ShrubHub: Procedural Plant Generation

Antonio Fiol-Mahon

INTRODUCTION 2

MOTIVATION 3

RELATED WORK 4

Algorithmic Beauty of Plants 4
Fast Simulation of Realistic Trees 4
Evaluation 5

ALGORITHM AND TECHNIQUES 6

Overview 6
Plant Representation and Generation 6
Species Generation System 7
Model Creation & Visualization 9

RESULTS 10

System Performance 10
Limitations 11
Testing 11

CHALLENGES ENCOUNTERED 13

Seamless Model Creation 13
Uniqueness of Results 13

CONCLUSION 14

BREAKDOWN OF WORK 15

Antonio 15
Matt 15

Introduction

This report outlines the creation of our graphics final project, ShrubHub: a procedural plant model generator. The goal of this project was to create a robust system that can be used to synthesize a wide variety of different plant models using a structure that makes it easy to specify a high-level description of a desired model appearance, and then produce a variety of models using that species description. A system like this would be a useful tool for generating assets to populate a natural environment with believable looking plant models. If a full workflow was built around this system, it could free designers from many tedious aspects of asset creation and allow them to focus on a high-level goal appearance, leaving the tool to generate many different individuals that the designer can select for use. ShrubHub can allow human creators to be more productive by eliminating time consuming and less creative aspects of modeling so they can focus on meaningful aspects of design. This tool would make certain modeling tasks more like keyframing, where high level goals are described, and the computer uses that description to create the bulk of the assets.

Motivation

Figure 1. Bonsai Like Model

My initial motivation for this project arose out of an interest in procedural generation, and simulating life/evolution. I was interested in software that is able to emulate some of the most interesting aspects of natural life processes, such as developing a coherent result using a set of rules and randomness. I found the "ordered chaos" of these systems fascinating because high level inputs can produce unintended and surprising results, allowing for many outcomes that were not initially expected from the creator. It is an exciting and dynamic process to find and describe the types of rules that capture desirable abstract plant qualities but also allows new designs to emerge organically. Creation of a coherent asset from start to finish is typically something that people do manually, so I found the idea of trying to develop rules and describe a generator that captures qualities that make something look "organic" or plantlike without over constraining the system to produce only one type of plant an interesting exercise.

Related Work

Algorithmic Beauty of Plants

Algorithmic Beauty of Plants by Przemyslaw Prusinkiewicz and Aristid Lindenmayer is one of the most influential texts in the narrow field of procedural plant generation. The book offered a description of many models and techniques for characterizing the highly variable and complex appearance of plants, ranging from tools for describing the way that plant structures grow and the rules for their development to geometric descriptions of plant organelles such as flowers and leaves. One of the most notable contributions of this book is the L-System, named for the late author of the book Lindenmayer. L-Systems use formal language grammar systems such as those taught in computer science to describe the recursive way that plants grow out from themselves to build large complex structures. While ShrubHub doesn't specifically use L-Systems, the L-System generation influenced the decision to use a language system to describe high level plant structure. The main difference between L-Systems and ShrubHub is that L-Systems recursively apply their language rules to replace existing structures with bigger ones, while ShrubHub applies the L-System in a linear fashion to describe transitions between components in different areas on a plant. Extending ShrubHub to grow using an L-System would be a straightforward task.

Fast Simulation of Realistic Trees

This paper provides an overview of a sophisticated new algorithm for quickly computing forces on a tree and propagating updates throughout the tree with a kinematic chain. Trees share many properties with smaller plants, and because all plants created by ShrubHub are described as a graph of plant nodes, it would be straightforward to incorporate this simulation

