

# Animation of Plant Development

## *229 Final Project Report*

Olga Karpenko

December 14, 2000

## 1 Overview

For 229 final project I studied L-systems that have been extensively used for generating plants, and implemented L-systems engine which generates, parses and displays L-strings. It includes the following capabilities:

- Extended 2D and 3D turtle interpretation
- Parametric L-systems
- Context-sensitive L-systems
- Exporting pre-defined models of plant organs as meshes in sm-format

I also implemented a simple version of the algorithm from the paper [1] for animating plants which combines discrete and continuous models of plant development. The system can be used for simulating growth process of several kinds of plants.

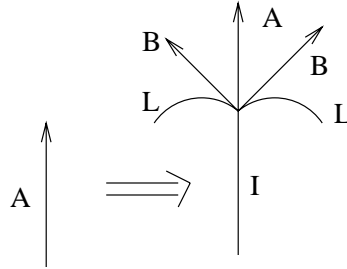
## 2 Background

Biological phenomenon of self-similarity in plants gave Lindemayer the idea of capturing the developmental processes in plants using special kind of grammars. He introduced L-systems in 1968, suggesting using them as a simplified theoretical model of the development of simple multicellular organisms. Since then, L-systems were developed a lot and applied to modeling of higher plants and trees [1],[3],[5].

### 2.1 L-systems: Introduction

Each plant with more or less “regular” structure can be viewed as a branching structure consisting of modules. L-system is a way to describe this structure. It consists of productions which define the rules on replacing one module with others. Productions can be *context-free* if a rule depends only on the predecessor

module or *context-sensitive*, if it depends on the neighboring modules. On Fig.1 a simple example of a production rule is given. It says that the module A should be replaced with the branching structure on the right: internode I, new apex A and two buds B with leaves L. Here is the formal definition of the context free



deterministic L-system (DOL-system):

$G = \langle V, w, P \rangle$ , where  $V$  denotes the alphabet,  $w$  is a nonempty word called the axiom and belongs to  $V^+$  - the set of all non-empty words over  $V$  -alphabet,  $P \subset V \times V^*$  is a set of productions (finite). Production rules are of the form  $a \rightarrow \chi$ , and mean that a symbol  $a$  should be substituted by string  $\chi$ .  $a$  is called “predecessor”,  $b$  is called “successor”.

Unlike Chomsky grammars, where productions are applied sequentially, in L-systems they are applied in parallel. All letters in a given string to which productions are applicable, are replaced simultaneously. The difference is caused by biological application of L-systems: in organisms many divisions may happen in the same time.

## 2.2 Parametric and Stochastic L-systems

An extension of L-systems is *parametric L-systems*, where each module can have parameters like length, angle, etc. For example,  $M=A(2, 7.8)$  defines a module  $M$  of type  $A$  with two parameters. The way the parameters are interpreted depends on the semantics of the definition of the module. The general form of a production rule of a parametric L-system is:

*Predecessor(parameters) : Condition  $\rightarrow$  Successor.*

In the following example,  $F(x, t)$  is predecessor,  $t = 0$  and  $t > 0$  are conditions, formulas to the right of the arrows are successors.

$$w : F(1, 0)$$

$$p_1 : F(x, t) : t = 0 \rightarrow F(0.3 * x, 2) + F(0.7 * x, 1) - F(x, 0)$$

$$p_2 : F(x, t) : t > 0 \rightarrow F(x, t - 1)$$

If L-system is deterministic, the resulting plant will have too “regular” structure and look artificial. In order to add noise to specific details preserving the main structure of a plant and thus get more natural pictures, *stochastic L-systems* are used.

A stochastic context-free L-system is defined like a quadruplet  $G_\pi = \langle V, w, P, \pi \rangle$ ,

where  $V$  is the alphabet,  $w$  is the axiom,  $P$  is the set of the productions. Function  $\pi = P \rightarrow (0, 1]$  is the probability distribution function which assigns each production rule a probability.

