Procedural fractal plants generation

Filip Rynkiewicz
Institute of Information Technology, Lodz University of Technology, ul. Wolczanska 215, 90-924 Lodz, Poland
`173186@edu.p.lodz.pl`

Piotr Napieralski
Institute of Information Technology, Lodz University of Technology, ul. Wolczanska 215, 90-924 Lodz, Poland

Abstract

Since the beginning of computer era we are trying to imitate nature. The results of those attempts are mathematical equations describing weather, snow flakes, plant growth or influence of species in individual biomes. In computer graphics fractal structure are often used because they can be simple characterize in mathematics. Those forms are common in nature. Self-similar fractals are created in computer graphics using Lindenmayer Systems. This article was formed to analyse efficiency of this algorithm in modelling 3D tree triangle meshes for games. The general motivation of this topic is to produce way to procedural modelling and simplified edition of complicated 3D tree models. By combining L-systems with Bezier Curve there is a possibility to produce complicated 3D models. For example tree created with 468 branches having 87200 vertices and it is made in few seconds. In conclusion, time-consuming modelling in 3D programs can be replaced by solution described in this article.

1 Introduction

Using 3D modelling programs to create a complex 3D mesh of tree is very time consuming, because every branch has to be shaped separately. To speed up and automatize this process Lindenmayer System can be used. Since 1968, when Aristid Lindenmayer created grammatic to represent simple multicellular organism, technique has evolved. Now L-Systems are used to manage 3D models of plants[1, 3, 5, 14, 4], fire visualization[2], music[6] or even neural web[7]. Variaty os usage, flexibility and simplicity of this technique creates a possibility to extend it. Developers over years have used this technique to create software like L-studio[8], plugins to 3DsMax to animate plant growth[9] or in Houdini[11]. Most of the algorithms to create 3D tree mesh are generating static models, to change or move branch user must change parameters inside L-System, and the recalculate the whole model. To prevent unnecessary actions and calculation the *Bézier Curve* was added as the tree skeleton. Adding possibility of changing Bézier Points along Bézier Curve, user can change every branch only by moving it's handler and recalculating only this alternated branch.

After long years of plants observations the conclusion was brought, every plant can be treated as a very complex fractal. The most important feature of this structure is self-similarity. In mathematics, a self-similar object is exactly or approximately similar to a part of itself (i.e. the whole has the same shape as one or more of the parts)[10]. Because of complexity of plants as fractals the simplifications must be applied. To do this the L-System can be used. Mentioned above technique is based on rewriting rules, so on replacing sub-terms of a formula with other terms. All formulas and terms are represented as strings, and L-system is all about changing one string to another based on rules. Every system has to have starting word, called axiom, rules of productions and number of iterations, so how many times rules must be applied to word.

2 Procedural Systems

Creating data algorithmically, based on mathematical equations, is the core of Procedural System. This mean that program creates content on its own, or with limited or indirect user input. In case of L-systems there are few possibilities of creating procedural content using simple or more complicated L-systems.

2.1 OL System

Most basic L-System is context-free L-system, called OL-System, where there can be multiple replacement rules to any given value. In example:

$$A \longrightarrow \alpha \qquad (1)\{?\}$$

$$A \longrightarrow \beta \qquad (2)\{?\}$$

That means that A can be replaced with either α or β.
When there is one and only one replacement for the same letter in word, the system is called *DOL-system*, deterministic OL-system. Giving *axiom*(starting word) as b, rules : $b \rightarrow a$ and $a \rightarrow ab$ with 5 iterations simple DOL-system can be created. Example production of this system is shown at Figure1.
Because L-systems was inspired by nature, usage of it can be found in there. In example development of cyanobacteria *Anabaena Catenula* can be simulated by DOL-system.