technique into our models to create realistic dynamic visuals. Upon further investigation, the main methods described in this paper ended up being too complicated and large in scope for ShrubHub. The contributions reliance on complicated dynamics and linear equation solving techniques and would have been more than enough to be the subject of its own project. Despite this, the paper provided a very useful overview of other existing approaches to animation/simulation with varying degrees of believability and physical accuracy. We decided to model our animation approach off the angular spring technique discussed in the paper. By treating the intersection between branches as an angular spring that oscillates back and forth form a specific starting point, a gentle swaying motion can be created that captures the appearance of swaying in the wind, and suggests that the plant being rendered is a malleable natural form and not a collection of static geometry. This effect compounds well throughout the structure, and the small sway applied at each point propagates up the structure, starting with a slight bend at the bottom that becomes a faster yet gentle movement at the top. While this approach is far from dynamic or capturing realistic motion, it adds a lot to the generator by giving the user an animated representation of the model for immediate feedback and exposes discontinuities at model joints that are only apparent during model movement.

Evaluation

Most prior work focuses on generating or characterizing a specific realistic plant form with some small variability rather than sacrifice realism for a more generic system (breadth vs depth). A more generic system has a lot of value considering how many graphics applications such as video games strive for variety and favor abstract design over realism. There is certainly a large potential use case for tools like ShrubHub which aim to be a tool for describing many

types of plant forms, and not exactly replicate a specific feature or trait of a real plant. Additionally, a language rule system allows the description of more particular systems to augment the base generator, allowing for designers to extend the base tool with more specialized functionality.

Algorithm and Techniques

Overview

To ensure that ShrubHub would be sufficiently useful as a basis for procedural modeling, the main design goal was to create a system for generating plants that all conform to a species description, allowing the designer to iteratively determine a high-level description of the desired plant, then subsequently create an arbitrary number of samples from the population. To achieve this, the ShrubHub generator works in two distinct phases, species level generation and plant level generation. After a plant model is created, a visualization stage presents the model in an interactive context. A key advantage of this system is the ability to independently update the species and the plant seed, allowing the user to visually observe the effect of a species level modification on an existing instance.

Plant Representation and Generation

Plants are represented using a tree data structure that stores branches as plant nodes. Plant nodes store information about individual branches such as their depth in the tree, children nodes, geometric properties, and a species description. A plant consists of a root node that has no parent and contains recursive references to all children. A node can have an arbitrary number of children, which represent various plant features like the trunk and

branches. All plant nodes within a plant share a reference to their procedural random generator, which ensures that a plant is always created in the same order, allowing for the generator seed to completely characterize a plant model. This simple and robust data structure implementation is well suited to modeling a plant because it captures the hierarchical and spatial structure of plant branches and connected organs and makes it easy to reason about the state of a plant and walk the graph to perform operations. To create a full plant from an initial plant node, a grow method recursively adds a new randomly generated child to any nodes that do not have children. Randomly generated children follow the rules specified in the parent nodes species, which provides a range of possibilities for the child's appearance. The language system augments the species description and modifies the species base trait probabilities to create specific features.

Species Generation System

The species generator determines high-level plant characteristics, so all plants of the same species have visual continuity. To achieve this, ShrubHub use two separate mechanisms: random range generation and a language tree. Random range generation is relatively straightforward. A range data structure stores the min and max values for all generator defined plant traits, such as branch length, RGB color, branch angle, branch probability, and more. When a new branch node is added to the tree, the generator picks values within the ranges described by the species, and this ensures that the variability within a specific plant is limited so there is a specified average value and variability from the mean.

While the range selection is great because it ensures a uniformness and is a convenient format to specify, these same qualities cause problems when the generator values are too

uniform to create meaningful variety or structure. This is where the language system comes in. Plants can be specified via a graph of transition possibilities between components, where each component has specific properties enforced. This allows for discrete local phases of generation in different parts of the plant and provides the designer a means for describing arbitrarily complex forms. To create a plant with a trunk, branches, and twigs, the individual characteristics of each component can be given a token and transition probabilities between components to organize them in the graph. This allows specification of overall traits and then specific composition of the structure where needed.

