L-Systems: A Simulation of Plant Development

Daphne Chiou, Kevin Shih, Calvin Chen, Wesley Tzou
University of California, Los Angeles
Artificial Life for Computer Graphics and Vision

Abstract

L-systems, or Lindenmayer systems, are often used to create realistic looking artificial plants. An L-system contains a set of alphabets that can be used to generate strings and geometric structures , a set of production rules that expand each alphabet into longer string of alphabets, and an initial axiom from which to begin the process.

In this project we aim to implement a parametric L-systems from scratch that is capable of generating realistic plant structures in 3D space and visualize them on 2D plane.

Keywords: L-system, plant, artificial life.

1 Introduction

In 1968, Aristid Lindenmayer introduced a parallel rewriting system, which is known as L-systems. Originally L-systems were introduced to model the development of simple *multicellular* organisms (for example, algae), including division, growth and death of an individual cell. The range of L-systems applications has subsequently been extended to high-level plants and complex branching structures.

A L-system contains a set of *modules* which are denoted by alphabets, a set of production rules, and an axiom. The modules represent the geometric structures of plants. It could be an inter node, a apex, a flower or a branch; Production rules are used to expand each module into longer sequence of modules; An initial axiom is where the growing process begins.

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

In this context, we will focus on generating plants and complex branching structures with the following systems:

- 0L-Systems
- Parametric 0L-Systems

2 L-System

2.1 0L-Systems

0L-Systems are the simplest class of L-Systems. The following summarizes fundamental definition and notion of 0L-Systems:

Def 1.

0L-system is an ordered triplet $G = \langle V, \omega, P \rangle$,
V are the alphabets,
V^+ is the set of all non-empty words over V,
$\omega \in V^+$ is the axiom,
V^* is the set of all words over V,
$P \subset V * V^*$ is a finite set of productions,
(a, χ) is a production, denoted by $a \rightarrow \chi$.

Def 2.

Let $G =< V, \omega, P >$ be a 0L-system. Suppose $\mu = a_1, ..., a_m$ is an arbitrary word over V. We say that the word $v = \chi_1, ..., \chi_m$ is generated by μ and write $\mu \implies v$ iff $a_i \rightarrow \chi_i$ for all $i = 1, ..., m$. A word v is generated by G in a derivation of length n if there exists a sequence of words $\mu_0, \mu_1, ..., \mu_n$ s.t $\mu_0 = \omega, \mu_n = v$ and $\mu_0 \implies \mu_1 \implies ... \implies \mu_n$.

2.2 Parametric L-Systems

Parametric L-systems operate on parametric words, which are strings of modules consisting of alphabets with associated parameters. A module with alphabets $A \in V$ and parameters $a_1, a_2, ..., a_n \in R$ is denoted by $A\,(a_1, a_2, ..., a_n)$ Parametric L-systems are defined as follows:

Def.

Parametric 0L-system is an ordered quadruplet
$G = \langle V, \Sigma, \omega, P\rangle$,
V are the alphabets,
Σ is the set of formal parameters ,
ω is the axiom,
P is a finite set of productions.

2.3 Turtle Interpretation

The operation of a L-system can be viewed as a set of rules that transform an initial state (axiom) into a sequence of operations recursively. After the transformation is completed, the generated operations are a sequence of commend which can be used to control the *turtle* to draw a picture. The configuration of the turtle includes *position*, *orientation* and *line width*.

The following list specifies the basic set of symbols interpreted by the turtle.

$!(w)$	Set line width to w.
$F(s)$	Move forward a step of length s and draw a line segment from the original to the new position of the turtle.
$-(\theta)$	Turn right by angle θ along z-axis.
$+(\theta)$	Turn left by angle θ along z-axis.
$\backslash(\theta)$	Turn right by angle θ along y-axis.
$/(\theta)$	Turn left by angle θ along y-axis.
[	Push the current state of the turtle onto a pushdown stack.
]	Pop a state from the stack and replace the current state of the turtle. The position and orientation of the turtle are changed but no line will be drawn.

3 Implementation

Among all types of L-Systems introduced above, we choose to implement parametric L-Systems not only because it is more challenging, but also because it is capable of generating more complex and realistic tree structures due to the flexibility that the parameters provide. To take advantage of the Turtle package, which is commonly used to visualize the L-systems, the parametric L-systems is implemented in Python. The program can be decomposed into several fragments, including the parser, the interpreter, the execution, and the visualization.

