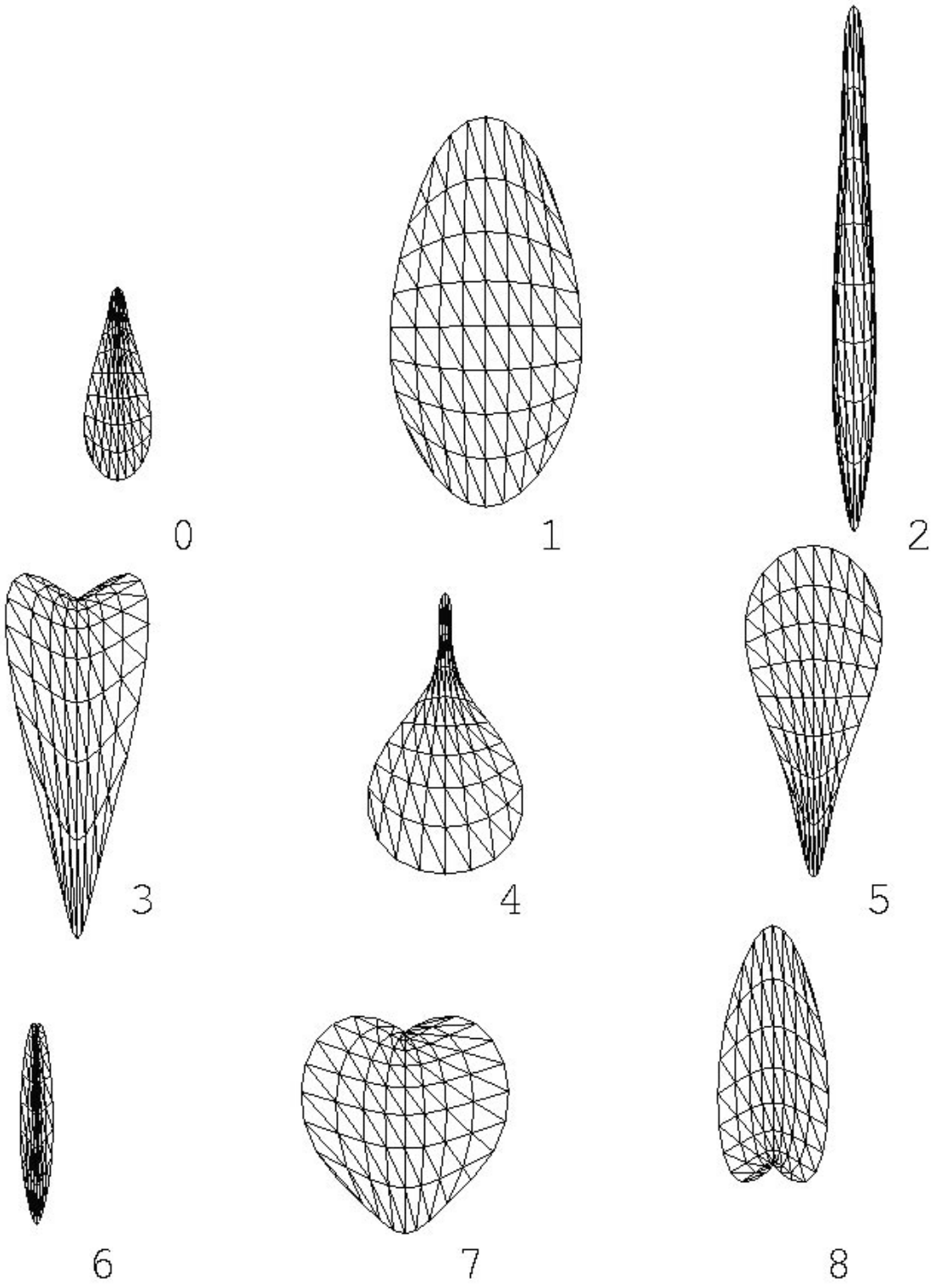


Figure 12: Collection of phenotypes generated using the structural model presented in Figure 11.

