Alternative Programming for Visualization and Animation

Shu-Wei Hsu Chuan Wang
shwhsu, vvang@ucdavis.edu

June 13, 2013

1 Abstract

Lindenmayer system is a formal grammar (a set of rules and symbols) most famously used to model the growth processes of plant development, which enables modeling the morphology of a variety of organisms. This report describes a interactive system that we build on top of L-system for creating computer graphics and animations. Leveraging the fractal nature of natural objects, we employed an alternative programming mechanism using recursive drawing. Using our system users can easily compile complicated graphical structurer using relatively simple grammar, yet the web-based interface provides interactive programing experience with real-time visual feedbacks, outweighs static graphical results generated by traditional programming models that require compilation before execution. We demonstrated the detailed design of the system as well as some preliminary results that show the effectiveness and efficiency of the system.

2 Introduction

Creating computer graphics and animation for new programmer can be a tough work. As lots of source-code need to be programmed and generated by using graphic library, e.g. OpenGL, or 3D graphics software such as Maya, the implementation is usually knowledge demanding and with long development process. Moreover, programmer has to start from building objects, color blending, and coordinate system transformation instead of connecting objects and applying animations among them regarding different time and space.

The goal of this report is to introduce a dynamic interactive graphics-drawing programming interface with which simple grammars can create objects and apply animation by translating the strings to geometric structures and animations.

The central concept of L-Systems is that of rewriting. It consists of a finite set of letters (collectively called the alphabet). Letters are arranged in arbitrary length sequences to form strings. Each letter is associated with a rewriting rule. The string length can grow quickly after only a few rewrites. Therefore, the morphogenesis nature of L-system guarantees an efficient way of drawing live creatures such as trees and other plants. The structure of the strings manipulates the structure of the plant and consequently defines the shape of drawing.

This dynamic drawing interface allows real-time displaying of the parsing results. For example, each of the visual environments allows the user to adjust parameters with sliders, and see the results of these changes in a real-time feedback. And the interface allows real-time changing in parameters that can manipulate the length of the drawing element, the degree of the branch towards the trunk, part of the element size and so on. In order to let the drawing more natural a random parameter is included to give randomized disturbances.

3 Related work

One recent work by Toby Schachman [1] seeks to broaden the view of what programming is, who programs, and how programming fits in to larger systems. It makes suggestions for the design of these new programming interfaces. It presents as

a case study and demonstration Recursive Drawing, which is a reimplementation of the textual programming language Context Free as a graphical, directly manipulable interface. Instead of a compiler or interpreter, Recursive Drawings programming interface is modeled as a constraint solver and consequently allows the programmer to modify the programs source code by manipulating the programs output. There work offers conveniences for new programmers from unlikely backgrounds such as the arts, who bring with them ways of working which are incompatible with mainstream programming practices.

Another research paper by Jon McCormack[2] describes an interactive modelling system for computer graphics in which the user is able to evolve grammatical rules and surface equations. Starting from any initial L-System grammar the evolution proceeds via repeated random mutation and user selection. Sub-classes of the mutation process depend on the context of the current symbol or rule being mutated and include mutation of: parametric equations and expressions, growth functions, rules and productions. As the grammar allows importation of parametric surfaces, these surfaces can be mutated and selected as well. The mutated rules are then interpreted to create a three-dimensional, time-dependent model composed of parametric and polygonal geometry. L-System evolution allows with minimal knowledge of L-Systems to create complex, lifelike images and animations that would be difficult and far more time-consuming to achieve by writing rules and equations explicitly. It demonstrated that evolution of iL-Systems provides a powerful method for creating complex computer graphics and animations.

Letters of the L-system alphabet are represented graphically as shorter or longer rectangles with rounded corners.The generated structures are one-dimensional chains of rectangles, reflecting the sequence of symbols in the corresponding strings. In order to model higher plants, a more sophisticated graphical inmethods terpretation of L-systems is needed. The first results in this direction were published by Frijters and Lindenmayer [3], and Hogeweg and Hesper [4]. In both cases, L-systems were used primarily to determine the branching topology of the modeled plants. The geometric aspects, such as the lengths of line segments and the angle values, were added in a post-processing phase.

