Documentation:

Intro:

The task was to create a single C# file that can procedurally generate a tree where the logic does not have any external dependencies and should work in Unity 6. The chosen logic for procedural generation was taken from the work of Aristid Lindenmayer, specifically his co-written work, The Algorithmic Beauty of Plants. While not a particularly practical solution for procedural tree generation in a 3D simulated world, it does present, in my opinion, an interesting numerical solution for graphically representing trees with strong ties to a number of other scientific disciplines.

The L-system is the main driver of the logic. For the purposes of my C# script, I have implemented a number of example L-systems present in The Algorithmic Beauty of Plants as well as the option to create your own L-system for the purposes of generating trees. A lot of underlying knowledge is required to use L-systems to more accurately generate realistic looking trees; however, by playing around with freehand setups, the user is able to create interesting shapes approximating a tree or otherwise. For example, L-systems are able to draw fractal patterns, non-tree geometric shapes, etc. The work I reference is specific about trees and tree related geometric shapes. There are tens of other references one could use to explore the concept of L-systems more deeply than just creating trees.

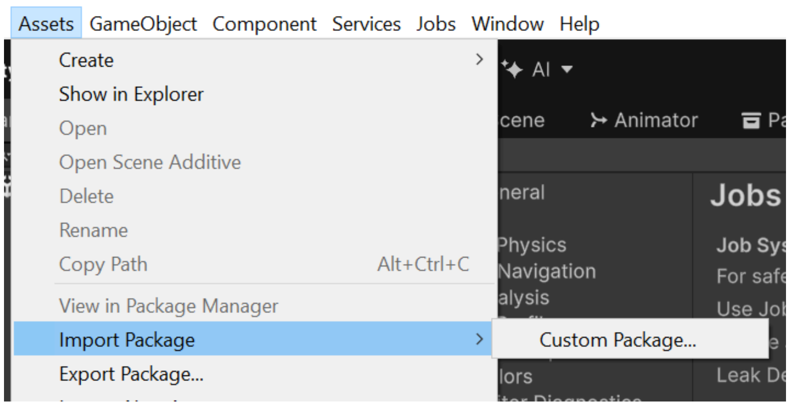
I will word everything in this document as if I am speaking to someone brand new with Unity. This doesn't mean that I believe every person who reads this is assumed to be incapable of using the engine.

High level flow of the the L-system:

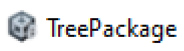
1) Start from the axiom.
2) Pass the axiom to the logic for handling generating subsequent generations. Rules are constructed to ensure each character produces a new set of characters.
3) Recursively run this logic until the desired number of generations is reached.
4) Parse the final string and implement logic based on every single character.
5) Depending on more complex logic (age, probability ranges, etc.), the L-system can either produce non repeating patterns or leaves/fruits/etc. on your tree.

Setup:

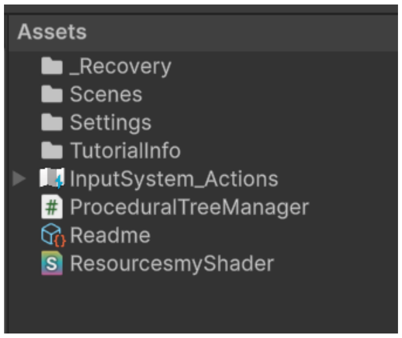
This document should come with a .unityasset file. This file contains the C# script that contains everything and can be imported by Unity. The logic is entirely contained in a single class that inherits from MonoBehavior. So you just need to add it to a game object in your project's Hierarchy for it to work whenever you hit play.

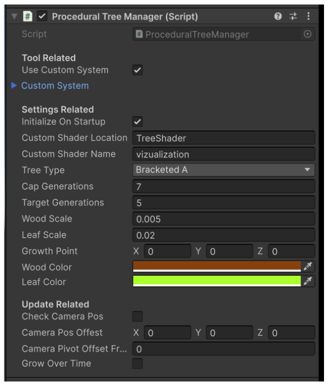


Select Custom Package… to find your package from wherever you saved it on your computer.



File to import.