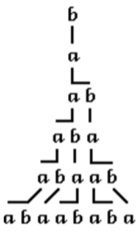


Figure 1. Example of DOL system[12]

⟨DOL⟩

2.2 Turtle interpretation of string

L-System algorithm generates characters, so it must be evaluated to create graphical content from it. Created word contains steering, drawing and omitted characters, they are represented at Figure1. The steering characters are the one which controls the behaviour of turtle. Drawn are those which simply adding a line between points. *Turtle graphics* is term where turtle have it's position, angle which describes the direction in which it is facing and any other parameters that user will use, but position and angle are mandatory.

Table 1. Steering characters

⟨tab⟩

sign	**discription**
+	Turn left by angle δ.
-	Turn right by angle δ
&	Pitch down by angle δ
ˆ	Pitch up by angle δ
$\backslash$	Roll left by angle δ
/	Roll right by angle δ
$\vert$	Turn around, by 180 °
[	Add branch
]	End branch
[A . . . Z]	Draw line between points

As it can be seen in the last column in Figure1 the steering character to draw line between points can be represented by multiple upper cases letters. User can change behaviour of L-system by manipulating those characters. Can be defined which one are representing draw line and which are just to create randomness of given L-system.

2.3 Bracket L-System

OL-systems always create one long continued line. To add branching like behaviour, new characters must be added. Using *[* and *]* the *branches* can be created. First character is responsible for starting the branch and next for closing it. Branch is simply new line in geometric interpretation of string created by L-system Figure2.

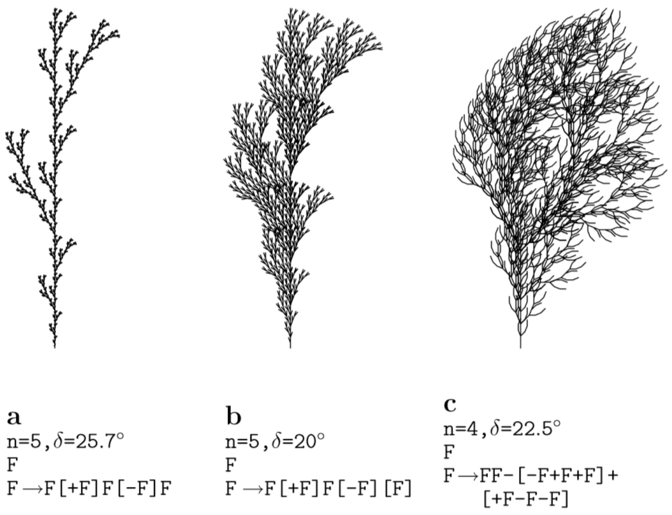


Figure 2. Example of Bracket L-system[12]

⟨branchingL⟩

2.4 Parametric L-System

To create more complicated models the Parametric L-System was created. Thanks to parameters L-system can change the values of variables, in example the steering angle of branch. When in DOL-System the sign + was strictly assigned to hard-coded value, in this approach the value can be changed in every iteration using *+(r)* where *r* is some kind of parameter. Every letter or string must have brackets immediately behind it and within them there can be multiple parameters, separated with comma,also mathematical expressions can be applied between parameters. Example:

$$C(x \cdot e, e \backslash 2, a+c, b-d, g) \quad (3)\{?\}$$

3 Bézier Curves

A B-spline is a generalization of the Bézier curve [13]. Let a vector known as the knot vector be defined

$$T = t_0, t_1, \ldots, t_m, \qquad (4)\,\{?\}$$

where **T** is a nondecreasing sequence with $t_i \in [0,1]$ and define control points $P_0, P_1, \ldots, P_n$. Define the degree as

$$p \equiv m - n - 1 \qquad (5)\,\{?\}$$

The "knots" $t_{p+1}, \ldots, t_{m-p-1}$ are called internal knots.

$$N_i^j(t) \begin{cases} 1 & if\, t_i \le t < t_{i+1}\, and\, t_i < t_{i+1} \\ 0 & N_i^j(t) = \frac{t-t_i}{t_{i+j}-t_i} N_i^{j-1}(t) + \frac{t_{i+j+1}-t}{t_{i+j+1}-t_{i+1}} N_{i+1}^{n-1}(t) \end{cases} \qquad (6)\,\{?\}$$

where $j = 1, 2, \ldots, p$. Then the curve is defined by

$$C(t) = \sum_{i=0}^{n} P_i N_i^p(t) \qquad (7)\,\{?\}$$

4 Methods

Unity Engine is Game Engine where developer can write application in C# using Mono. L-System was implemented with couple of simple structures. This is description of those:

LType	description of parameter, its definition as float and name of object
LFunction	definition of production and collection of parameters used in this production
LObject	object of L-System
PosRot	quaternion and position

Every *LObject* have list of *LFunction* and Dictionary *LType*, where *Key* is string name and *Value* is float. Class where are all methods that calculates L-System productions is *Interpreter*, where there are 2 functions *CreateTreeString and CreateBezierTree(LObject currentObject)*. First create word from rules and second creates tree skeleton based on this word.

Word is created by replacing one string by another using rules corresponded to LObject. Because in this implementation only Parametric L-System is used, all parameters must be evaluated in every iteration. There are two main cases. When parameters inside bracket have mathematical expression, and when they don't. Second event is simpler to evaluate because algorithm have to find corresponded value to string representation in Dictionary, and than change it for float value. It's performed only at the beginning of algorithm, and with every function. Thanks to this, algorithm does not need to change every parameter in every iteration, it's done only once.

$$!(vr)F(50)[\&(a)F(50)A]/(d2) \quad \longrightarrow \quad !(2.13)F(50)[\&(0.45)F(50)A]/(11.3) \tag{8}$$
{?}

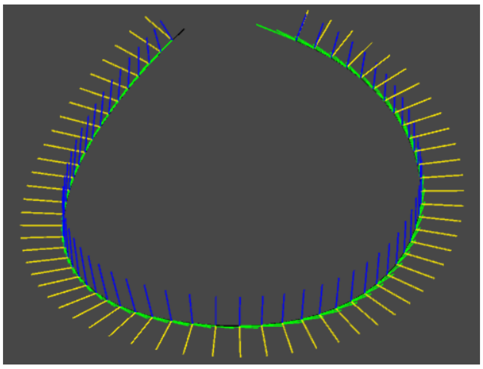
Parameters that have mathematical expression between them have another approach to evaluate them. At first parameters have to be changed to float values then expression must be specified and the last step is to interpret, calculate value as float and then return it as string value.

$$F(l) \rightarrow F(l \cdot lr) \tag{9}$$
{?}

In this example the *l* and *lr* parameters will have to be found in Dictionary, then the · must be interpreted as multiplication sign, and the last step is to return result as string representation of float, so if *l = 6* and *lr = 0.32* the result of this rule would be *F(1.92)*.

Second part of algorithm is to generate tree skeleton, based on created before word. String may consist of several thousand characters, so analysing it is the most time consuming part of this algorithm. Every Bézier Curve has it's parameters such as *normal, bi normal and tangent vector*, corresponded to *X,Y and Z position*. Every branch is separate Bézier Curve, which consist of Bézier Points. Those structures can be moved, deleted or added. That operations are based on steering characters in the word. Visualisation of those parameters are at Figure3a. Tree skeleton created based on word is at Figure3b, after modification Figure3c. Figure3d present created mesh placed on points of skeleton.

Last step is to generate mesh based on points of skeleton. Every point on Bézier Curve is center of one 3D cylindrical mesh. Normal, binormal and tangend are used in this part to calculate properly bending angles. All calculation are based on Quaternions rotations, to ensure appropriate angles, and because that how the Unity is handling the angles.

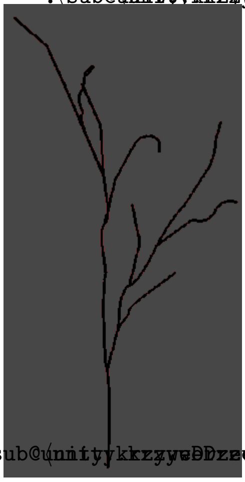


(a) Visualisation of direction(green), normal and binormal (yellow) of splain.