4 Approach

4.1 Interpretation of strings in L-system

An L-system is a formal grammar (a set of rules and symbols) most famously used to model the growth processes of plant development, which enables modeling the morphology of a variety of organisms. L-systems can also be used to generate self-similar fractals such as iterated function systems. As the grammar allows importation of parametric surfaces, these surfaces can be mutated and selected as well. The mutated rules are then interpreted to create a 3D, time-dependent model composed of parametric and polygonal geometry.

The basic idea of interpretation is like turtle in Python. A state of the turtle is defined as a triplet (x, y, α) [5], where (x, y) represents the turtle's position, and the α is a angle that represents the heading direction of the turtle. Given a step size d and the angle increment δ. The turtle can respond to commands represented by the following symbols:

F	Move forward a step of length d. The state of the turtle changes to $(x\prime, y\prime, \alpha)$, where $x\prime = x + dcos\alpha$ and $y\prime = y + dsin\alpha$. A line segment between points (x, y) and $(x\prime, y\prime)$ is drawn.
+	Turn left by angle δ. The next state of the turtle is $(x, y, \alpha + \delta)$. The positive orientation of angles is counterclockwise.
−	Turn right by angle δ. The next state of the turtle is $(x, y, \alpha - \delta)$.
[	push a command
]	pop a command

Given a string ν, the initial state of the tur-

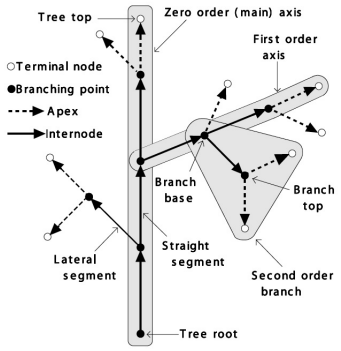


Figure 1: Graphical model

tle $(x_0, y_{0,0})$ and fixed Interpretation parameters d and δ, the turtle interpretation of ν is the figure (set of lines) drawn by the turtle in response to the string ν.

According to the rules presented, the head of the drawing tool interprets a character string as a sequence of line segments. Depending on the segment lengths and the angles between them, the resulting line is self-intersecting or not, can be more or less convoluted, and may have some segments drawn many times and others made invisible, but it always remains just a single line. However, the plant kingdom is dominated by branching structures; thus a mathematical description of tree-like shapes and methods for generating them are needed for modeling purposes. The graphical model using L-systems is showed in graph 1.

The following graph 2 shows a implemented result for some examples:

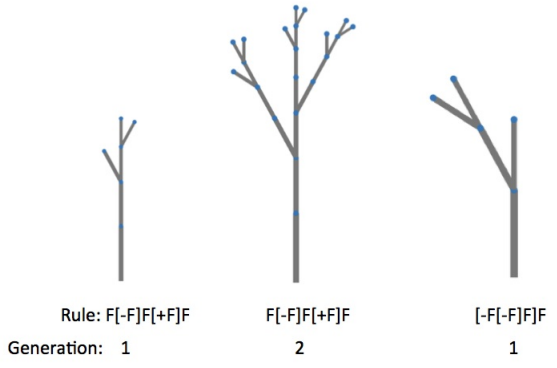


Figure 2: Examples of string interpretation

The recursive nature of the L-system rules leads to self-similarity and thereby fractal-like forms. Plant models and natural-looking organic forms are easy to define, as by increasing the recursion level the form slowly "grows" and becomes more complex.

4.2 Environment Configuration

This web-based L-system dynamical visualization application is implemented using D3.js. It is known as Data-Driven Document, a JavaScript library developed by Ph.D. student Mike Bostock, Prof. Jeff Heer and M.S. student Vadim Ogievetsky of the Stanford University's Stanford Visualization Group, which uses widely implemented SVG, JavaScript, and CSS standards. It is a famous tool for data visualization embedded within HTML webpage, using javascript functions to create, style, and transit SVG objects. Scalable Vector Graphics, abbreviated as SVG, is a XML based vector image format for two-dimensional graphics standard developed by W3C. The vector image is composed of a fixed set of shapes instead of a fixed set of dots. Moreover, it is easy to be animated and interacted. Therefore, t is pretty ideal to apply D3.js library for code visualization of L-system.