Procedural Tree Manager (Script)

Script	ProceduralTreeManager
Tool Related	
Use Custom System	✓
Custom System	
Settings Related	
Initialize On Startup	✓
Custom Shader Location	TreeShader
Custom Shader Name	vizualization
Tree Type	Bracketed A
Cap Generations	7
Target Generations	5
Wood Scale	0.005
Leaf Scale	0.02
Growth Point	X 0 Y 0 Z 0
Wood Color	🟫
Leaf Color	🟩
Update Related	
Check Camera Pos	☐
Camera Pos Offest	X 0 Y 0 Z 0
Camera Pivot Offset Fr...	0
Grow Over Time	☐

ProceduralTreeManager.cs imported in a fresh project and added to an empty game object. There are a number of serialized fields barely visible in the right image that will be discussed in greater detail below.

Usage:

Serialized fields, specifically the attribute [SerializeField], are private (or public, but not relevant here) members exposed in Unity's inspector. Every value modified is able to affect the results of the script's logic at runtime. Do not become emotionally attached to any of the default values. Some of them are more important than others for reasons that will be discussed. The less important ones can be considered subject to the preferences of the user.

API Related:

Definitions are exposed and could theoretically be used for any L-system purpose. Doing so would require a developer to implement their own logic that converts the L-system to a graphical representation. I do not have any XML documentation in Unity. These descriptions will differ from what you see when you look up the code of what each thing does.

1) LSystem (class): Main data structure for handling everything.
    a) IsValid() (returns: boolean): Quick check to see if this system is set up to work without errors.
2) sRule (struct): Ties a letter to either one or many results when a new generation is generated.
3) sStochasticRuleHandle (struct): Wrapper for allowing a rule to be constructed where one letter can have different results depending on sampling a value from 0-1.
4) ParameterizedVariable (class): Allow a character in the L-system to have variable behaviors based on parameters external to the L-system itself. Defaults to having age as a parameter for the sake of animating a tree's growth. Inherit from this functionality with a child class to create your own examples.
    a) Initialize(currentAge (float), c (char)): Can be overridden.
    b) IsValid() (returns: boolean): Can be overridden.
    c) IncrementAgeByAmount(amount (float))
5) ParametrizedForwardMovement: A given example where the displacement character in the L-system can have different lengths.
6) ParametrizedAngleMovement: A given example where the rotation character in the L-system can have different angle values.

Tool Related:

These two values are related to construct one's own L-system.

1) _useCustomSystem (boolean): If set to false, the script will create a hardcoded tree based on a value that will be discussed later. If set to true, _customSystem will be used instead.
2) _customSystem (LSystem):
    a) SystemLabel (string)
    b) Axiom (string): Generation 0. The tree is built from this starting point.
    c) Rules (List<sRule>): Mechanism governing how the tree grows. Depending on how each letter in the L-system is handled, construct appropriate rules to iterate through generations and grow your tree.
    d) Length (float): Distance between characters in the L-system that govern displacement.

e) MinLength (float)
f) MaxLength (float)
g) RandomLengths (boolean)
h) Angle (float): Angle difference applied when reaching characters in the L-system that govern rotations.
i) MinAngle (float)
j) MaxAngle (float)
k) RandomAngles (boolean)
l) GrowRate (float): If growing your tree, it is recommended to use a low value to prevent the growing happening too quickly.

Settings Related:

These are generic settings that apply broadly rather than any specific thing.

1) _initializeOnStartup (boolean)
2) _customShaderLocation (string): Folder to be generated in the Resources folder. Your shader will be saved here. Omit any forward slashes.
3) _customShaderName (string): Name of your shader. Omit .shader or any forward slashes.
4) _treeType (eTreeType): Select L-system from hardcoded examples.
5) _capGenerations (int): Keep the value relatively low (~7). Large values of generations (n) can cause unmanageable exponential growth in computation requirements without any meaningful gains.
6) _targetGenerations (int): Keep the value relatively low (~5) and less than CapGenerations. If you test this with too large of a number, you will hurt the performance on your computer. Different systems work better with different values.
7) _woodScale (float): Adjust the size of wood components.
8) _leafScale (float): Adjust the size of leaf components.
9) _growthPoint (Vector3): Adjust where you want the tree to grow in 3D cartesian space.
10) _woodColor (Color): Adjust the color of wood components.
11) _leafColor (Color): Adjust the color of leaf components.