A Implementation

A.1 L-system files

enlarged.l

```
#define STEPS 11
#define W0 5
#define a 0.5
#define b 0.71
#define e 0.40

Lsystem: 0

Start: {
                fp = fopen("enlarged.p", "r");
                fscanf(fp, "%lf", &ANG1);
                fscanf(fp, "%lf", &ANG2);
                fscanf(fp, "%lf", &RATE1);
                fscanf(fp, "%lf", &RATE2);
                fscanf(fp, "%lf", &DIV1);
                fscanf(fp, "%lf", &DIV2);
                fscanf(fp, "%lf", &MINSIZE);
                fclose(fp);
        }

derivation length: STEPS
Axiom: A(0.3,W0)

A(s,w) : s>= MINSIZE*0.1  --> !(w)F(s)!(w*b);
         [+(ANG1*120)/(DIV1*360)A(s*(0.75+0.3*RATE1),w*a^e)]
         [+(ANG2*120)/(DIV2*360)A(s*(0.75+0.3*RATE2),w*(1-a)^e)]
A(s,w) : s<MINSIZE*0.1  --> ;(15)@O(0.1*MINSIZE)

endlsystem
```

current.l

```
#define STEPS 12
#define W0 5
#define a 0.5
#define b 0.71
#define e 0.40

Lsystem: 0

Define: {
                array xc[9] = {0,1,2,0,1,2,0,1,2};
                array yc[9] = {0,0,0,1,1,1,2,2,2};
        }

Define: {
                array ANG1[9],
```

```
                ANG2[9],
                RATE1[9],
                RATE2[9],
                DIV1[9],
                DIV2[9],
                MINSIZE[9];
        }

Start: {
                fp = fopen("individuals.p", "r");
                fscanf(fp, "%d", &NUMS);
                fscanf(fp, "%lf", &FACTOR);
                i = 0;
                while(i < 9)
                {
                        fscanf(fp, "%lf", &ANG1[i]);
                        fscanf(fp, "%lf", &ANG2[i]);
                        fscanf(fp, "%lf", &RATE1[i]);
                        fscanf(fp, "%lf", &RATE2[i]);
                        fscanf(fp, "%lf", &DIV1[i]);
                        fscanf(fp, "%lf", &DIV2[i]);
                        fscanf(fp, "%lf", &MINSIZE[i]);
                i = i + 1;
                }
                fclose(fp);
        }

derivation length: STEPS
Axiom: Z(0)Z(1)Z(2)Z(3)Z(4)Z(5)Z(6)Z(7)Z(8)

Z(i) : NUMS != 0 --> [-(90)f(xc[i]*FACTOR)-(90)f(yc[i]*FACTOR+(0.5*FACTOR/3))|
       @L("%.f",i)f(0.5*FACTOR/3)?(1,1)A(0.3,W0,i)$]
Z(i) : NUMS == 0  --> [-(90)f(xc[i]*FACTOR)-(90)f(yc[i]*FACTOR)|?(1,1)A(0.3,W0,i)$]

endlsystem

Lsystem: 1

derivation length: 1
Axiom: A(1,W0,1)

A(s,w,i) : s>= MINSIZE[i]*0.1  --> !(w)F(s)!(w*b);
           [+(ANG1[i]*120)/(DIV1[i]*360)A(s*(0.75+0.3*RATE1[i]),w*a^e,i)]
           [+(ANG2[i]*120)/(DIV2[i]*360)A(s*(0.75+0.3*RATE2[i]),w*(1-a)^e,i)]
A(s,w,i) : s<MINSIZE[i]*0.1  --> ;(15)@O(0.1*MINSIZE[i])

endlsystem
```