3.1 Parser

The Parser takes two types of string input, the axiom and the production. The axiom is the initial state of the system and is parsed into expression, while the production is parsed into 4 components, the production name, the corresponding parameters, the condition, and the expression. In the following example, the axiom is denoted by ω, in which consists of a function name A, and the corresponding production of A is denoted by p_1.

$$\begin{aligned}
\omega &: A(100, \omega_0)\\
p_1 &: A(s, w) : s >= min \rightarrow !(w)F(s)\\
&\quad [+(\alpha_1)/(\psi_1)A(s * r_1, w * q \wedge e)]\\
&\quad [+(\alpha_2)/(\psi_2)A(s * r_2, w * (1 - q) \wedge e)]
\end{aligned}$$

In this example, A is the function name and s, w are the corresponding parameters. $s >= min$ is the condition and the following function-like sequence is the expression of the production. Note that only when the condition is true will the expression be executed.

3.2 Interpreter

In this section, the output of the parser is interpreted and stored structurally. The 4 components of the production are interpreted into different formats respectively. The function name and its parameter are split by () and ,. The rest of components are relatively problematic. The condition is further parsed into a list of elements, e.g. [‘s‘, ‘ $>=$ ‘, ‘0‘] and the expression is parsed in to a list of actions represented by pairs. A Pair contains a Turtle command and its corresponding parameters, e.g. (‘!‘, ‘ω‘). To simplify the problem we view [and] as function and take $emptystring$ as its parameter. So far all elements are still in string format. To interpret the axiom, we can simply reuse the method for interpreting expression.

3.3 Execution

The function details are stored in a dictionary that takes function name as key and a tuple of (function parameters, condition, and expression) as value. To unfold the axiom ω into n iterations, in each iteration, the customized functions(other than the built-in functions in Turtle package) are replaced with its expression by looking up the dictionary. Note that the replacing process requires parameter-passing, where the calculation of the parameters according to the expression has to be done. This is rather hard to implement from scratch as all numerical information is in string format. Alternatively, a Python build-in function $eval()$ is used to interpret a string of equation into its value. For example, given '$2 * 3 + 1$' it will return 7.

Finally, the L-Systems are ready to visualize with Turtle.

3.4 Visualization

Although Turtle package supports turtle interpretations in 2-dimensional space, it does not allow L-Systems to grow trees in 3-dimensional space. To achieve the goal, we decided to manually build the Turtle system in 3D space, and then project it onto a 2D-plane. In our system, the turtle is represented by its *state* consisting of turtle *position* and *orientation* in Cartesian coordinate system. The position is defined by a vector $\vec{P}$ and the orientation is represented by three vectors: $\vec{H}$, $\vec{L}$, and $\vec{U}$, indicating the turtle's *heading* and the direction to the *left* and *up*. These vectors are in unit length and are perpendicular to each other. Rotations of the turtle are expressed by the equation:

$$\begin{bmatrix}\vec{H'} & \vec{L'} & \vec{U'}\end{bmatrix} = \begin{bmatrix}\vec{H} & \vec{L} & \vec{U}\end{bmatrix}\mathbf{R}$$

where $\mathbf{R}$ is a 3*3 rotation matrix. Specifically, matrices rotating by angle α about vectors $\vec{H}$, $\vec{L}$, and $\vec{U}$ are shown below:

$$\mathbf{R_U}(\alpha) = \begin{bmatrix} cos(\alpha) & sin(\alpha) & 0 \\ -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_L}(\alpha) = \begin{bmatrix} cos(\alpha) & 0 & -sin(\alpha) \\ 0 & 1 & 0 \\ sin(\alpha) & 0 & cos(\alpha) \end{bmatrix}$$

$$\mathbf{R_H}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix}$$

To project the turtle onto 2D-plane, there are two turtle actions needed to be focused on, which are $rotation$ and $moving forward$. When rotating the turtle, after the orientation vectors are transformed, $\vec{H}$ should be projected on the 2D-plane (xy-plane):

$$\vec{H} \rightarrow \vec{H'}$$

To calculate the angle (heading direction) for the Turtle Graphics, use $\mathbf{atan}$ of $\vec{H'}$:

$$\alpha = atan(\vec{H'}[1]/\vec{H'}[0]) * 180/\pi$$

According to the cycle of *tan*, if $\vec{H'}$ is in the second or third quadrant of xy-plane, an additional angle π is added:

$$\alpha' = \alpha + \pi$$

When drawing on the plane, the length would also be projected:

$$L' = L * cos(\vec{H}[2])$$

By doing the modification above, the turtle draws the trees perfectly on a 2D-plane.

