Using L-Systems to Procedurally Generate Diverse Plants in Virtual Reality

Abstract

This project focuses on the generation of plants using Lindenmayer Systems (L-Systems) to create strings that define plant structures. L-Systems used to generate plants typically have human-defined rules, which result in plants that appear highly realistic but have little or no variation. In contrast, this project produces diverse plants using one system without human definition of L-System rules. To do this, I implemented methods for generating deterministic three-dimensional plant meshes from strings, as well as created a separate context-free grammar for generating the L-System rules themselves.

The result of this project is a three-dimensional virtual reality app that runs in iOS 9 using the Google Cardboard headset. While playing the game, users can navigate the environment and spawn new plants by planting seeds in the terrain. As time passes, plants grow larger and more complex, and are saved in between playthroughs so the user can return to their virtual garden at any time. Due to the machine-generated L-System rules and randomized attributes of the plants upon their creation, every plant the user spawns has a completely unique physical appearance and growth rate.

Introduction

Aristid Lindenmayer initially proposed L-Systems for modelling cellular growth of simple organisms like algae [1], but L-Systems have since been used for graphical modelling of higher plants [2]. While L-Systems have generated structures that range from small flowering shrubs and sunflower heads to massive branching tree-like structures, L-System rules for these structures are decided by and refined by humans until the plants generated from these rules most closely resemble existing plants. In some cases, rules are taken directly from analysis of existing plants, generating a structure that looks almost exactly like the particular plant being analyzed [3].

In these systems, any plant generated from these pre-decided rules will invariably turn out the same, creating highly realistic but non-diverse plants that require human intervention to decide on the correct set of rules. In contrast, this project focuses on diverse generation of plants such that the user can plant hundreds of plants in their virtual garden with no two plants being alike, without any pool of pre-set L-System rules that are assigned to plants upon their creation. To do this, every plant must generate a unique set of L-System rules as well as have randomized attributes that determine plant size, exact angles of growth, color, and so on.

This project has two main components. First, the generation of three-dimensional, VR-ready plant meshes from a string generated by an L-System. This component incorporates my decisions about the L-System vocabulary I use, as well as methods for procedurally generating these meshes. Second, this
project generates the L-System rules themselves using a separate context-free grammar (CFG) such that every plant has a completely unique set of rules and thus a unique structure. During the course of this project I moved away from my original plan to test the app’s effectiveness as Virtual Reality Therapy (VRT), and instead towards the generation of the L-System rules themselves.

I implemented this project using Unity, C# for Unity scripting, and Objective C for iOS code. I used the Google VR SDK for Cardboard development [4].

**Implementation Details & Process**

Initially, I wanted to design a VR game to reduce the user’s stress and anxiety, since VRT has been shown to be an effective tool for reducing these feelings in users [5]. Even after I moved away from this goal and towards the production of L-Systems, I was still interested in VR as a setting for my project given the affordability of Google Cardboard headsets and the increasing availability of VR apps on iOS and Android devices. Since I have an iPhone and a Cardboard headset, I decided to implement this project as an iOS app targeted at iOS 9 on the iPhone.

Designing this project as a game that the user can save and load adds restrictions to the generation of plants, first and foremost that plants must remain consistent between playthroughs. Although plants can change over time by becoming larger, sprouting new branches, and developing more mature leaves, the basic structure of the plant should remain the same. In addition, if the user re-opens the game before a given plant has had time to change, that plant should look exactly the same as when the user last closed the game. As a result, plant generation must be deterministic, and cannot rely on randomness which would lend itself well to the creation of unique plants. Instead, the uniqueness of plants must come from other sources during plant creation. I implemented this through the generation of L-System rules themselves using a CFG and through random generation of particular variables at plant creation, which are saved and in turn can be used to deterministically generate the string that defines the plant structure. Given this string, I then generate the vertices and triangles of the plant mesh and render it in the game at runtime.

*Plant Generation From A String*

L-System grammar defines rules that map keys - in this case, characters - into strings. To iterate the system, every character in the input string is replaced by its mapped value. For each successive iteration, the previously generated string is used as the input. As such, recursive iteration of the L-System very quickly generates long and complex strings that can be used to create fractal-like botanical structures. A full definition of each character in my L-System vocabulary is in Table 1.
Table 1: Definitions of L-System Vocabulary