enlarged_surface.l

```
#define DBZE 0.000001

Lsystem: 0
```

```
Start: {
                fp = fopen("enlarged.p", "r");
                fscanf(fp, "%lf",  &CC);
                fscanf(fp, "%lf",  &L1);
                fscanf(fp, "%lf",  &L2);
                fscanf(fp, "%lf",  &L3);
                fscanf(fp, "%lf",  &W1);
                fscanf(fp, "%lf",  &W2);
                fscanf(fp, "%lf",  &W3);
                fscanf(fp, "%lf",  &W4);
                fscanf(fp, "%lf",  &F1);
                fscanf(fp, "%lf",  &F2);
                fclose(fp);

                L1 = fabs(L1*10);
                L2 = fabs(L2*10);
                L3 = fabs(L3*10);
                W1 = fabs(W1*10);
                W2 = fabs(W2*10);
                W3 = fabs(W3*10);
                W4 = fabs(W4*10);
                F1 = F1*45;
                F2 = F2*45;

                A1 = atan(L1/(W2-W1+DBZE));
                S1 = sqrt(L1^2+(W2-W1)^2);
                A2 = atan(L3/(W3-W4+DBZE));
                S2 = sqrt(L3^2+(W3-W4)^2);
                TL = CC*sqrt((L1+L2+L3)^2+(W1+W2+W3+W4/2)^2)/4;
        }

derivation length: 2
Axiom: P

P --> [S[l][r]B[L][R]D]
S --> @PS(0)
B --> f(L3)^(F2)[+(90)^(-45)f(TL);@PC(0,1,1)][-(90)^(-45)f(TL);@PC(0,1,2)]
      f(L2)^(F2)[+(90)^(-45)f(TL);@PC(0,2,1)][-(90)^(-45)f(TL);@PC(0,2,2)]f(L1)
D --> ;(7)@PD(0,8,8);

l --> +(90)f(W3);@PC(0,0,0)[-(A2)f(S2)@PC(0,1,0)][+(180-A2)f(S2)@PC(0,0,1)]
r --> -(90)f(W3);@PC(0,0,3)[+(A2)f(S2)@PC(0,1,3)][-(180-A2)f(S2)@PC(0,0,2)]
L --> +(90)f(W2);@PC(0,3,0)[+(A1)f(S1)@PC(0,2,0)][-(180-A1)f(S1)@PC(0,3,1)]
R --> -(90)f(W2);@PC(0,3,3)[-(A1)f(S1)@PC(0,2,3)][+(180-A1)f(S1)@PC(0,3,2)]

homomorphism
maximum depth: 1

@PC(i,j,k) --> @O(.05)@L("%.f,%.f",j,k)@PC(i,j,k)
f(x) --> F(x)

endlsystem
```