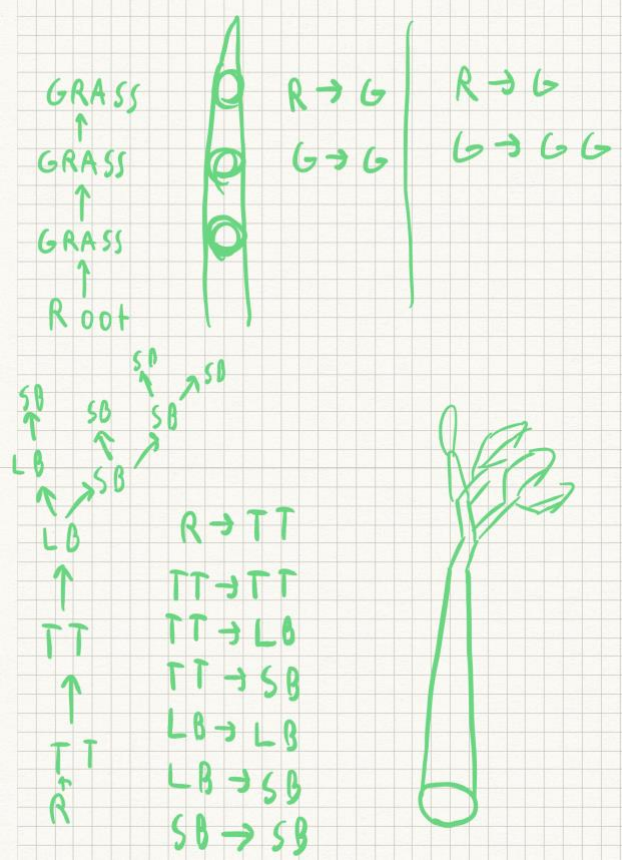


Figure 2. Diagram of Composition Rules top right grammar is a traditional L System, on the left is the corresponding append grammar. Other grammars describe what structures can be added as children of the left hand symbol

Model Creation & Visualization

Branches are mainly visualized using cylinder primitives with a longer radius which is further down the tree and a shorter one higher up. A sphere is placed at the intersection between nodes and centered on their end radius. This allows the geometry to pivot arbitrarily without showing the flat top of the cylinder, giving the idea that it is one continuous piece of geometry. To build a plant, the graph structure is walked, and each branch is positioned relative to its parent recursively so that the entire structure is built up from the reference frame of its parent. When the angle of a branch away from its parent change, its position update propagates through the kinematic chain and its descendants are correctly updated. To animate the structure, each branch is rotated about its bottom face using a sine function to create a smooth rocking motion. The small angle difference is magnified as the motion increases up the tree, giving the trunk a subtle wave and the branches a faster wider swing. This animation is simple and not intended to be physically accurate but provides a way to view the model in motion and adds to the realism and plausibility greatly.