<table>
<thead>
<tr>
<th>Character</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Draw a segment (F-segment) of length plant.unitLength with the radius and angle of the current turtle, and update the turtle state.</td>
</tr>
<tr>
<td>f</td>
<td>Move forward by plant.baseLeafSize at the angle of the current turtle and store the coordinates.</td>
</tr>
<tr>
<td>[</td>
<td>Push the current turtle state onto the stack.</td>
</tr>
<tr>
<td>]</td>
<td>Pop the top turtle state from the stack and set it as the current turtle state.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Increase the heading of the angle of growth by plant.increaseHeading radians.</td>
</tr>
<tr>
<td>^</td>
<td>Decrease the heading of the angle of growth by plant.decreaseHeading radians.</td>
</tr>
<tr>
<td>+</td>
<td>Increase the pitch of the angle of growth stored in the turtle state by plant.increasePitch radians.</td>
</tr>
<tr>
<td>-</td>
<td>Decrease the pitch of the angle of growth stored in the turtle state by plant.decreasePitch radians.</td>
</tr>
<tr>
<td>,</td>
<td>Increase the pitch of the angle of marking leaf boundaries by plant.increaseLeafPitch radians.</td>
</tr>
<tr>
<td>\</td>
<td>Decrease the pitch of the angle of marking leaf boundaries by plant.decreaseLeafPitch radians.</td>
</tr>
<tr>
<td>!</td>
<td>Decrease the radius stored in the turtle state by plant.decreaseRadius.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>Begin marking leaf boundaries and keep track of all vertices generated by f.</td>
</tr>
<tr>
<td>}</td>
<td>Stop marking leaf boundaries and compile all stored vertices generated by f’s into a single polygon.</td>
</tr>
</tbody>
</table>

My project uses the turtle interpretation of L-Systems described by [6]. The turtles store the angle, radius, and location of growth as well as attributes to wrap the trunk material correctly around the mesh. During the generation of a mesh from a string, special characters change the pitch and heading of the angle of growth and the radius of the stem, and these changes are stored in the turtle state so they persist as the stem continues to grow.

I started out by implementing the two-dimensional version of L-System generation using only the characters [ + - F and ]. Each F-segment was drawn as a separate hexagonal prism, the vertices of which often did not exactly meet up with the vertices of the previously drawn F-segment. I expanded the vocabulary to generate plants that grew in three-dimensions by adding the characters & and ^ into the
vocabulary and rules, but the plants generated using this method still appeared strongly two-dimensional, non-organic, and disjointed (Figure 1).

To address this, I changed my method of generating meshes to create continuous meshes where each F-segment directly connects with the next, as well as expanded the L-System vocabulary and changed the L-System rules I was using to generate strings. To create continuous meshes instead of drawing each F-segment as a separate prism, I save the coordinates of the ending face of the previous F-segment in the turtle state, and use these as the coordinates of the starting face of the next F-segment. I also added the character \(!\) which decreases the radius of the next F-segment to create branches that get continually thinner, resulting in more realistic meshes (Figure 2)

The last step I implemented for mesh generation for leaves. I used the technique described in [7] which defines a leaf as a filled two-dimensional boundary delineated by the characters \{ f, \}, \( \text{' and } \)\). Instead of drawing a new segment for each \( f \), within the bounds of \{ and \} the mesh simply stores the end coordinate of each \( f \)-segment and fills the space enclosed by these vertices after the closing \}.

With this complete set of rules, the system can generate plant meshes with leaves from strings as in Figure 3.

Finally, I implemented methods for controlling the growth of plant meshes over time. The number of iterations of the L-System is a function of how long the plant has existed. One of the plant
attributes assigned at creation time is the duration of time in between iterations of the L-System. Upon startup, the app determines how many iterations the plant should undergo, and recursively runs the L-System rules on the plant string the appropriate number of times to generate the final string. Figure 4 shows the development of a single plant from Iterations 1 to 6 (Iteration 0 has no structure).
Generation of L-System Rules and Plant Attributes

Appropriate L-System rules are necessary for generating realistic looking plants. When I first experimented with manually entering my own L-System rules, I noticed that many plants I created appeared strangely geometric and non-organic as in Figure 5.

During the course of this experimentation I decided that there are two main factors that determine how realistic a plant mesh appears. First is the choice of appropriate values for the special angle-changing characters + - & ^ , and `. Values that are too small lead to thin plants that grow almost straight up without noticeable branches, while values that are too big make the angle changes too stark and reduce the organic appearance of the plant. Second, and most important, is asymmetry. In this case, I define asymmetry as branches spawned from the same point having different rules. Since the L-System builds strings recursively, small differences between these branches result in a highly varied final structure that looks much more natural than its symmetrical counterpart. As a result, I emphasize asymmetry in the construction of the L-System rules.

I generate L-System rules through a CFG that iterates to a predetermined depth. While I wanted each plant’s L-System rules to be generated uniquely without human intervention, the rules for the rule-generating CFG are rules I decided upon and manually entered after some experimentation (see Limitations & Future Work for more discussion). I decided upon three major types of internodes:

- Branching internode (BrIn)
  - Branching internodes spawn multiple branch-type internodes or branching internodes outward from the same point with some change of heading in between them.
  - For example, if we define the branching internode A → [[FA]&&[IFA]&&[IFA]], this rule spawns three new branches spread out from each other.

- Drawing internode (DrIn)
  - Responsible for increasing the growth of the stem by expanding a drawn unit into multiple drawn units, and may also spawn new branches.
  - For example, F→ FF will cause the length of the stem to double after every iteration.

- Branch-type internode (BrTyp)
  - Branch-type internodes define new branches. Branches can spawn other branches and are responsible for growing leaves.