surfaces.l

```
#define DBZE 0.000001

Lsystem: 0
Define: {
                array xsc[9] = {0,1,2,0,1,2,0,1,2};
                array ysc[9] = {0,0,0,1,1,1,2,2,2};
        }

Define: {
                array CC[9],
                L1[9],
                L2[9],
                L3[9],
                W1[9],
                W2[9],
                W3[9],
                W4[9],
                F1[9],
                F2[9],
                A1[9],
                A2[9],
                S1[9],
                S2[9],
                TL[9];
        }

Start: {
                fp = fopen("individuals.p", "r");
                fscanf(fp, "%d", &NUMS);
                fscanf(fp, "%lf", &FACTOR);
                i = 0;
                while(i < 9)
                {
                        fscanf(fp, "%lf",  &CC[i]);
                        fscanf(fp, "%lf",  &L1[i]);
                        fscanf(fp, "%lf",  &L2[i]);
                        fscanf(fp, "%lf",  &L3[i]);
                        fscanf(fp, "%lf",  &W1[i]);
                        fscanf(fp, "%lf",  &W2[i]);
                        fscanf(fp, "%lf",  &W3[i]);
                        fscanf(fp, "%lf",  &W4[i]);
                        fscanf(fp, "%lf",  &F1[i]);
                        fscanf(fp, "%lf",  &F2[i]);

                        L1[i] = fabs(L1[i]*10);
                        L2[i] = fabs(L2[i]*10);
                        L3[i] = fabs(L3[i]*10);
                        W1[i] = fabs(W1[i]*10);
                        W2[i] = fabs(W2[i]*10);
                        W3[i] = fabs(W3[i]*10);
                        W4[i] = fabs(W4[i]*10);
                        F1[i] = F1[i]*45;
                        F2[i] = F2[i]*45;
```

```
                        A1[i] = atan(L1[i]/(W2[i]-W1[i]+DBZE));
                        S1[i] = sqrt(L1[i]^2+(W2[i]-W1[i])^2);
                        A2[i] = atan(L3[i]/(W3[i]-W4[i]+DBZE));
                        S2[i] = sqrt(L3[i]^2+(W3[i]-W4[i])^2);

                        TL[i] = CC[i]*sqrt((L1[i]+L2[i]+L3[i])^2+
                                (W1[i]+W2[i]+W3[i]+W4[i]/2)^2)/4;

                        i = i + 1;
                }
                fclose(fp);
                FACTOR = FACTOR + 10;
        }

derivation length: 3
Axiom: Z(0)Z(1)Z(2)Z(3)Z(4)Z(5)Z(6)Z(7)Z(8)

Z(i) : NUMS != 0 --> [-(90)f(xsc[i]*FACTOR)-(90)f(ysc[i]*FACTOR+(0.5*FACTOR/3))|
       @L("%.f",i)f(0.5*FACTOR/3)?(1,1)P(i)$]
Z(i) : NUMS == 0  --> [-(90)f(xsc[i]*FACTOR)-(90)f(ysc[i]*FACTOR)|?(1,1)P(i)$]

endlsystem

Lsystem: 1

derivation length: 1
Axiom: P(i)

P(i) --> [S(i)[l(i)][r(i)]B(i)[L(i)][R(i)]D(i)]
S(i) --> @PS(i)
B(i) --> f(L3[i])^(F2[i])[+(90)^(-45)f(TL[i]);@PC(i,1,1)][-(90)^(-45)f(TL[i]);
         @PC(i,1,2)]f(L2[i])^(F2[i])[+(90)^(-45)f(TL[i]);@PC(i,2,1)][-(90)^(-45)
         f(TL[i]);@PC(i,2,2)]f(L1[i])
D(i) --> ;(7)@PD(i,8,8);

l(i) --> +(90)f(W3[i]);@PC(i,0,0)[-(A2[i])f(S2[i])
         @PC(i,1,0)][+(180-A2[i])f(S2[i])@PC(i,0,1)]
r(i) --> -(90)f(W3[i]);@PC(i,0,3)[+(A2[i])f(S2[i])
         @PC(i,1,3)][-(180-A2[i])f(S2[i])@PC(i,0,2)]
L(i) --> +(90)f(W2[i]);@PC(i,3,0)[+(A1[i])f(S1[i])
         @PC(i,2,0)][-(180-A1[i])f(S1[i])@PC(i,3,1)]
R(i) --> -(90)f(W2[i]);@PC(i,3,3)[-(A1[i])f(S1[i])
         @PC(i,2,3)][+(180-A1[i])f(S1[i])@PC(i,3,2)]

endlsystem
```

A.2 View file

evolve.v

```
angle factor: 4
initial color: 1
color increment: 1
```

```
initial line width: 2 pixels
line width increment: 0
viewpoint: 0,0,1
view reference point: 0,0,0
twist: 0
projection: parallel
front distance: -100000.0
back distance: 100000.0
scale factor: 0.9
z buffer: on
cue range: 0
font: -*-courier-medium-r-normal--34-*-*-*-m-200-iso8859-1
shade mode: 1
light direction: 1.0, 1.0, 1.0
diffuse reflection: 0
tropism direction: 0.0,1.0,0.0
initial elasticity: 0.0
elasticity increment: 0.0
```

surfaces.v

```
angle factor: 4
initial color: 16
color increment: 2
initial line width: 0.4
line width increment: 0.1
viewpoint: 0,0,1
view reference point: 0,0,0
twist: 0
projection: parallel
front distance: -10000.0
back distance: 10000.0
scale factor: 0.9
z buffer: on
cue range: 0
font: -*-courier-medium-r-normal--34-*-*-*-m-200-iso8859-1
shade mode: 3
light direction: 1.0,0.0,0.5
diffuse reflection: 10
tropism direction: 0.0000,-1.0000,0.0000
initial elasticity: 0.1100
elasticity increment: 0.0000
surface ambient: 0.75
surface diffuse: 0.9
```

B Evolve controls

The **Contribution** slider bar controls the amount of genetic information each parent contributes to their offspring. This control only applys to certain genetic operations (*Single-point crossover*, *Multi-point crossover*, and *Weighted average*).

