Procedurally Generated Trees

TNM084
Procedural Methods for Images

Ola Steen

February 10, 2019

Contents

1 Introduction

An L-system is a way to describe the structure and growth of plants. It was introduced by Astrid Lindenmayer in 1986 and it works by applying a set of production rules to an axiom string to model how a plant or organism grows. By designing production rules and an initial axiom where the rule can be applied, a tree like structure can be generated. This report covers how an L-system can be used to generate branching structure for a tree that can be drawn in OpenGL using simple geometry. [1]

2 Background

There are different variations on the regular L-system but the regular L-system can be described as:

- A set of symbols, V, that correspond to different actions. These symbols come in two forms: variables and constants, depending on whether or not they can be replaced.
- An initial axiom, ω, that contains symbols from V. This axiom describes the initial state of the L-system.
- And a set of production rules, P, that describes how the axiom will change. These production rules are essentially a set of if-statements that dictates what symbols a variable in the axiom should be replaced with.

The growth of the L-system is done by iterating over the symbols in the axiom and applying the rules to any variable in the axiom. This iteration is done a couple times resulting in a larger and more complex axiom. [2] A simple example of this is:

$V = \{a, b, c\}$ where a and b are variables and c is a constant.
$\omega = a$
$P =$
$a :\rightarrow bab$
$b :\rightarrow c$

Iterating over the axiom gives us the following results:

a
bab
$cbabc$
$ccbabcc$
$\dots$

2.1 Variations on L-system

In order to achieve more complex structure the L-system can be modified with stochasticity and parameters. In a stochastic L-system one variable can have multiple replacements and which one is used is determined using a discrete probability distribution. For example:

$a :\xrightarrow{0.25} ab$
$a :\xrightarrow{0.25} ba$
$a :\xrightarrow{0.50} bab$

In this example the there is a 25% chance that an a will be replaced by a ab, a 25% chance that an a will be replaced by a ba and a 50% chance that an a will be replaced by a bab.

Another variation of the L-system is one where symbols receives a parameter value. When each symbol in the axiom has an associated parameter we can extend the production rules to modify those parameters. An example of an L-system with parameters can be seen below.

$$\omega : a(1)$$
$$a(x) :\rightarrow a(x * 2)b(0)$$

In this example an a is replaced with an a with a parameter twice as big as that of the original a and a b with the parameter 0.

3 Method

In our implementation of an L-system we will use both stochasticity and parameters. This will give us a degree of randomness and the ability to propagate information across the tree as we calculate its structure. The full source code can be found at https://github.com/MrZtone/Tr-d-Trees.

3.1 Classes

The implementation has many classes. This section focuses on those relevant to the Tree and its structure and won't detail the classes used for drawing.

3.1.1 Symbol

This class corresponds to a symbol in the axiom string. It has a signifier, which is used when comparing symbols. It also has a pointer to a function that takes a float value as input and outputs a transformation matrix that will be used when drawing the tree.

```
class Symbol {

    private:
    char signifier;
    glm::mat4 (*function)(float);
    float parameter;

    public:
    Symbol(char sig, glm::mat4 (*fun)(float) , float p);
    Symbol(const Symbol& c);
    Symbol(char sig);
    char getSignifier();
    bool equals(const Symbol& C);
```

3.1.2 Rule

The rule class corresponds to one production rule. It has one symbol which is the value in the axiom that is to be replaced. It has a vector of potential replacements and a discrete probability distribution that is used to determine which of the available replacements will be used. This is all done in the class method getReplacement. Which receives a value from the axiom and outputs a vector of symbols to replace it with.

```
class rule {

    private:
    Symbol condition;
    std::vector<std::vector<Symbol>> replacements;
    std::discrete_distribution<int> distribution;

    std::random_device rd;
    std::mt19937 gen;

    public:
    rule(Symbol cond, std::vector<std::vector<Symbol>> rep, std::discrete_distribution<> dist);
    rule(rule && r);
    std::vector<Symbol> getReplacement(const Symbol& c);
```