Results

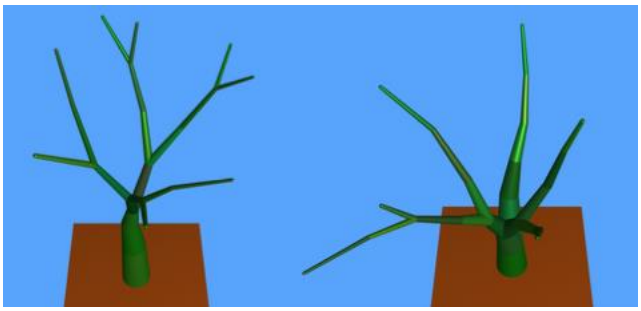


Figure 3. Trees from the same species

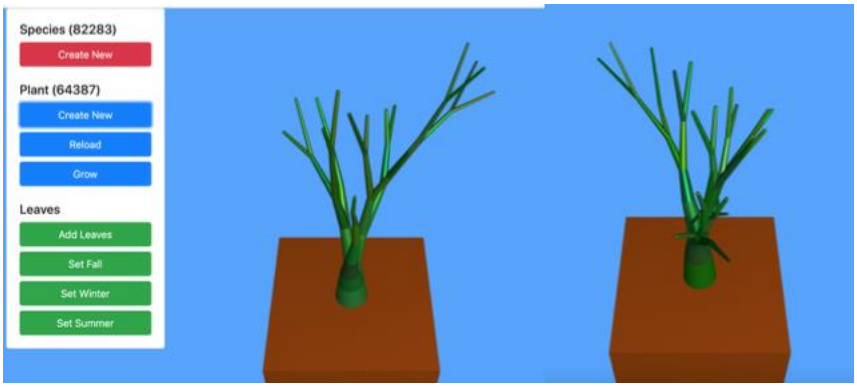


Figure 4. Fern Like Species

System Performance

The system developed is a working prototype of a plant generation tool that demonstrates the core functionality of the system, without spending too much time on a UI. To configure parameters such as the L-System and pass custom species definitions to the generator, the scripts must be modified. The species level generator creates a spectrum of possible plants that follow a species high level design goals. The plant level generator uses a species description to generate a plant data structure and model that adheres to the constraints of the species, then visualizes the model in an interactive view. One of the main features of the generator is the deterministic nature of the species and plant generation. A seed value allows the plant and species to be separately characterized, allowing the exact same

model to be recreated if the same species and plant seed are provided as input. Aside from repeatability, this has the benefit of allowing species level changes to be tested on the same input model, streamlines development of a new species with different parameters because the original species can be observed before and after a species change. This allows most parameters of the plant to remain unchanged through species modification, and the user can see specific updates to the length, color, etc. in real time, which is ideal for model creation. The generator created great results that were consistent with our goals, allowing variation at species and individual level.

Limitations

The main limitation of the current system is the lack of interface for creating more custom models. All the current logic runs through the random number generator, but the system is organized so that providing custom overrides would be a straightforward process. Additionally, the L-System that is currently in use is defined via a production application function, and not through data. A more general function that could parse any language specified in a dictionary would make it easy to select from a variety of systems and allow for specific ones to be applied as needed.

Testing

Testing a procedural generator is complicated, because many functions cannot be empirically benchmarked. The test phase had two main goals: ensuring repeatability and ensuring the entire range of possibilities using provided generator values were plausible organic forms. To ensure repeatability, throughout the process we tested that the same seed combination continued to generate the same plants without changes in between successive

generations. Updating the generator will change the way random numbers are drawn and repeatability between generate updates is not guaranteed, so by ensuring that several back to back generations are identical, the systems deterministic guarantee can be verified.

To test that results are plausible organic shapes, many sample plants are generated and evaluated to ensure that most of the plants don't have "plant breaking features". Plant breaking features are defined as cracks in the mesh or sharp edges between connected branches (not the angle of the branch), unnatural shapes (branch length shorter than radius, branch length implausibly long without arcing). It was also important to observe qualitative aspects and ensured that it was in line with preconceived ideas about what constitutes a plant shape. The generators correct function can be verified by testing these traits throughout development.