?⟨sub@unitykrzyweDrzewo⟩?

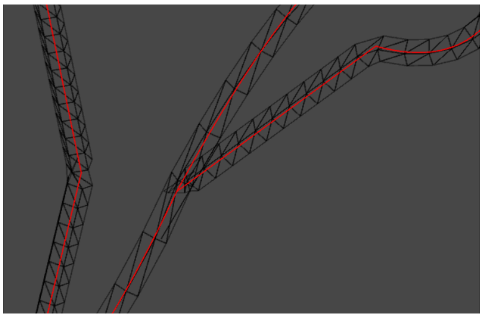
drawDirection⟩?

(b) Tree skeleton.



?⟨sub@unitykrzyweDrzewoSiatka⟩?

(c) Tree skeleton after modyfication.



(d) Mesh on skeleton after modyfication.

krzyweDrzewoMod⟩?

Figure 3. ddd

ity.direction⟩?

5 Results

For each of the example the same values was used. Five segments on the side surface and five points on spline, it's mean that every curve was divided on five equal parts. Each result of implemented algorithm was divided into two pictures, screenshot of model generated by algorithm and model after changes. Changes was made by transforming Bézier Points on corresponded to branch Bézier Curve. First all the example and it's description was shown, then the pictures. Description of every example System contains rules, axiom and number of iteration used within it.

Example 1

In first example for 5 iterations the number of branches is 468, word forming tree have 46086 chars and number of vertices in model is 87200.

This example is modification of simple fractal plant described in [12]. Every branch is directed at 25 ° relative to parent branch. As it can be seen at the Figure4a, the model created by algorithm is complex, using only two rule and one variable. Model after changes have been shown at Figure4b. Every branch in tree have been altered according to user actions.

Rules:

$$F(l) \to F(l \cdot 2) \tag{10}\{?\}$$

$$X(l) \to F(l)[/(r)X(l)]F(l)[\backslash(r)X(l)-(r)X(l)]F(l)[\backslash(r)X(l)+(r)X(l)] \tag{11}\{?\}$$

Axiom:

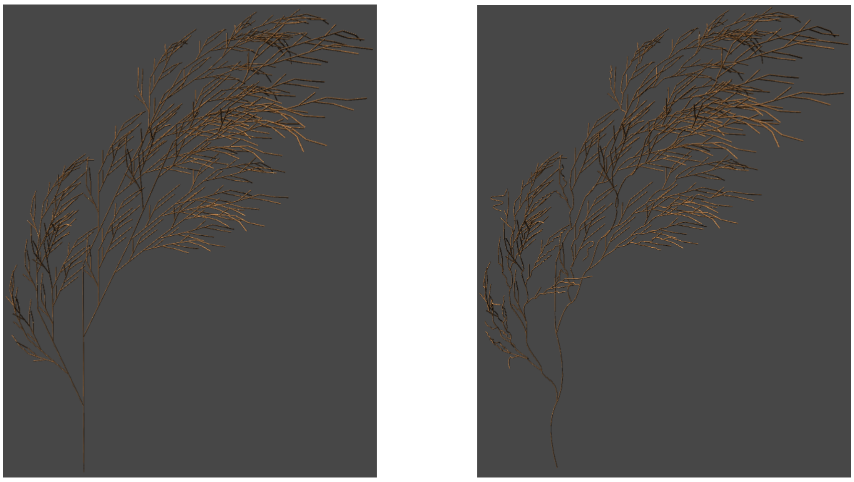
X(1)

Number of iterations:

5

Variables:

$$r \to 25 \tag{12}\{?\}$$



(a) Mesh without modification.
Own source.

(b) Mesh after modification.
Own source.

zyklad1siatka⟩? ?⟨sub@przyklad1siatkaMOD⟩?

Figure 4. Comparison of meshes for example 1.

?⟨przyklad1⟩?

Example 2

Next example have 171 branches, world forming tree have 24918 chars and 73700 vertices in model.

