3D Tree Generation using LSystems in Unreal Engine 4

Maths and Graphics for Computer Games
Lecturer: Prof Frederic Fol Leymarie

Witold Gawłowski
MSc in Computer Games Entertainment

November 2016

Abstract

This document is a report from the process of creating a 3D tree generation application using L-Systems in the framework of choice. The application was to accept several text files with generation rules and provide ways of interaction for generated content. Results of work are demonstrated in short youtube video:

`http://www.youtube.com/watch?v=ieAfzjdyIcM`

1 Approach

I have been considering a number of tools to approach this task: *Octet*, *Unity3D* and *Unreal Engine*. *Octet* was an opportunity for learning *OpenGL* API. Using *Unity3D* would be easier as I already know it and therefore I could have focused on some bonus features. Finally there was *Unreal Engine* at which I am a beginner. Knowing that it is widely used in the industry and is capable of producing stunning visual effects I have decided to learn it and create use it for the project.

To keep the code separated from "the framework" as much as possible, my project uses pure *c++* to finalize the task. Only integration with the editor is via overriding the Pawn class.

2 Feature Overview

The code generates 3D, low-poly trees using L-Systems. For each tree segment a cuboid with decagonal base is created. The basis are skewed and rotated to smoothly merge with each other using *Unreal*'s *SplineMesh*s and *SplineComponent*s.

Six sets of rules from the assignment specification as well as two additional ones[1] were used to create eight input files. It is possible to switch between input files during generation using keys **1-8**. Each file consists of four separate sections: *variables*, *start*, *roll angle* and *rules*. Following rule symbols are used to describe the L-system turtle's actions:

F	+	-	&	^	[	]
go worward	turn right	turn left	pitch down	pitch up	save state	restore state

To give an example, I provide a file generating tree, that is displayed in the project solution by default:

```
variables:
F
start:
F
roll angle:
25
rules:
F -> F[^+&F]&F^[^-^F]F
```

The path to the specified file is displayed on the screen after pressing corespondng button.

Is is possible to adjust four parameters that influence the generation. Parameter needs first to be selected: *roll angle* is selected with **R** key, *pitch angle* with **T**, *length multiplier* with **U** and *width multiplier* with **Y**. To adjust selected parameter one then needs to press **up arrow** or **down arrow**. The number of steps in tree generation is changed with **left arrow** and **right arrow**.

3 Code structure

The *c++* part of the code is linked to *Unreal* by the TreePawn class. To simplify project class architecture, It handles multiple tasks: input files, player input, camera movement. It also holds reference to the Tree class. Finally, it is responsible for generating tree's string representation. Tree's core functionality is to "walk" the string and generate points that form the skeleton of generated structure. The points are divided into subsets and fed into Branch class instances. Tree is consisting of an array of Branches. Each instance of a Branch class has a Draw method that is responsible for generating meshes for a elements, each defined by neighbouring points.

4 Results

Enclosed images depict reconstruction of the L-System trees from assignment specification. Rule sets for specific trees are provided below each picture. Two last pictures depict two additional rule sets. Speed of generation of trees of complexity 7 is ranging between 1 and 3 seconds. It is about an order of magnitude slower comparing to on-line generators. This is explainable by the fact that most of them are generating segments as squares, not meshes.

5 Conclusion

The demo of the project is available on youtube:

`http://www.youtube.com/watch?v=ieAfzjdyIcM`

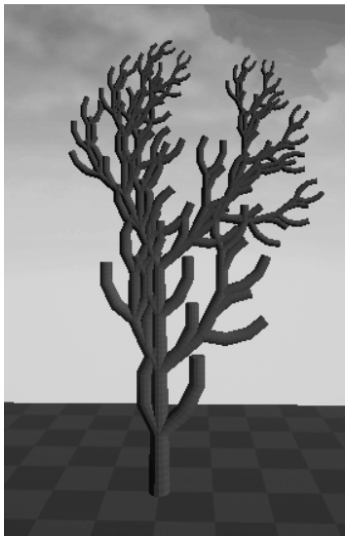
also the repository with the code source is available on github:

`https://github.com/witold-gawlowski/LUnreal`

Overall *Unreal engine* posed multiple challenges. The engine is complex and documentation is spread across forums, official docs and youtube tutorials. Therefore its frequently hard to find information and code examples. Also Unreal engines is created for big projects and therefore compilation

F −> F[^+&F]&F^[^−^F]F

F −> F[^+F]F[−F][&F]



F −> FF−[^−F+F+F]+[+F−&F−F]

starting symbol: X
F −> FF
X −> F[+X]F[−X]+&X^

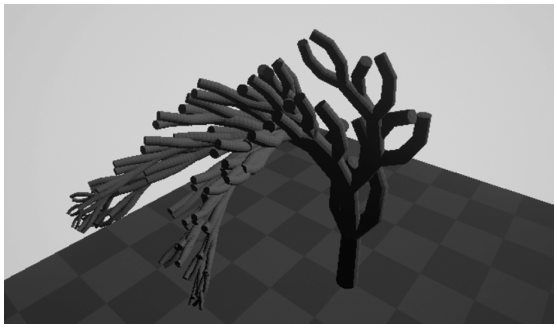
starting symbol: X
F −> FF
X −> F[++X][+X][−X][−−X]F&X^

starting symbol: X
F −> FF
X −> F−[[&X^]+^X&]+F[+F^X&]−X

times for the simple boilerplate code takes as much as 6-10 seconds after optimization.

On the other hand, Unreal Engine provides beautiful rendering out-of-the box. Also editor's GUI was highly intuitive for me. Overall, the process of learning the engine and creating a tree generation system in Unreal was a great experince.

starting symbol: F
F −> FF
X −> F[+X]F[−X]+&Xˆ



starting symbol: F
F −> FF
X −> F[++X][+X][−X][−−X]F&Xˆ

References

[1] https://github.com/abiusx/L3D.