### 2.3 Graphical Interpretation of L-systems

Prusinkiewicz used interpretation based on LOGO-style turtle [4]. Here is the basic idea of turtle interpretation:

A state of the turtle is the triple  $(x, y, \alpha)$ , where  $(x, y)$  - is the turtle position, and  $\alpha$  is the direction at which the turtle is facing. If we know angle increment  $\delta$  and step size  $d$ , then we describe the commands that can be given to the turtle as follows:

- F Move one step forward. New state is  $(x', y', \alpha)$ , where  $x' = x + d \cdot \cos\alpha$  and  $y' = y + d \cdot \sin\alpha$ . On the picture, a line segment between  $(x, y)$  and  $(x', y')$  is drawn.
- f Move a step forward without drawing a line.
- + Turn left by angle  $\delta$ . New state of the turtle is  $(x, y, \alpha + \delta)$ .
- - Turn right by angle  $\delta$ . New state of the turtle is  $(x, y, \alpha - \delta)$ .

So, when given the string  $v$  consisting of instructions as above, initial state  $(x_0, y_0, \alpha_0)$  and parameters  $d$  and  $\delta$ , the turtle can start walking creating the picture of its motion. The *turtle interpretation* of  $v$  is the figure drawn by the turtle in response to string  $v$ .

Prusinkiewicz used this idea to interpret strings generated by L-grammars. The basic notations for parametric L-system representation are these:

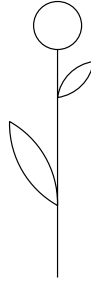
- F(x) line segment of length  $x$
- $+(\alpha)$ ,  $-(\alpha)$  orientation change of the following line by  $\pm\alpha$
- @X(s) a predefined surface X scaled by factor  $s$

Figure below shows the turtle interpretation of this sample L-grammar:  
 $F(1)[+45]@L(0.75)]F(0.8)[(-30)]@L(0.5)]F(0.6)]@K(1)$ , where @L and @K denote predefined figures of a leaf and a flower correspondingly, square brackets mean branch description.

### 2.4 dL-systems

For animation of plants, differential L-systems (dL-systems) are used. For dL-systems productions describe qualitative changes to the structure of plant, while differential equations (PODEs - piecewise-continuous ordinary differential equations) define continuous processes like gradual growth of branches.

As long as parameters  $w$  of a module  $A(w)$  remain within the domain of valid values  $D_A$ , the module develops in a continuous way, but once values reach  $C_A$  - the boundary of  $D_A$ , a module  $A$  will be replaced by its successors based on the production rule.



### 3 Animation

In traditional approach, the states of the L-system model are known only at discrete time intervals. There are several reasons why this model can not be used directly for plant animation:

- Once we decided about the time step, it becomes part of the model and we can not adapt it during simulation, while it is more preferable to be able to control it during animation.
- As we want plants to develop smoothly, we want to define some continuity criteria.
- Conceptually, it makes sense to separate *plant growth observation* happening at discrete time steps and *the process of plant growth* which is continuous.

The process of growth can be viewed as continuous expansion of modules with module branching happening at specific moments of a plant growth. Formally, this can be captured with differential L-system approach.

The state of the plant at specific time  $t$  is specified by a string:

$$\mu = A_1(w_1)A_2(w_2)\dots A_n(w_n).$$

The continuous behavior of  $A(w)$  is described by an ordinary differential equation:

$$\frac{dw}{dt} = f_A(w_l, w, w_r),$$

where  $w_l$  and  $w_r$  are the parameter vectors of the neighboring modules. This equation applies only while parameters lie in the domain of valid values. Once they reach the segment  $C_{A_k}$  of boundary  $C_A$  of  $D_A$  (let's assume, it is time  $t = t_\beta$ ), then an appropriate production rule is applied to the module A:

$$\lim_{t \rightarrow t_\beta} w(t) \in C_{A_k}$$

The production rule which replaces module A with its "branches":

$$p_{A_k} = A_l(w_l) < A(w) > A_r(w_r) \rightarrow B_{k,1}(w_{k,1}) \cdot B_{k,2}(w_{k,2}) \dots B_{k,m}(w_{k,m}).$$

So, we choose the production rule dependent on which piece of valid values boundary we crossed.

## 4 Design and Details of Implementation

The design of the system is depicted in Fig. 3.