4 Experiments

4.1 0L-Systems

For 0L-Systems, we build a model that allow user to determine alphabet, axiom, rules and interpretation. The inputs of our 0L-System model are as following:

V	alphabets
ω	axiom
P	rules of production
i	interpretation for each alphabet
n	time of interpretation

Here are several examples of using this 0L-System model:

Example 1.

$V : (F, +, -)$
$\omega : F$
$P : F \rightarrow F[+F]F[-F]F$
$i : (F$: forward 5, $+$: left 20, $-$: right 20)
$n : 5$

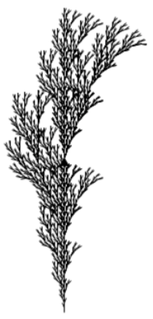


Table 1: Parameters used

Figure	r_1	r_2	α_1	α_2	ψ_1	ψ_2	ω_0	q	e	min	n
a	.75	.77	35	-35	0	0	30	.50	.40	0.0	10
b	.65	.71	27	-68	0	0	20	.53	.50	1.7	12
c	.50	.85	25	-15	180	0	20	.45	.50	0.5	9
d	.60	.85	25	-15	180	180	20	.45	.50	0.0	10
e	.58	.83	30	15	0	180	20	.40	.50	1.0	11
f	.92	.37	0	60	180	0	2	.50	.00	0.5	15
g	.80	.80	30	-30	137	137	30	.50	.50	0.0	10
h	.95	.75	5	-30	-90	90	40	.60	.45	25.0	12
i	.95	.95	-5	30	137	137	5	.40	.00	5.0	12

Example 2.

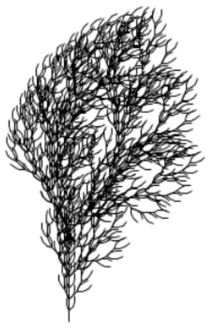
$V : (F, +, -)$

$\omega : F$

$P : F \rightarrow FF - [-F + F + F] + [+F - F - F]$

$i :$ (F : forward 5, $+$: left 22.5 , $-$: right 22.5)

$n : 4$



4.2 Parametric L-Systems

We re-implement the result from (Prusinkiewicz et al., 1996). The axiom ω and the production used in the experiments p_1 are described as following:

$$\omega : A(100, \omega_0)$$

$$p_1 : A(s, w) : s >= min \rightarrow !(w)F(s)$$
$$[+(\alpha_1)/(\psi_1)A(s * r_1, w * q \wedge e)]$$
$$[+(\alpha_2)/(\psi_2)A(s * r_2, w * (1 - q) \wedge e)]$$

To generate different tree structures, our Parametric L-System take r_1, r_2, α_1, α_2, ψ_1, ψ_2, ω_0, q, e, min, n as parameters. Each of them describe different character of a tree structure. The α and ψ denote the orientation of the apices. The r value determines the decrease in internode length from the bottom to the top of the structure. The ω_0, q, and e denote the width of branches. On the other hand, the condition, $s >= min$, prevents the system to grow branch with width less than min. The parameters used are shown in Table 1 and the visualizations are shown in Table 2 on the next page.

5 Conclusion

In this project, we successfully implement a 0L-System and a Parametric 0L-Systems from scratch. The most challenging parts of our re-implementation are the parser and the interpreter in the Parametric 0L-System, and the rotation of the Turtle visualization. The strength of the parser and the interpreter directly affect the generalness of the system. By defining the data structure carefully and parsing the information into structural data, we manage to interpret the input correctly. In addition, as mentioned in the progress report, we failed to rotate Turtle in 3D space with built-in Turtle functions in the very beginning. We found the built-in functions no longer meet our needs. Therefore, we manually implement the simulation of Turtle rotating in 3D space. Although the structures might be slightly different from the results in the original paper due to computational error and different projection perspective, the visualization results are still very satisfying.

References

Prusinkiewicz, P. 1986. *Graphical applications of L-systems. In Proceedings of graphics interface*, (Vol. 86, No. 86, pp. 247-253).

Prusinkiewicz, P., Hammel, M., Mech, R., & Hanan, J. 1995. *The artificial life of plants. Artificial life for graphics, animation, and virtual reality,*, 7, 1-1.