Furthermore, since it is our goal to create a real time visualization corresponding to the dynamical modification of simple L-system program, D3.js can also handle the changes of data and vector graphs efficiently. As a data-driven visualization library, it changes the vector graphs only when needed. The braches and relative nodes (as leaves) are defined by a dynamical created array. If there is a new change of the L-system program, the array is regenerated. Afterwards, D3.js library compare the differences between the array and the vector graphes. It sets the new must-draw branches into the list of enter(), and the must-dump branches into the list of exit(). The objects inside the list of exit() are removed by exit().remove(). The backgroun svg object as the canvas is upated and rendered with new existing data in the array with interpolation (transition) between the previus and new state.

5 Implementation and Results

The interface is built on the top of JavaScript based on the d3.js model. The rules of L-system program is pre-interpreted before generating branches of each generations in our L-system tree. The branches inside the array are defined by a struct:

$$
\begin{aligned}
&b = \{\\
&\quad i : id,\\
&\quad x : x - axis\ coordinate\ of\ branch's\ start\ point,\\
&\quad y : y - axis\ coordinate\ of\ branch's\ start\ point,\\
&\quad a : angle\ of\ bifurcation,\\
&\quad l : length\ of\ branch,\\
&\quad d : depth\ of\ branch,\\
&\quad parent : parent\ of\ this\ branch\\
&\};
\end{aligned}
$$

The *id* is defined by the index of branches array, since we push the new branch into the array right after it is created. The x and y defines the start point of this branch, which is derived from the end point of its parent branch. a defines the angle of bifurcation of this branch to its parent; l define the length of branch; and *parent* define the if of the parent of this branch. Therefore, the new branch is derived from its parent as shown in the following:

$$
\begin{aligned}
&b = \{\\
&\quad i : branches.length,\\
&\quad x : parent.end.x,\\
&\quad y : parent.end.y,\\
&\quad a : parent.a + newAngle + aRandom,\\
&\quad l : parent.l * newlength,\\
&\quad d : parent.d + 1,\\
&\quad parent : parent.i\\
&\};
\end{aligned}
$$

newAngle defines the direction of new bifurcation. It is a negative value if it turns left, and positive for turning right. aRandom randomized the angle of branches to make it more realistic as live trees. *newLength* defines the length ratio comparing to its parent. The depth is incremented by 1 when assigning a new child connected to its own parent. The depths are shown in Figure 5d . Depths can also be used to decide the thickness of branches to make it reasonable. That is, thick to thin gradually from the root to the top of the tree. The parent passes the generation and stack for push and pop to its child. If the end of this generation is reached, the generation number is incremented by 1. The pre-interpreted rules will be executed again to create the branches of next generation till reaching the max limit number of generations.

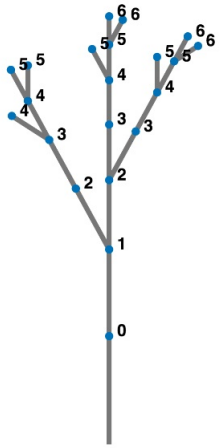


Figure 3: Depths of each branch

Figure 4 shows its interface. On the top right window, user can edit command, and the parsed command will dynamically draw the result on the large window on the left. In order to make editing of parameters convenient, slide bars and a button are added to the bottom of the editing window.

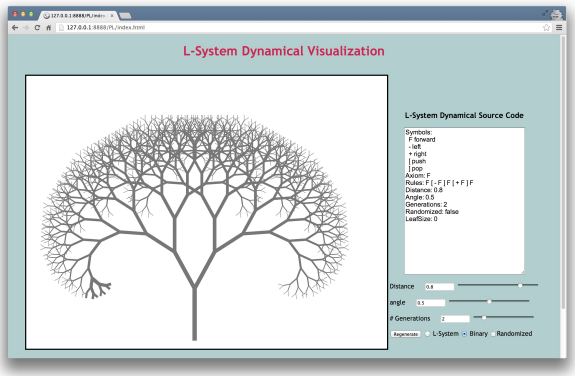
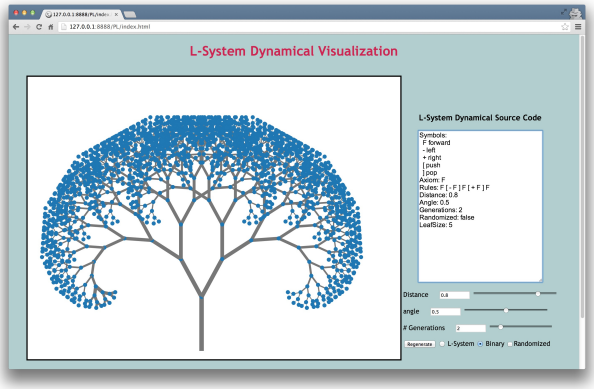
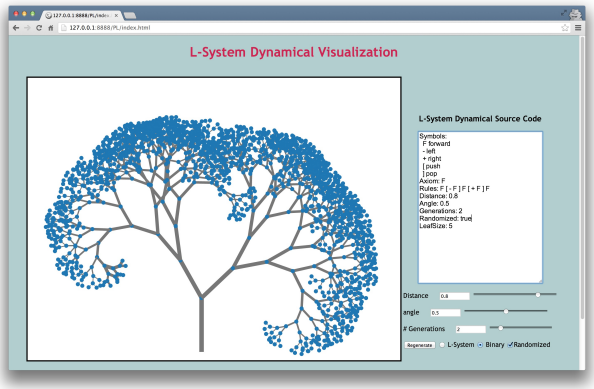