The **Enlarge** button can be used to display a larger copy of any individual in the current population. Note

that this button is active only when a single parent is selected. Use the **parent selection** keypad to select which individual to enlarge and then click the **Enlarge** button.

Clicking the **Evolve** button applys the currently selected genetic operation to the currently selected individuals.

The **Genetic operation** selection box allows the user to specify which genetic operation will be applied the next time the **Evolve** button is pressed.

Clicking the **Load** button will allow the user to load either an individual, a generation, or an entire evolution. When performing a load operation, the user is prompted to choose the type of load (load an individual, load a generation, or load an evolution). Once the user specifies what type of load they wish to perform a file selection dialog box will open. The user then selects which file to load and clicks okay, completing the process.

The **Mutation factor** slider bar is used to control the magnitude of mutations which result when a *Single-parent mutation* operation is applied.

The **Parent selection** numeric keypad is used to select which individuals will be involved in the next genetic operation. The number of parents currently selected determines which functions can be performed and which genetic operations can be applied. To select an individual click on the numbered key corresponding to that member of the population. When an individual is selected the button will remain in the down position. An individual can be unselected by clicking it a second time.

The **Playback** controls are a panel of five playback buttons which can be used to navigate through the current evolution. The associated operations allow the user to :

1. Rewind the evolution to the beginning.
2. Move backwards a single generation in the evolution.
3. Playback the evolution starting from from the current generation.
4. Move forward a single generation.
5. Fast forward to the last generation.

The **Quit** button is used to exit the evolve application.

Clicking the **Reset** button will replace the existing population with nine new randomly created individuals.

The **Save** button is used to save either an individual, a generation, or an entire evolution. When performing a save operation, the user is prompted to choose the type of save (save an individual, save a generation, or save an evolution). Once the user specifies what type of save they wish to perform a file selection dialog box will open. The user enters a new filename or selects an existing file and clicks okay.

The **Seed** button is used to seed the the random number generator. This is useful because it allows experiments to be repeated with the same set of random numbers. Clicking the **seed** button will open an entry dialog box. Enter an integer in the range $[-2_{31}, 2_{31})$ and click okay.

Clicking the **View** button opens a dialog box that can be used to toggle enumeration or adjust spacing of the individuals displayed in the current population window.

References

[1] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants.* Springer Verlag, 1990.

[2] R. Měch. CPFG Version 3.4 User's Manual. Department of Computer Science, University of Calgary, 1998.

[3] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[4] D. Goldberg *Genetic Algorithms in Search, Optimization, and Machine Learning* Addison-Wesley, 1989.

[5] M. Mitchell. *An Introduction to Genetic Algorithms.* The MIT Press, 1996.

[6] R. Dawkins *The Blind Watchmaker.* W.W. Norton & Company, 1986.

[7] C. Darwin. *On the Origin of Species by Means of Natural Selection.* Penguin Paperbacks, 1859.

[8] G. Rozenberg & A. Salomaa (Eds.) *Handbook of Formal Languages - Volume 3 - Beyond Words.* Springer.

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

[10] K. Sims. Artificial Evolution for Computer Graphics Computer Graphics, Vol. 25, No. 4, July 1991, pp.319-328.

[11] J. McCormack. Interactive Evolution of L-System Grammars for Computer Graphics Modeling. Computer Science Dept. Monash University, AUSTRALIA

[12] J. Foley, A. Van Dam. *Fundamentals of Interactive Computer graphics.* Addison-Wesley, 1984.

[13] R. Měch, P. Prusinkiewicz, J. Hanan. Extensions to the graphical interpretation of L-systems based on turtle geometry. Department of Computer Science, University of Calgary, 1998.

[14] E. Coen *The Art of Genes.* Oxford University Press, 1999.