Bendy Plants: Plant Structure Generation with Object-Oriented L-Systems and Physics Simulation

Pravin Visakan
UCLA
Los Angeles, California
pravin.visakan@gmail.com

***Abstract*—Real plants exist as physical entities in a physical world, and the interaction between the two can affect overall plant structure. This project explores the use of physics simulation to produce physically-influenced structures. In the course of doing so, a variant of the L-System formalism is developed. This more object-oriented plant model leverages common programming constructs and compatibility with common game engine architectures.**

***Index Terms*—L-Systems, artificial life, plants, physics, computer graphics**

I. Introduction and Motivation

Fig. 1. Examples of physical (bendy) plant structures in nature

In the real world, the branching structure of plants is highly varied and is influenced by a number of factors. [1] These can include light and proximity to resources such as water, as explored in other projects such as that by Radomir et al. [2]. Perhaps one of the most dramatic can be seen in the effect of mechanical forces on that operate physically on these structures. The leaves of a willow tree droop downward; the head of a flower may bow under its own weight. It is the opinion of this author that these structures can be quite graceful, and so in this project I set out to generate them procedurally.

To do so, I will leverage previous work [3] done in the domain of computer modeling of plants- namely that of the Lindenmayer System, or L-System. As will be explained in a quick overview in the next section,

II. L-System: Theory and Rendering

An L-System is a formal method of modeling the growth and development of plants. These are similar to the notion of

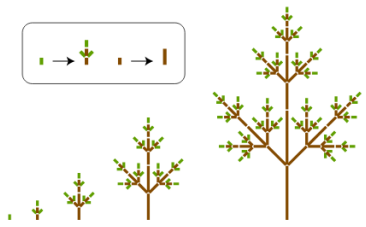

Fig. 8: The first five stages of the developmental of a simple branching structure modeled using the L-system in Table 4. The inset shows a graphical representation of the productions.

An L+C program equivalent to the symbolic notation in Table 4 is shown below:

```
module A;          // apex
module I(float);   // internode (length)

Axiom: SetWidth(0.4) A;

A :    produce I(1) SB Left(45) A EB SB Right(45) A EB I(1) A;

I(s) : produce I(2*s);

interpretation:
A    : produce SetColor(1) f(0.2) F(0.8);
I(s) : produce SetColor(2) f(0.2) F(s-0.2);
```

Fig. 2. Example of a rule set definition, both graphically and with code. From [3]

formal grammars; given a string of symbols, further symbols are derived by applying one of several rules, or productions, to them. Symbols in a string may also have some amount of state, upon which arbitrary logic can be executed. The combination of a symbol and its state can be interpreted graphically to produce a tree. In Figure 2, we can see the rule set and graphical interpretation of a simple binary tree from Prusinkiewicz et. al. The tree is a product of a string composed of green A nodes and brown internodes. The internodes have a state variable, length, that doubles for each discrete growth step. Further explorations of this idea have been explored, including the use of context sensitivity in the grammar to model message passing through hormones. Stochasticity, can be introduced to avoid regularity in the pattern, which can decrease fidelity. [4]

Each symbol produced by an L-System is assigned a meaning. In the conventional model, these are taken to be imperative instructions given to a turtle. This turtle navigates, or draws out the plant as prescribed by the current symbol in the string, before moving on to the next. In two or even three dimensions, this can produce functional and lifelike images of plants. [3]

For the current application, this model produces some challenges. The original plan of this project was to interweave steps of growth/production from an L-System with an interval of physical simulation, in order to produce the curved shapes.

Turtle command	Symbol	L+C keyword
draw line segment	*F*	F
move without drawing a line	*f*	f
turn left | right	+ | −	Left | Right
bend up | down	^ | &	Up | Down
roll to the left | right	\ | /	RollL | RollR
start | end branch	[|]	SB | EB
set line width	#	SetWidth
set line color	,	SetColor

Table 3: Basic turtle commands in a symbolic notation and in the L+C language.

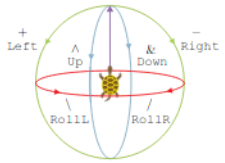

Fig. 7: Specification of turtle rotations in three dimensions.

Fig. 3. Symbols of an L-System, and corresponding turtle movements. From [3]

It is not immediately apparent how a turtle is to learn the physics necessary to redraw the plant in our case, without some awkward extensions in both the metaphor and the architecture of the code based on it. Additionally, the formal step of introducing symbols and rules and managing state in an ad-hoc manner can get awkward.