Update Related:

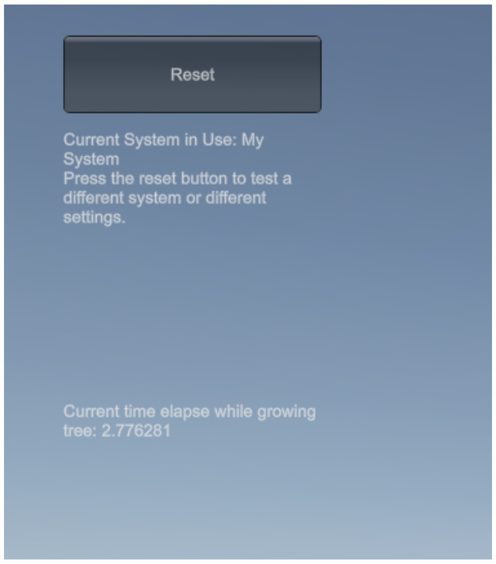
These are things related to logic that fires every frame. They may or may not be related to the L-system itself. The camera controls implemented here are not indicative of how I would normally handle user inputs. I know how to use both Unity’s new and legacy input systems. This was just a quick implementation for something that wasn’t really all that necessary for the purposes of this project.

1) _checkCameraPos (boolean): Turn off Camera Transform check if it is not needed anymore.
2) _cameraPosOffest (Vector3): Use Camera offset for the purposes of viewing the tree. This value is updated in realtime if CheckCameraPos = true.

3) _cameraPivotOffsetFromGround (float): Used to offset from the GrowthPoint as the point at which the camera will orbitally rotate around.
4) _growOverTime (boolean): Toggle on to see the tree grow rather than instantly appear.

UI:

The UI is simplistic, but handles all relevant user feedback and functionality for the purposes of this project.



Current System in Use: My System
Press the reset button to test a different system or different settings.

Current time elapse while growing tree: 2.776281

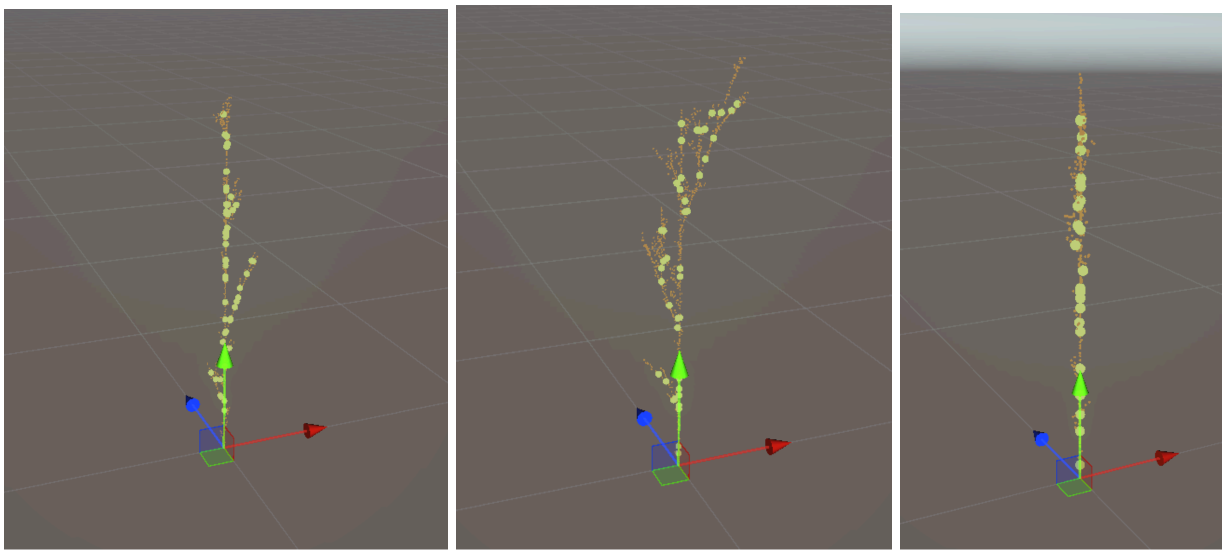
1) Reset button: This will rerun the L-system logic. Changes in the inspector will be reflected immediately.
2) If _growOverTime is true, a label will display how long the tree has been growing for. This isn't really important. It can be interesting if you are familiar with how the age parameter detailed above relates to the growth duration thresholds so that you can see during runtime how generations affect the shape of the tree.