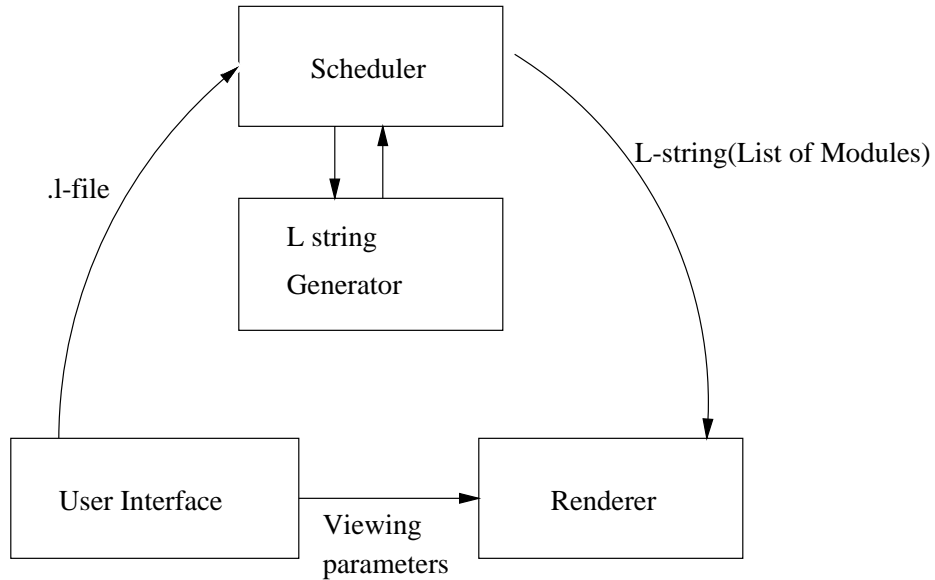


Fig. 3.

### 4.1 Static plants

The process of generating static plants consists of the following steps:

- Parse a file containing an L-grammar description for a particular plant and initialize L-system state
- Generate L-string by applying appropriate rules to the modules of the initial string (axiom). Repeat this number of iterations times.
- Parse final string using turtle interpretation described above and create a scenegraph for this model.
- Display final geometry.

Now we describe each of the above steps in more details. The input L-system file has the following general structure:

```

l-system: 0
num_iterations: 5
angle: 24.5
step: 1
radius: 0.05
ignore: +-F
axiom: XB(2)F(1,2)C(3,4)
rules:
F(x,n):
if x<5 produce A(x*3,n-2)F(x+4,n-1)
if x=5 produce F(2*x,1)[+F(x,2)]^(45)@(0.3)#(1)
endrule

B(y)<A(x,n)>C(z,t):
if x+y+z>=10 & t>0 produce B(y-1)A(1.5*x+z,n-1)
endrule

... //other rules
end

```

L-string is a list of modules (corresponding to different parts of a plant, like apex, internode, flower), each of them has a letter associated with it and can have any number of numeric parameters. After we created a list of modules of the axiom and gave the parameters specified initial values, we can start applying rules to it. As L-grammars are parallel rewriting systems, rules are applied to the modules in parallel. Each rule consists of a predecessor, a list of conditions and a list of productions. Predecessor may consist of left and/or right neighbors called the context of the module (for example, B and C are neighbors for the module A in the second rule) and the module itself (called *main*) which should match the one in the string. For each module of the original string, rule  $i$  applies to it, if :

1. Its predecessor *matches* the module (that is, module's letter is the same as the letter associated with *main* in the predecessor, number of parameters is the same as in *main*).
2. The context of the module matches the context of the rule (that is, neighboring modules have the same letters and the same number of parameters as the corresponding left/right modules of the rule's predecessor). Keyword "ignore" in the description file specifies which modules we should ignore while searching for context of the current module. In the example above, if at certain step the module A were part of the string B(4)F(1,1)+A(2,3), we would say that it's left neighbor is B, not F or +.
3. One of the conditions of rule holds for this module. To check the condition, we substitute variables in the rules with the values of corresponding parameters from the string module and its context, and evaluate the resulting logical expression.

If these three conditions hold, we apply the appropriate production of the rule (from the list of the rule productions) which corresponds to the condition which was evaluated to “true”. After the application of some production, the modules will be replaced by a list of successive modules with their parameters initialized from the parameter values of this module and its context. For example, after we apply the first rule of the above file to module  $F(1,1)$  we generate a module  $A(3,0)$  replacing it in the updated string.