I define characters A through E as branching internodes, F through K as drawing internodes, and S through X as branch-type internodes. For each of these characters, I use the predetermined CFG rules.
to generate a string rule appropriate to the internode type of that character. The system then continues to expand this string to some depth, at which point it is returned as the final L-System rule for that character. Every word in the CFG has multiple possible expansions, one of which is selected during each single expansion.

A complete set of the CFG rules can be found in the source code for the project, but key features include:

- Chaining angle-changing symbols to random depths
- Chaining F’s to random depths
- Turn-segment (TSeg), which consists of some amount of chained angle changes, followed by a chain of Fs and an internode. This creates the branching structure used by branching internodes.

Once the L-System rules for a plant are created, they are finalized, stored within the plant object, and serialized and saved for reuse upon saving and loading the game. Because the rules are finalized after plant creation and cannot change, the rule-generating CFG is an appropriate place to include randomness while still making final plant generation deterministic. The multiple options possible for each expansion and random selection of one of these options means that every plant has a unique rule generated for each character, giving each plant a unique structure as was my goal.

Finally, several plant attributes are also selected randomly upon plant creation and serialized with the plant object upon saving the game; since these do not change after the plant is created, plant generation remains deterministic. Although the L-System rule generation is most responsible for the unique forms of the plants created in the game, the random selection of the attributes also contributes heavily to the final structure of the plant. Randomized attributes of each plant include:

- Unit length of each F-segment and f-segment
- Radius of the stem, and how much and how frequently the radius increases over time
- How much the angle changes for each special angle-changing character + - & ^ ,`
- How much the radius of the stem decreases after !
- How much time passes between iterations of the L-System
- Which material to use for the stem
- The color of the leaves

As seen in Figure 6, variation of these attributes alone can create quite different plants. Every plant in Figure 6 was created with the same set of L-System rules, but has different values for its unit length, radius, and angle changes.
The result of all of these components brought together is the generation of unique plants that vary greatly in size, shape, structure, color, and growth rate. Figure 7 shows the kind of variation possible using the current system.

Figure 6: Multiple plants created from the same set of rules but with different attributes.

Figure 7: Five different plants generated by the current version of the application
Deliverables

The primary deliverable of this project is the completed iOS app that I have successfully installed on an iPhone 5 running iOS 9 in which the user can play this game. The game has the following characteristics:

- Users can choose to start a new game, or can continue playing a saved game (if one exists) which will load their saved plants.
- Users can move around the world by pressing the trigger of the cardboard headset.
- If the user is looking at the nearby ground and it is possible to plant a new plant there, they will see the option to do so. Pressing the trigger at this time will create a new plant in this location.
- Plants develop over time. When a user continues a game, plants will have undergone growth if enough time has passed for them to do so. Growth includes more iterations of the L-System, larger radius size of the stem, and large leaf size.
- The app responds to head tracking, so moving around while wearing the headset will allow the user to look around and navigate the scene.

Source code for my C# scripts can be found at https://github.com/rebeccanickerson/cs490-project.

Limitations & Future Work

The main limitation I have identified in my work is the manual assignment of the rule-generating CFG rules. While I made an effort to create rules that would generate “good” L-System rules - “good” meaning that the plants generated from the rules are realistic and diverse - my system has shortcomings. For example, while there is a lot of variety in the exact structure of plants, most are still very tree-like or shrub-like and I would have liked to generate different kinds of plants like ferns, small flowering plants, and so on. Also, there is the possibility, although slim, of the current CFG generating L-System rules that result in poorly formed plants, e.g. plants that immediately burrow underground and are not at all visible to the user. In an ideal project, I would have generated the rule-generating CFG rules through machine learning, eventually developing a CFG that generated near-optimal L-System rules. Given the limited time in the semester, I did not have time to implement machine learning in addition to everything else in the project.

Another limitation is the extremely sparse user input, which is partially a side effect of the Cardboard headset having only one trigger. In the app, I use this single trigger for user movement within the world, selecting menu options, and planting new plants, but would have preferred to have more input options and more interaction between the user and the plants. For future development, I could add options to move or remove plants after selecting them, or require the user to water them to stimulate growth.
Sourced Assets and Code

In the process of developing this project I used the following assets from the Unity Store. They are available to download for free and are not included in my Github source.

- Foliage Pack Free - Jake Sullivan (terrain grass texture and grass detail)
- Yughues Free Sand Materials - Nobiax / Yughues (terrain sand texture)
- Yughues Free Bark Materials - Nobiax / Yughues (bark material)
- Wispy Skybox - Mundus Limited (skybox)
- Nature Starter Kit 2 - Shapes (skybox)

I used the Google VR SDK to integrate the project with Google Cardboard. The SDK code is owned by Google [4].

I also used Paul Welter’s code for creating an XML serializable dictionary to store L-System rules in the file system [8].

Acknowledgements

Many thanks to Holly Rushmeier for supporting me in this project and advising me throughout the semester.

References