Example custom L-system:

Axiom: F
Rules:

1) Target variable: F
2) Result (upon generating new generations):
    a) Sentence: F[+F]F[-F]F, Probability: 0.17, Age decrement rules: 01111111110
    b) Sentence: F[^L]F[vF]F, Probability: 0.17, Age decrement rules: 01121111110
    c) Sentence: F[+F]F, Probability: 0.17, Age decrement rules: 011110
    d) Sentence: F[-F]F, Probability: 0.16, Age decrement rules: 011110
    e) Sentence: F[^F]F, Probability: 0.17, Age decrement rules: 011110
    f) Sentence: F[vF]F, Probability: 0.16, Age decrement rules: 011110

Min Length: 0.005
Max Length: 0.02
Random Lengths: true
Angle: 20



First tree, Second tree (Reset pressed once), Third tree (Reset pressed a 2nd time)

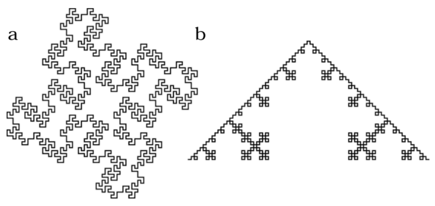
Main takeaway:

1) I took some inspiration from sources online for the shape of the sentences above. While you have the freedom to pick whatever sentences you want, there are definitely certain forms that work better than others.
2) There are more + and - (indicating roll rotations) than ^ and v (indicating yaw rotations). So the trees end up looking more flat than a fully 3D tree. If your final generation L-system has an equal number of rotations around each axis, your tree will appear more 3 dimensional.
3) There are ways to control the amount of branch growth before attempting to grow leaves. I didn't make these considerations, so you can see a lot of awkward leaf growth.

The inclusion of a definition for growing leaves (indicated by the character L) was primarily for example purposes and not for creating the most realistic tree possible.

4) Probabilities are hardcoded such that they must add up to 1. They don't need to be similar in their probabilities. This is tied to what I mentioned above about controlling branch growth. Rules like (F: FF, Age decrement rules: 01) are common for adding more obvious instances of branch elongation before applying other more interesting branch/growth rules.
5) Age decrement rules can also be controlled to make things more interesting. Rather than making every new generation progress immediately after the previous generations' growth, having an informed ruleset that includes greater age decrement amounts per subsequent generation can create interesting effects in your growth animation. I am applying a common simple example where leaves will grow on its branch only once that branch has fully grown indicated by the age decrement rule of 2 rather than 1.
6) I haven't explored this, but it is referenced in many of the derivations in The Algorithmic Beauty of Plants. L-systems can be used to generate interesting fractal patterns and geometric shapes beyond trees.

1.3. Turtle interpretation of strings 9

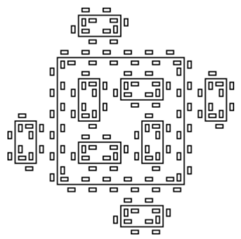


n = 2, $\delta = 90^\circ$
F-F-F-F
F $\rightarrow$ F+FF-FF-F-F+F+F

F-F-F+F+FF+FF-F

n = 4, $\delta = 90^\circ$
-F
F $\rightarrow$ F+F-F-F+F

Figure 1.7: Examples of Koch curves generated using L-systems: (a) Quadratic Koch island [95, page 52], (b) A quadratic modification of the snowflake curve [95, page 139]



n = 2, $\delta = 90^\circ$
F+F+F+F
F $\rightarrow$ F+f-FF+F+FF+Ff+FF-f+FF-F-FF-Ff-FFF
f $\rightarrow$ ffffff

Figure 1.8: Combination of islands and lakes [95, page 121]

Figure 1.8, The Algorithmic Beauty of Plants by Przemysław Prusinkiewicz and Aristid Lindenmayer, page 9 Chapter 1

This, in my opinion, justifies pursuing this approach for procedurally generating trees. There is so much going on under the hood that has something for everyone whether we are talking about mathematicians, biologists, artists, computer engineers, and, I'm sure, others I can't think of off the top of my head.

Conclusion:

I’ve said all I care to say. Have fun!