If no rule applies to some module, we add the module itself to the updated string.

After application of rules to all modules, we glue lists of successive modules for each module from the initial string, and get a new state of the L-system. The whole process of rewriting is repeated `num_iterations` times.

The final string is the one our “turtle” is going to parse to generate a geometry-representing tree for it. Let’s say, after applying rules from our l-file we got the following string:

$XB(2)A(3,0)F(10,1)[+F(5,2)]^{45}@(.0.3)\#(1)$

2D turtle interpretation described in introduction is extended to 3D in a natural way: besides  $(x,y,z)$  coordinates describing turtle position, we have three vectors  $h,u,l$  (heading, up, left) representing the orientation of the turtle in space. The following symbols control turtle orientation now:

- $+(\delta)$  Turn left by angle  $\delta$ , using rotation around current u-axis.
- $-(\delta)$  Turn right by angle  $\delta$ , using rotation around current u-axis.
- $\&(\delta)$  Pitch down by angle  $\delta$ , using rotation around current l-axis.
- $\wedge(\delta)$  Pitch up by angle  $\delta$ , using rotation around current l-axis.
- $\swarrow(\delta)$  Roll left by angle  $\delta$ , using rotation around current h-axis.
- $\searrow(\delta)$  Roll right by angle  $\delta$ , using rotation around current h-axis.

Several other symbols are used to control the development of a plant. Below are some of them:

- $@(s)$  Scale by  $s$ .
- $\#(num)$  Use a pre-define mesh number “num” for this node.
- $\$$  Rotate around  $h$  so that vector  $l$  becomes horizontal
- $!(x)$  Decrease the radius of the cylinders representing geometry by  $x$

As the turtle parses the string, it recursively creates a tree (its branches correspond to the branching structures in the string) (Java3D scenegraph), filling out transformations for each module. Java3D scenegraph is displayed for each evaluation time step. Example of a tree for some simple L-production rule is given in Figure 4.

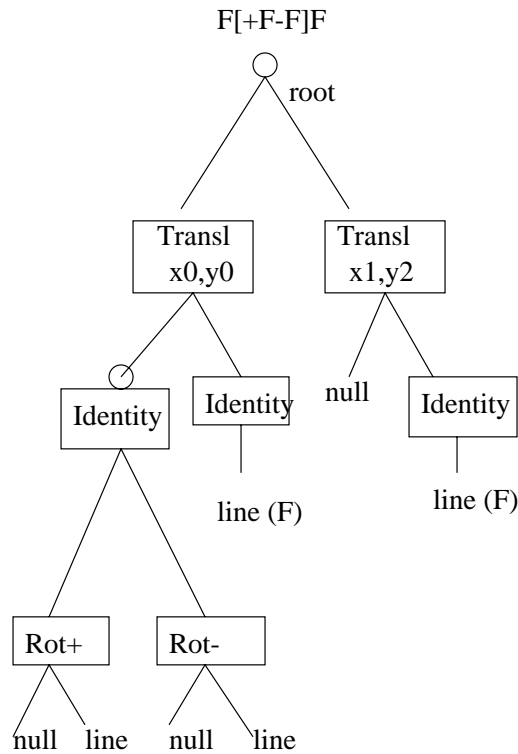


Fig. 5

## 4.2 Dynamic plants

The process of plant growth is divided in two parts:

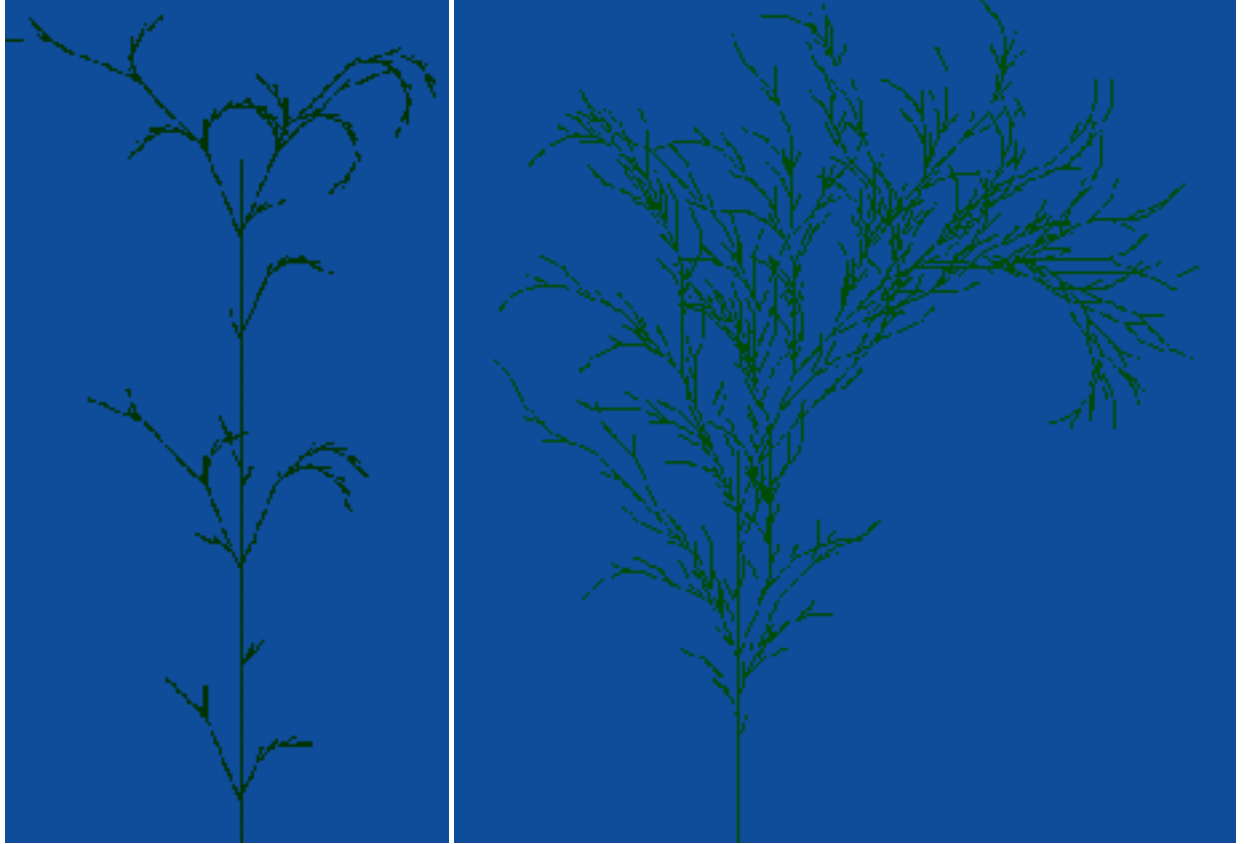
- Continuous growth (like increasing the apex length),
- Discrete even occurrence (like producing two new branches out of the stem).

Parameters of the modules are changed according to some functions till the point when parameter value reaches some value, and then some appropriate discrete even occurs. In the current implementation, there is no global time and parameters depend on the previous values linearly (see the example of l-system describing rose flower growth). Length of internodes, angles of branches and sizes of leaves and the flower are all changing continuously, depending on previous values of the corresponding modules and their neighbors.

Ongoing work is to introduce “global” time to the system and update all parameters according to some hard-coded predefined functions which may de-