Prusinkiewicz, P., Hammel, M., Hanan, J., & Mech, R. 1996. *L-systems: from the theory to visual models of plants. In Proceedings of the 2nd CSIRO Symposium on Computational Challenges in Life Sciences*, (Vol. 3, pp. 1-32). CSIRO

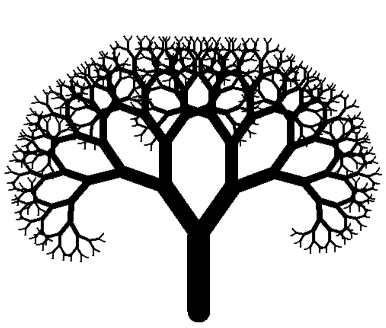


Figure 1: a

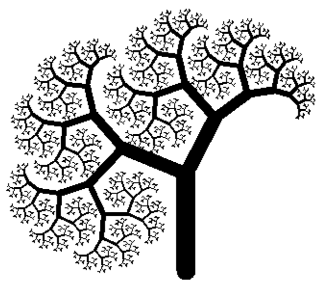


Figure 2: b

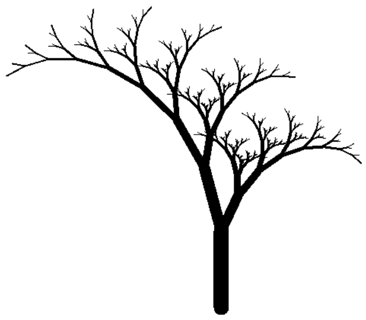


Figure 3: c

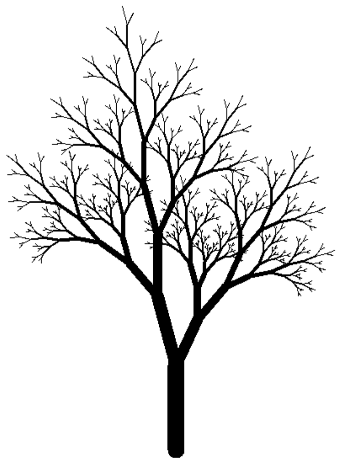


Figure 4: d

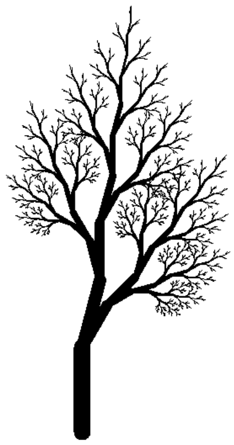


Figure 5: e

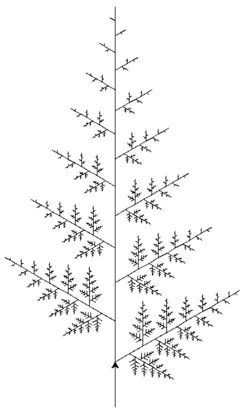


Figure 6: f

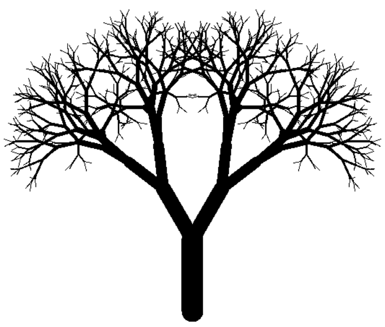


Figure 7: g

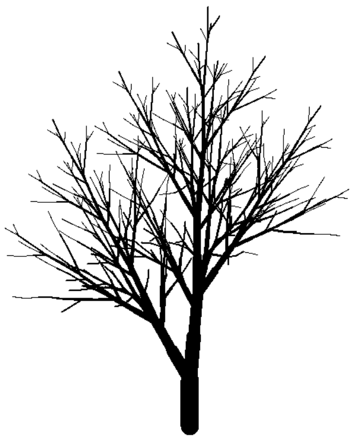


Figure 8: h

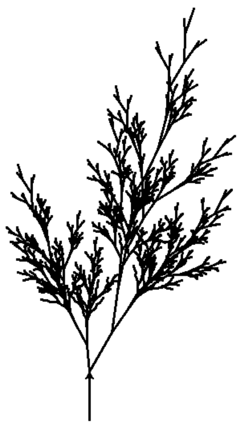


Figure 9: i

Table 2: Results of Parametric L-Systems