3.1.3 LSystem

The LSystem class has two main properties. The axiom, which is a vector of symbols, and rules which is a vector of rules. It also has the method apply_rules() and apply_rules(int counter) that iterates over the axiom and applies the rules in accordance to its rules. It also has three functions that returns transformation matrices. These function are used to initialize the symbols and when drawing the tree.

```
class LSystem {

    private:
    std::vector<Symbol> axiom;
    std::vector<rule> rules;

    void apply_rules();

    public:
    LSystem();
    void apply_rules(int counter); // apply rules multiple times
    std::string getAxiom();

    //tree functions
    static glm::mat4 grow(float distance);
    static glm::mat4 split(float angle);
    static glm::mat4 rotate(float angle);
```

3.1.4 Tree

The tree class has three simple pieces of geometry that are used as building blocks for the entire tree:

- A cylinder that makes up most of the stem and the branches.
- A cone that is used when a branch ends.
- A polygon that vaguely resemble a leaf. It is also used when a branch ends.

It also has an LSystem, whose production rules are applied to its axiom five times when the tree is initialized and a draw function.

The draw function works by iterating over the symbols in the axiom and calling their associated functions in order to calculate transformation matrices that can be added to the MatrixStack object which acts as a sort of scenegraph. if it comes across as symbol that has a drawing action associated with it then it also draws a cylinder, a cone or a leaf depending on what action it is.

```
class Tree {

    private:
    LSystem L;
    Cylinder branch;
    Leaf leaf;
    Cone cone;

    public:
    Tree();
    void Draw(Shader shader, MatrixStack& MS);
```

3.2 L-system used to draw the tree

$$V = \{G(), S(), R(), [,], X\}$$

- $G()$: Grow the branch or stem. The distance of the growth depends on the provided parameter. This is a drawing action (cylinder).
- $S()$: Split the branch or stem. This is done by rotating the branch around the Z-axis. The angle of the rotation depends on the provided parameter.
- $R()$: Rotate the branch or stem around the Y-axis. The angle of the rotation depends on the provided parameter.
- [and]: [saves the current transformation matrix to the Scenegraph. That matrix is restored with].] signifies the end of a branch and is therefore a drawing action (cone and leaf).
- X: X doesn't correspond to any kind of transformation or drawing action. It is used to control how the system grows.

$\omega = X$

$P =$

$$X :\rightarrow G(0.5)[S(45.0)R(137.5)G(0.2)]X$$

$$G :\xrightarrow{0.43} G(0.2)R(137.5)[S(-45.0)R(-137.5)G(0.2)X][S(-45.0)R(-137.5)G(0.2)]G(0.2)$$

$$G :\xrightarrow{0.43} G(0.2)R(137.5)[S(45.0)R(137.5)G(0.2)X]G(0.2)$$

$$G :\xrightarrow{0.14} G(0.3)[X]$$

$$R(x) :\rightarrow R(x + 137.5)$$

4 Results

Figure 1: Three different trees generated using the L-system described in section 3.2.

5 Discussion

Although the production rules used to generate the trees creates results that look ok. They were created through experimentation and weren't chosen to mimic a specific type of tree. A more realistic tree structure could probably be achieved by someone who actually knows something about how trees tend to grow.

The drawing of the trees could also be improved. Instead of using transformation matrices to stitch together simple geometry into the shape of a tree. The axiom could instead be used to create a mesh of vertices that actually make up the surface of the tree. This would mean that the tree would only have to be built once when initialized instead of assembled each time it is drawn. It would also remove the need of a dedicated draw function for the tree since it at that point is just an ordinary mesh and could use the same draw function as all the other meshes.

References

[1] Wikipedia contributors. (2019, February 3). L-system. In Wikipedia, The Free Encyclopedia. Retrieved February 10, 2019, from https://en.wikipedia.org/w/index.php?title=L-system&oldid=881596984

[2] Charlie Hewitt. (2017 May 19). Procedural generation of tree models for use in computer graphics. Trinity Hall Cambridge. Retrieved December 6, 2018, from https://chewitt.me/Papers/CTH-Dissertation-2017.pdf