Goldsmiths University of London

Spirograph

Coursework 2

MSc in Computer Games and Entertainment
Mathematics and Graphics for Computer Games 1

Andrea Castegnaro, Aldo Curtis

26 January 2014

Abstract

This document present the result for the second assignment of the Maths and Graphics course.
Spirograph are geometric curves based on recursive functions and a set of parameters that control the output shape. The find very interesting application in art and design. Our proposal has been creating a fast tool to create shapes providing the user the possibility to have total control on the creation by changing function type, line drawing parameters and colours. Our best effort has been thinking on using math techniques to have a smooth and fast drawing.

Contents

List of Figures

Chapter 1

Spirograph

1.1 Introduction

Spirograph was originally a geometric drawing toy that produces mathematical roulette curves of the variety technically known as hypotrochoids and epitrochoids. It is possible also to has custom function for this purpose as long as they agree with the current rules for a spirograph. In this chapter we will discuss the general math theory behind spirograph and we will describe also how to customize the function. The application has been developed using Andy Thomason's Octet Framework.

1.2 Mathematical Basis

A Spirograph is formed by rolling a circle inside or outside of another circle. The inner circle has control points that describes the curve. If the radius of fixed circle is R, the radius of moving circle is r, and the offset of the inner point in the moving circle is 0. The equation for the having a inner circle or the so called Hypotrochoid is the following:

$$x(\theta) = (R - r)\cos(\theta) + d\cos(\frac{R - r}{r}\theta) \tag{1.1}$$

$$x(\theta) = (R - r)\sin(\theta) - d\sin(\frac{R - r}{r}\theta) \tag{1.2}$$

$$\tag{1.3}$$

And in the same way we have the equation for the Epitrochoid or the outer circle such as:

$$x(\theta) = (R + r)\cos(\theta) - d\cos(\frac{R + r}{r}\theta) \tag{1.4}$$

$$x(\theta) = (R + r)\sin(\theta) - d\sin(\frac{R + r}{r}\theta) \tag{1.5}$$

And by combining the two of them we cna have nice configuration such as:

$$x(\theta) = (R - r)\cos(\theta) + r\cos(\frac{R - r}{r}\theta) \tag{1.6}$$

$$x(\theta) = (R - r)\sin(\theta) - r\sin(\frac{R - r}{r}\theta) \tag{1.7}$$

We added also a custom function with 5 parameters which gave us nice results. The equation is the following:

$$x(\theta) = \cos(a\theta) - (\cos^c(b\theta)) \tag{1.8}$$

$$x(\theta) = \sin(d\theta) - (\sin^f(e\theta)) \tag{1.9}$$

1.3 Implementation Idea

Our purpose has been creating a fast general tool for creating more than one graph in real time and to use this we focused on optimizing three aspect:

- computation time for calculating the points of the curve
- optimization of resolution step using de Casteljau subdivision
- fast GPU rendering using Bezier interpolation

So in order to create a fast drawing tool we had to achieve a fast computational time for the points on the curve which will be drawn in the render step (using a custom shader) using an interpolation method.

1.3.1 Fast subdivision

To achieve the first two points we decide to use a big initial resolution for calculating the points given by the function and we increment the step size only when needed. In order to do we took inspiration from the Casteljau subdivision.

Given a big initial resolution we calculate our point along the curve and using a variant of de Casteljau algorithm we calculate the curvature to see if we need more precision to describe the curve. Given a set of four following point we calculate the area of the triangles given by initial points and interpolated If the product of the areas is over a certain precision we end up using the interpolated point to evaluate the area again. The result of this method can be found in Fig 1.1

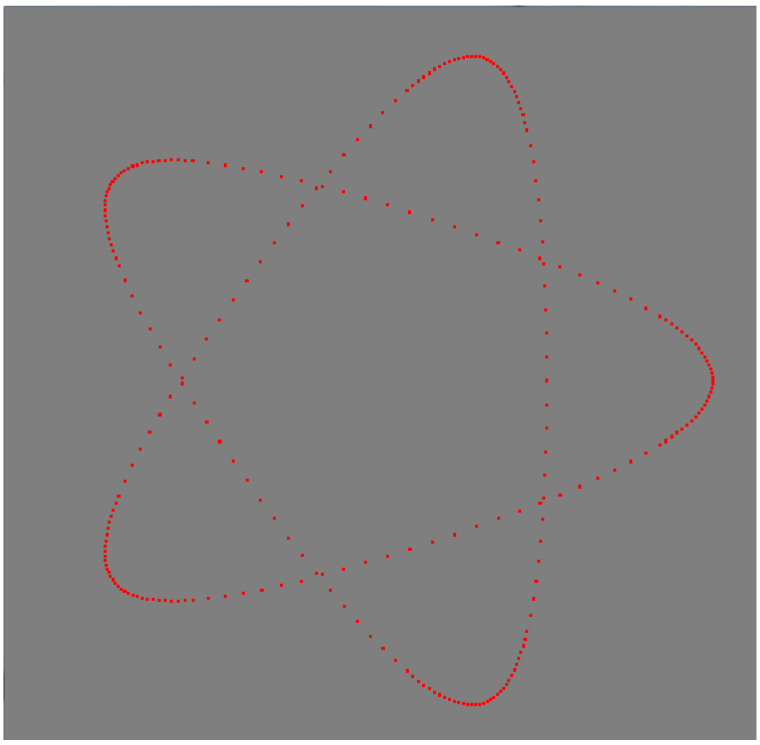


Figure 1.1: Effect of the subdivision algorithm implemented

1.3.2 Bezier interpolation

A Bezier curve is a parametric curve frequently used in computer graphics and related fields. It is used for smooth path calculation and it assure at least a C^0 continuity.