Challenges Encountered

Seamless Model Creation

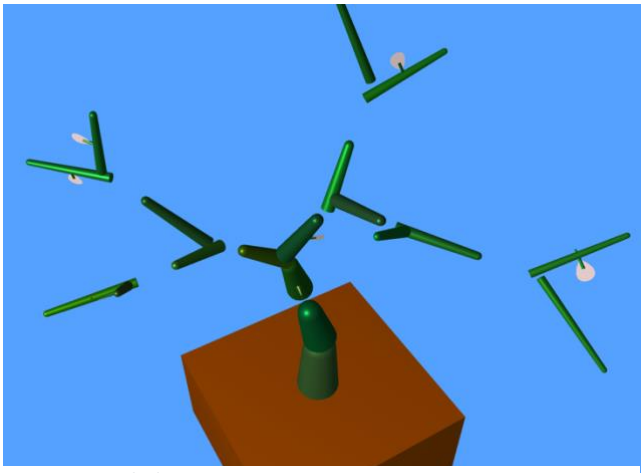


Figure 5. Exploding Tree

Smooth model creation was initially a challenge. The system needed to ensure that no matter how it created a branch, it would connect to its parent without discontinuities or seams along the edge. In addition, many subtle defects quickly compounded into explosions as they compounded up the branch. Seams significantly hurt the realism and believability of results and made the model much less "plant like". To prevent this, a sphere is placed on the axis of rotation that intersects perfectly with the radius of the branch ends. This allows the branch to rotate arbitrarily about the parent without exposing sharp edges on the cone segment.

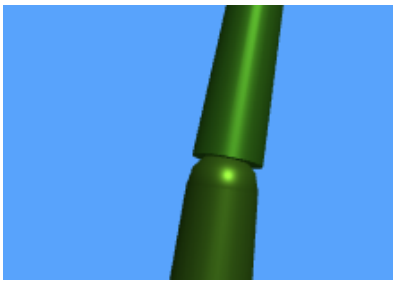


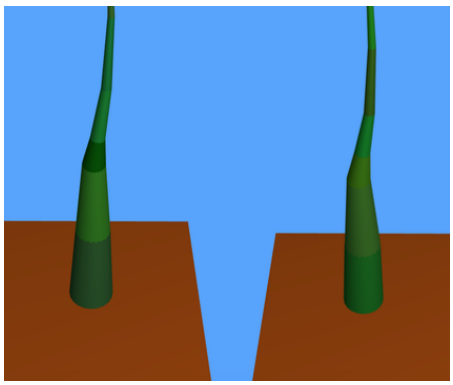
Figure 6. Seam between Joints Illustrating Sphere connector gap

Uniqueness of Results

Creation of results that are visually distinct from each other is a common problem encountered with procedural generation. With branching structures, high density can create issues in a variety of ways, and the geometric form of the plant tends toward a dense bush with high parameters. It was a challenge to ensure that models created were both unique from one another, while remaining clearly identifiable as plants. The challenge of implementing a

stateless generator prompted the addition of the Language System to provide high level direction for the structure. This has a variety of benefits, the generator can be more customized and its potential use as a real tool increases, and it makes it easy to describe meaningful relationships between parts of the plant. However, a drawback is that this requires rule writing and if excessive rule creation is necessary, the purpose of using a generator for model creation is defeated, and humans are still forced to do a large amount of work for desirable results. By creating a set of broad rules, the plants can be given a variety of generic forms without sacrificing most of the generality of the system, and it makes an appropriate addition to the systems power.

Figure 7. Similar Grass



Conclusion

With believable results and believable results, the project does what it sets out to do well, proving that a tool like this could be successful and make a good extension to modeling and environment creation workflows. The main thing that prevents this from being used today is a robust UI, and control over more aspects of the system to make extension easier. A good extension to this project would be a way to modify species without changing the seed and providing a UI for tweaking individual parameters of the species and individually generated models. Aside from modeling tool extensions, a recursive growth system could be implemented on the existing plant data structure to provide for an L-System defined plant to be generated with the tree.

Breakdown of Work

Antonio

- Initial Research
    - Concept
    - Sources
- Entire Concept Report
- Entire Final Report
- Entire Presentation (Except for leaf generation slide)
- Project
    - Entire design and implementation of project except for:
        - Leaf Generation Code
        - Leaf Placement Code
        - Initial Model View Camera Controller Plugin Setup
        - UI Buttons for leaf placement/customization

Matt

- Leaf Generation/Placement Code
- Presentation Leaf Generation Slide

Bibliography

[1] J. P. Weber, "Fast Simulation of Realistic Trees," *IEEE Computer Graphics and Applications,* no. May/June 2008, pp. 67-75, 2008.

[2] P. Prusinkiewicz and A. Lindenmayer, The Algorithmic Beauty of Plants, Przemyslaw Prusinkiewicz, 2004.