Figure 4: Interface of our system

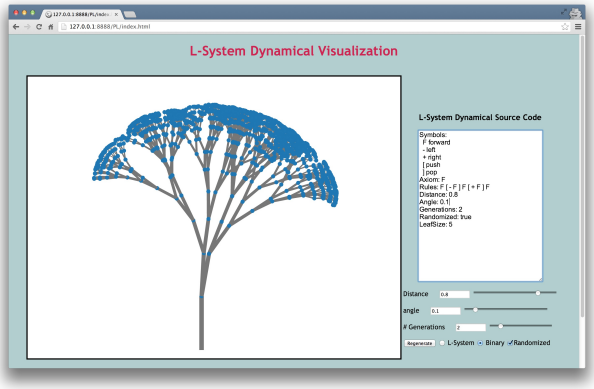
The following figures shows the result of drawing binary trees. Figure 5b is the randomized tree and it is more natural compared to the binary tree generated in Figure 5a. Figure 5c and Figure 5d shows the tree changes under the degree parameter and distance parameter respectively. We can see that a slight change in parameter will cause an apparent changing in the result as the changes are accumulated by the recursion in the program.



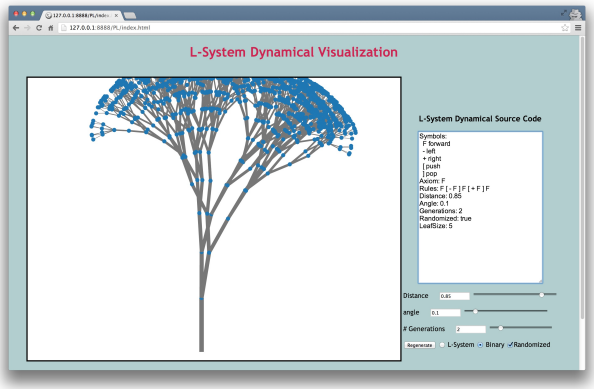
(a) A Binary Tree



(b) Randomized: true



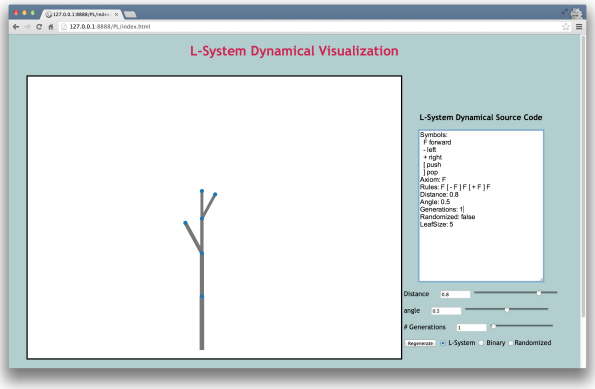
(c) Degree: $0.5(* \ 60°)$



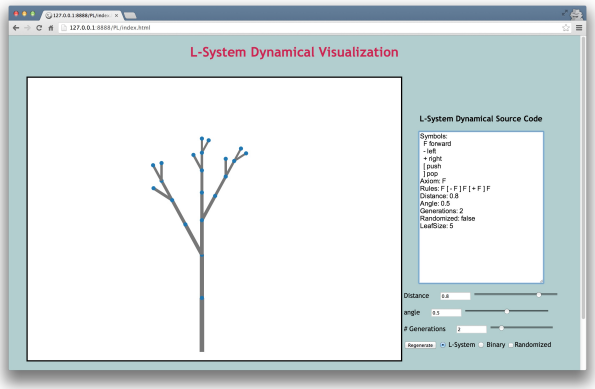
(d) Distance Changing

Figure 5: Parameters

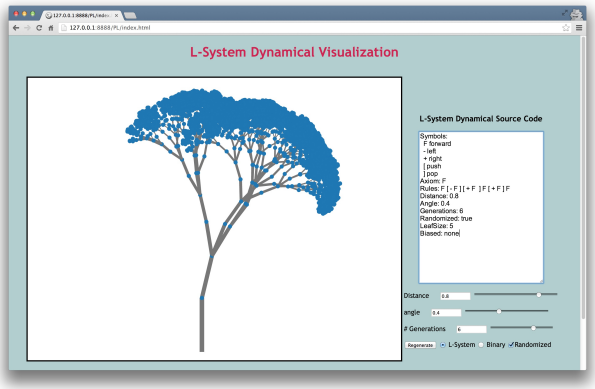
For L-system based tree drawings, the structure of the tree can be determined by different strings that act as the rules. In the following example, Figure 6a and 6b demonstrates the drawing done by string $F[-F]F[+F]F$ with depth one and two respectively. And Figure randLsystem6 shows a bigger tree that draw by depth of six.



(a) Lsystem with one iteration



(b) Lsystem with two iterations



(c) Lsystem with six iterations

Figure 6: Trees on L-system

Figure 7 shows the result of our sys-

tem. Visit `http://vis.cs.ucdavis.edu/~hsus/PL/index.html` to create your own drawings!

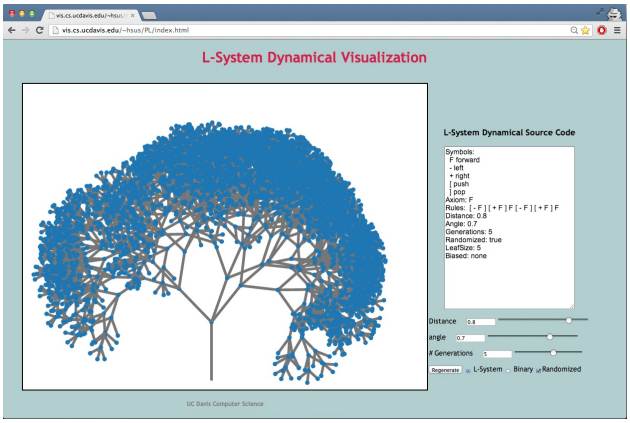


Figure 7: Implementation Result

6 Conclusion