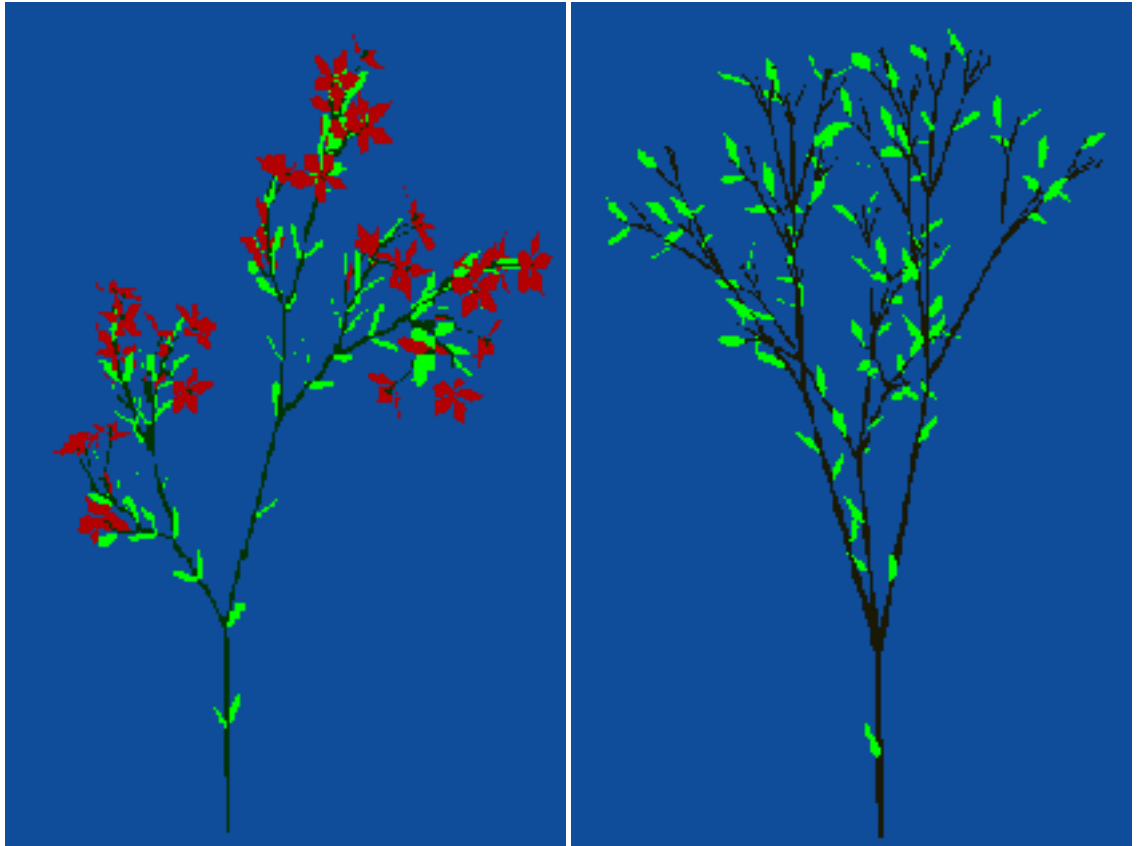
pend on time directly. The problem with this approach is that it becomes UN-intuitive to create l-files for the animation.