In the next section, I'll discuss the design principles behind the game engine used in the project, and how that informs an alternate, object-oriented conception of L-Systems.

III. The Godot Game Engine and Physical Simulation

Godot is an open-source game development environment similar to popular suites such as Unity or Unreal. In Godot, digital environments are composed into scenes, which themselves are composed in a hierarchical manner with node objects [5]. Other engines use similar design principles, and the use of the hierarchy is useful for a few reasons. For one, it allows easier organization of complex environments, collaboration between people, and integration with various components with different, complex logic. For another use, it allows the construction of hierarchical graphical models of several objects, that all move relative to each other by using the appropriate transforms. [**?**]

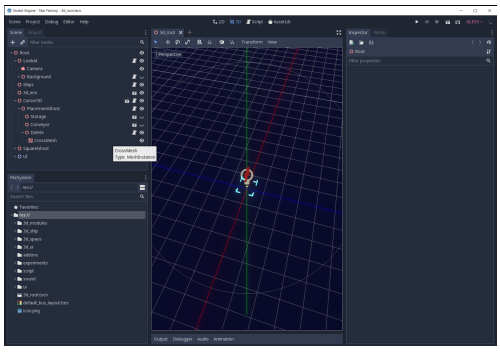

Fig. 4. The main Godot interface. Note the Node Hierarchy on the left.

Physics simulation is achieved by defining individual rigid or soft bodies as nodes, and introducing them into the scene tree.

As is evident, the conception of graphics as artistic turtle does not map easily to this structure. One could implement the requisite symbol generation and graphics system on top of this one, but to do so would be in opposition to design practices of modern systems. In the next section, we highlight a modification to the L-System concept that more easily integrates with environments like Godot, and can better accommodate physical simulation.

IV. Object Orientation

For this project, a sort of *object-oriented* L-System was used. Instead of symbols in a string, a set of objects is used (Nodes in Godot parlance, but these could be anything in practice). This conception leverages the object oriented features of modern programming languages. Nodes can inherit behavior from each other. State changes are better encapsulated within the relevant nodes, and arbitrarily complex logic involving stateful changes can be more easily implemented. The use of pointers to create a tree structure directly, as opposed to inferring on from a string of symbols, is also an advantage. Defining a model purely in code does have the disadvantage of perhaps being less useful in other disciplines of plant science. But, this model is more compatible with conventions of software and game development. This is in contrast to the earlier implementations [3], in which extensions to the C programming language were used (see Figure 2).

```
1  extends MeshInstance
2  class_name Symbol
3  # Generic "base" class for symbols
4
5  #Constructor
6  func _init():
7      pass
8
9  # modifies the given list with this symbol advanced one iteration step
10 func grow():
11     self.grow_children()
12
13 # recursive function, grows all child nodes, DFS order
14 func grow_children():
15     var children = self.get_children()
16
17     for child in children:
18         child.grow()
19
```

Fig. 5. GDScript implementation of the base "Symbol" class

This configuration should also make it simpler to integrate physics simulation into the model - however, as we'll see in the Results section, there were complications.

V. Results

Figure 6 shows the Godot implementation of an object oriented L-System in action. It uses the equivalent of the ruleset seen in Figure 2, with some modifications - four branches are added at each step and arranged around each other, in a way not possible in 2D. The A Nodes (seen as oblong shapes) spawn two I symbols and more A Nodes; the I Nodes (the cylinder meshes) double in length in each turn. The tree progresses in discrete steps with time, and forms a simple demonstration of the concept.

Figure 7 depicts an object-oriented L-System equivalent to the the stochastic grammars described in [4]. In the original, formal conception, multiple applicable rules for a given symbol were given. The interpreter would select among these