In this project, we build up a graphics creating system. Real-time feedback is given by the dynamic drawing interface, and the formal grammar of L-system enables an efficient way of generating an amount of elements in the graph with short code string.

In the future, animation can be appended to the system. For example, slit changes in leaf size and color level could generate the animation of breeze and storm can be animated as a blending process from a tree to the biased tree. The interpolation of frames needs to be calculated. Also, currently, the changing in the depth of the tree are draw by the framework autometically. If blending between the graphs of from low depth to high depth, the animation of tree growing can be created.

References

[1] Toby Schachman: Alternative Programming Interface for Alternative Programmers, Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software, New York University, Interactive Telecommunications Program, 1-10, 2012.

[2] J. McCormack: Interactive Evolution of L-System Grammars for Computer Graphics Modelling, in Complex Systems: From Biology to Computation, David Green and Terry Bossomaier (eds.), ISO Press, Amsterdam, pp. 118-130, 1993.

[3] D. Frijters, A .Lindenmayer: A Model for the Growth and Flowering of Aster Novae-Angliae on the Basis of Table $< 1, 0 >$ L-Systems. L Systems 1974: 24-52.

[4] P. Hogeweg , B. Hesper: A model study on biomorphological description, Pattern Recognition, 1974.6:165-179.

[5] P. Prusinkiewicz, A Lindenmayer: The algorithmic beauty of plants, Springer-Verlag New York, Inc. 1996.