## 5 Results: Static plants



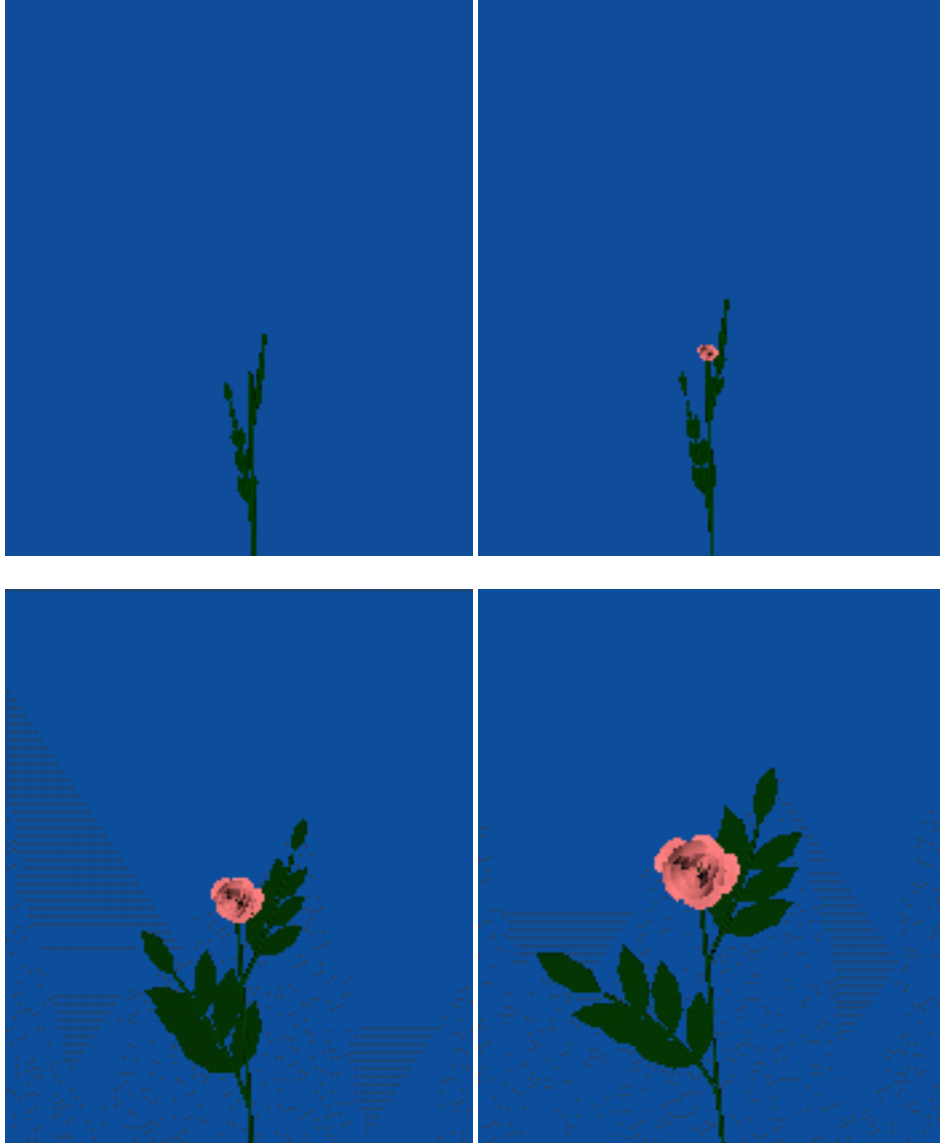
Plants generated with context-sensitive non-parametric L-systems

<i>First plant</i>	<i>Second plant</i>
num_iterations: 24	num_iterations: 25
angle: 25.75	angle: 22.5
ignore: +-F	ignore: +-F
axiom: FKFBFB	axiom: FBFBFB
rules:	rules:
K<K>K=B	K<K>K=K
K<K>B=K[+FBFB]	K<K>B=B[+FBFB]
K<B>K=K	K<B>K=B
K<B>B=BFB	K<B>B=B
B<K>K=B	B<K>K=K
B<K>B=B[+FBFB]	B<K>B=BFB
B<B>K=B	B<B>K=K
B<B>B=K	B<B>B=K
+=-, -=+	+=-, -=+



The pictures of a flower plant and a young tree.

## 6 Animation Results





Above are several frames of a growing rose animation. The following L-grammar was used to generate this sequence:

```

lssystem: 0
num_iterations: 61
angle: 25
step: 1
radius: 0.05
ignore: +~F
axiom: +(0.5)C(1,3)
rules:
C(x,n):
if x<21&n>0
produce B(0.2)C(x+1,n-1)
if x<21&n=0
produce B(0.2)[+(3)A(1,3)]D(x+1,8)
if x=21
produce @(0.2)#(2)
endrule

D(x,n):
if x<21&n>0
produce B(0.2)D(x+1,n-1)
if x<21&n=0
produce B(0.2)[~(8)A(1,3)]C(x+1,8)
endrule

A(x,n):

```



## 7 References

1. Przemyslaw Prusinkiewicz, Mark S. Hammel, Eric Mjolsness, *Animation of Plant Development*. SIG-93.
2. Przemyslaw Prusinkiewicz, Aristid Lindenmayer, *The Algorithmic Beauty of Plants*. Springer Verlag 1990.
3. Joanna L. Power, A. J. Bernhein Brush, Przemyslaw Prusinkiewicz, David Salesin. *Interactive Arrangement of Botanical L-System Models*. 1999 Symposium on Interactive 3D Graphics.
4. H. Abelson, A. A. diSessa, *Turtle geometry*. MIT Press, Cambridge 1982.
5. Przemyslaw Prusinkiewicz, Mark James, and Radomir Mech, *Synthetic Topiary*. SIG-94.