In this case the custom and longer rules have been used to make model looks like non-deterministic and more chaotic. Model before changes was shown at Figure5a, and after changes at Figure5b. In first example every variable had exactly one parameter without any mathematical operations, except $F(l \cdot 2)$. But in the second every operation on parameter had to be evaluated and changed to float value, so the time of the calculation had increased significantly. Thanks to usage of multiple parameters and operation between them user can create more complicated structures, using less rules, but for the price of time for parameter evaluations.
Rules:

$$F(l) \to F(l \cdot 2) \tag{13}$${?}

$$X(l,w) \to F(w \cdot l)[/(r \cdot l)X(l,w) - (r)X(w,w)] + (r)F(l)[\backslash(l)X(l,w) + (r)X(l,w)] - (r)F(w) \tag{14}$${?}

Axiom:

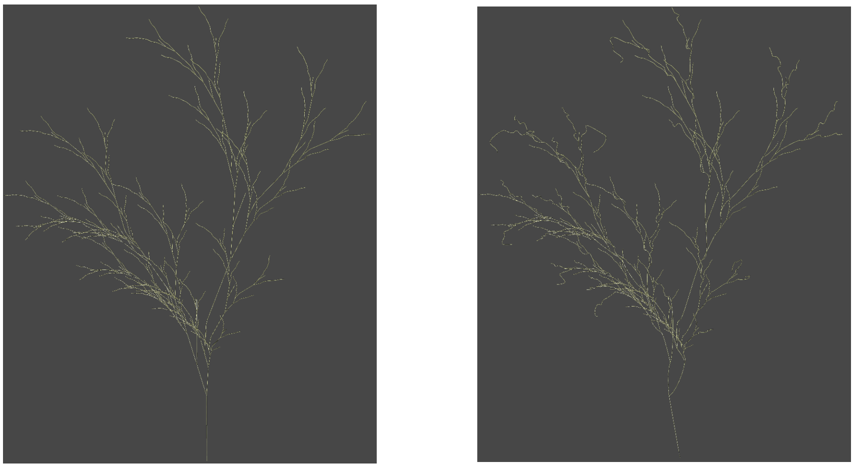
X(1)

Number of iterations:

5

Variables:

$$r \to 17.2 \qquad (15)\{\mathbf{?}\}$$



(a) Mesh withouth modification. Own source.

(b) Mesh after modification. Own source.

zyklad2siatka⟩? ?⟨sub@przyklad2siatkaMOD⟩?

Figure 5. Comparison of meshes for example 2.

?⟨przyklad2⟩?

Example 3

Last example have 214 branches, world world forming tree have 24918 chars and 73700 vertices in model. Figure6a presents unchanged model of last example, Figure6b same model after changes. Creating more complicated rule, based on those in example 2, and adding two more simple rules the model in example 3 is more non-deterministic and tree-like than other examples. Rules:

$$F(l) \rightarrow F(l\cdot 2) \qquad (16)\{?\}$$

$$\begin{aligned} X(l,w) \rightarrow F(w\cdot l)[/(r\cdot l)X(l,w) - (r)X(w,w)C(l)] \\ + (r)F(l)[G(l) \setminus (l)X(l,w) + (r)X(l,w)] - (r)F(w) - (r)F(w) \end{aligned}$$

$$G(l) \rightarrow F(l/5)[X(l,l) + (r\cdot k)] \qquad (17)\{?\}$$

$$C(l) \rightarrow F(l/5)[X(l,l)/(r\cdot k)] \qquad (18)\{?\}$$

Axiom:

X(1,2)

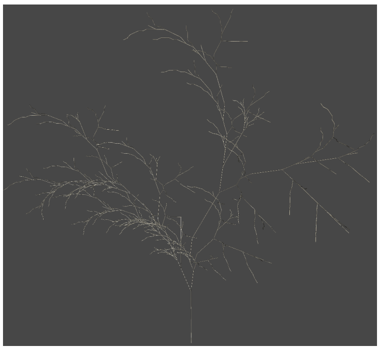
Number of iterations:

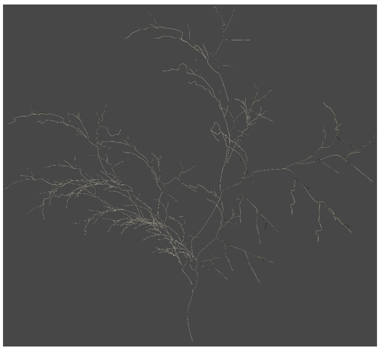
5

Variables:

$$r \rightarrow 23.5 \qquad (19)\{?\}$$

$$k \rightarrow 0.707 \qquad (20)\{?\}$$


(a) Mesh without modification.
Own source.


(b) Mesh after modification.
Own source.

?⟨przyklad3siatka⟩? ?⟨sub@przyklad3siatkaMOD⟩?

Figure 6. Comparison of meshes for example 3.

?⟨przyklad3⟩?

6 Conclusion

By using technique described before the user can simply generate tree-like model in short time, small amount of vertices, and with ability to change every point of model using Bézier handles. Creating model with 300 branches using normal 3d software would be very time consuming.We can move our work to machine and automatize process of modelling, using simple parametrized mathematical equations, without even knowing how the modelling software works. And we can do this inside game engine, without other programs.

To evaluate all string into float values the simple parser have been created. Parser created with this implementation is simple, but creating possibility to evaluate more than 3 parameters, and operations between them, using approaches from this implementation would be very difficult. To ensure that every parameter would be evaluated properly whole implementation will be rewritten into new *Boost Spirit X3* C++ library, and created as *Dynamic-Link Library*. Unity,Unreal Engine and others game engines have ability to write plugins based on *DLL* written in **C++**, so every engine could have their own implementation of this algorithm.

References

Herbaceus [1] Prusinkiewicz P., Lindenmayer A.,Hanan J., *Development models of herbaceous plants for computer imagery purposes*, SIGGRAPH Comput. Graph., New York, 1998

Sim [2] Zaniewski T., Bangay S., *Simulation and visualization of fire using extended lindenmayer systems*, AFRIGRAPH '03, Cape Town, 2003

WebTree [3] Qi H.,Qiu R.,Jia J., *L-system based interactive and lightweight web3D tree modeling*, VRCAI '11, Hong Kong, 2011

EnviroTree [4] Zhuming L., and King S., *Simulating tree growth based on internal and environmental factors*, GRAPHITE '05, Dunedin, 2005

Spray [5] Hanan J., Renton M., Yorston E., *Simulating and visualising spray deposition on plant canopies*, GRAPHITE '03, Melbourne, 2003

MLSystem [6] Manousakis S., *Musical L-Systems*, The Royal Conservatory,Hague, 2006

LBrain [7] Palmer M., *Evolved neurogenesis and synaptogenesis for robotic control: the L-brain model*, GECCO '11, Dublin, 2011

L-Studio [8] L-studio, `http://algorithmicbotany.org/virtual_laboratory/`

ABartniak [9] Bartniak A., *3D animation of plant development using L-systems*, M.Sc. thesis, Lodz, 2013

SelfSimi [10] Mandelbrot, B., *How long is the coast of Britain? Statistical self-similarity and fractional dimension*, Science, vol. 156, no. 3775, 1967, pp. 636–638. http://www.jstor.org/stable/1721427.

LHoud [11] Houdini Engine L-System, `https://www.sidefx.com/docs/houdini/nodes/sop/lsystem`

prusinABOP [12] Prusinkiewicz P., Lindermayer A., *The Algorithmic Beauty of Plants*, Springer , 2004

BSpline [13] Bézier curve, `http://mathworld.wolfram.com/B-Spline.html`

wlosy [14] Fuhrer M. Hairs, *Textures, and Shades: Improving the Realism of Plant Models Generated with L-systems*, M.Sc. thesis, University of Calgary, 2005

?LMat**?** [15] Rozenberg G., Salomaa A. *The mathematical theory of L systems*, Academic Press, New York, 1980