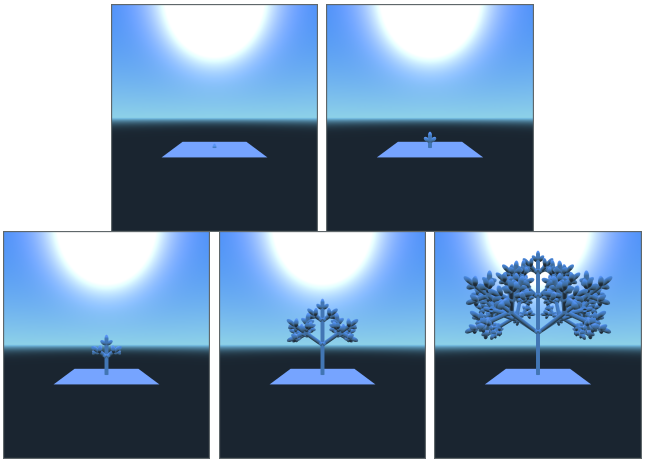

Fig. 6. Growth progression of an L-System over five iterations

with some weighted probability. By defining a model of this kind, a modeler could introduce randomness in the growth patterns of their plants, mimicing the seeming arbitrary shapes of plants and increasing fidelity. Later models [2] instead used interaction with light or water to get similar effects more inspired by biology - but the random culling appears to perform fairly well. In this model, the randomness is generated imperatively, and used to decide on the placement and number of branches.

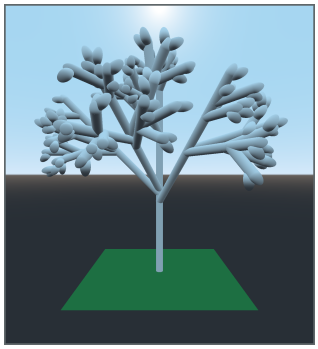

Fig. 7. An L-System tree with stochastic/random culling of branches.

Each Node tracks its children, or subsequent Nodes. The tree structure of the tree model is reflected in the tree structure of the object hierarchy. For every step, a simple recursive call is made to the base node. This allows for the simple execution of arbitrary logic, such as that seen here with the branch culling.

The use of physics components ended up being more troublesome, in practice. Figure 8 depicts one of many issues run into. The underlying environment, Godot, was thought to have robust support in this area, but upon use the soft body dynamics was found to be somewhat underdeveloped. Though details on the underlying implementation were scarce, it seems from demos that the primary use case was cloth simulation. Implementing stiffer materials, or materials that bore the weight of other objects (such as other apices) ended up exhibiting clear numerical errors. It is possible that it uses a mass-spring-damper system based on the explicit Euler integration method. As Figure 8 shows, the native soft-body support was not robust enough for this purpose.

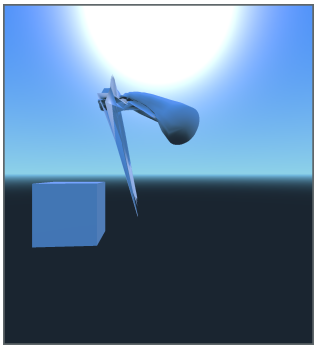

Fig. 8. The Godot soft body implementation, attached to a rigid object. One of the more egregious failures.

Alternate models, such as one based on multiple rigid bodies, were attempted, but also bore stability issues.

VI. Conclusions and Extensions

The reconception of classical L-Systems to object oriented L-Systems remains valid. The movement from structures of symbol strings to entirely code-based models makes sense for code-based uses of L-Systems. More thought needs to be put into implementing a physical model. A finite element approach, or the use of different tools, may have made the process easier. The effort was useful, however, and can still holds promise for this and other applications, such as those involving more complex logic and interactions.

References

[1] James D. Mauseth, 2003. Botany: An Introduction to Plant Biology, Third Edition.

[2] Radomir Mech and Przemyslaw Prusinkiewicz. Visual Models of Plants Interacting with Their Environment. Proceedings of SIGGRAPH 96 (New Orleans, Louisiana, August 4-9, 1996). In Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 397-410.

[3] Przemyslaw Prusinkiewicz, Mikolaj Cieslak, Pascal Ferraro, and Jim Hanan. Modeling plant development with L-systems. In Richard J. Morris (Ed.) Mathematical Modelling in Plant Biology, Springer (in press).

[4] Prusinkiewicz, Przemyslaw, and Aristid Lindenmayer. The Algorithmic Beauty of Plants. Springer-Verlag, 1996.

[5] "Scenes and Nodes" Scenes and Nodes Godot Engine (Stable) Documentation in English, docs.godotengine.org/en/stable/getting_started/step_by_step/scenes_and_nodes.html.

[6] "Using 3D Transforms¶." Using 3D Transforms Godot Engine (Stable) Documentation in English, docs.godotengine.org/en/stable/tutorials/3d/using_transforms.html.