We used this method because we wanted to achieve very good looking result and we cannot use LERP method since we are not ending with simple linear motions.

The general equation for the Bezier curve is the following

$$B^n(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-1} s^i p_i \tag{1.10}$$

where $0 <= t <= 1$ is a parameter.

We used a third order Bezier curve to interpolate the points since it is not recommended to use a greater one.

We implement a custom feature for testing our Bezier result which consist of having a real time Bezier drawing system. A screenshot of the tool can be found in Fig. 1.2

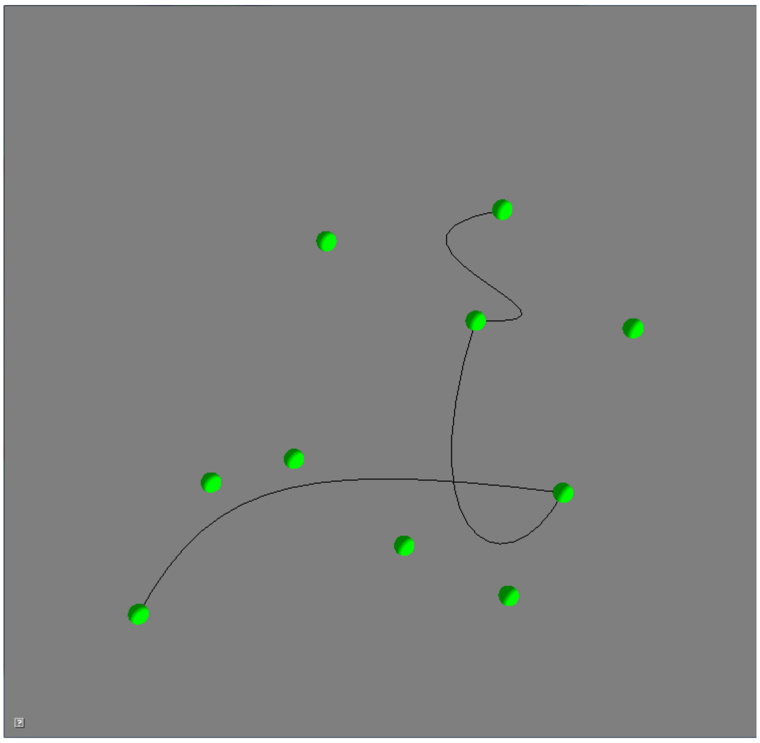
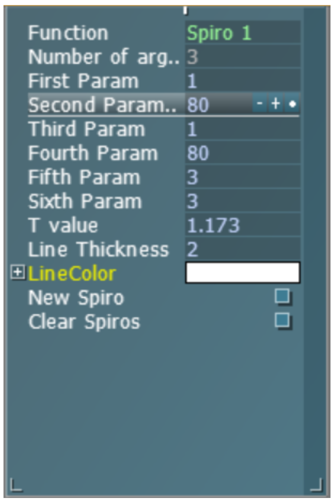


FIGURE 1.2: Effect of the subdivision algorithm implemented

1.3.3 UI interface

In order to customize the drawing real time we added a user interface to our project that had the following characteristics:

- Changing function equation
- T value parameter for the drawing
- Line thickness
- Line Color
- Add new spirograph
- Clear last spirograph



Function	Spiro 1
Number of arg..	3
First Param	1
Second Param..	80
Third Param	1
Fourth Param	80
Fifth Param	3
Sixth Param	3
T value	1.173
Line Thickness	2
LineColor	
New Spiro	☐
Clear Spiros	☐

Figure 1.3: User Interface screenshot

The user interface can be seen in figure 1.3

We slightly go off from our initial idea since the target for this project was understanding the math and graphics technique to draw the lines, so we skipped the shader interpolation but we still put all the feature we want for the algorithm meant to be implemented.

1.4 Implementation

The actual implementation is schematized in figure 1.4

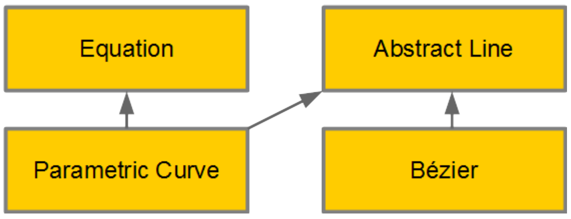
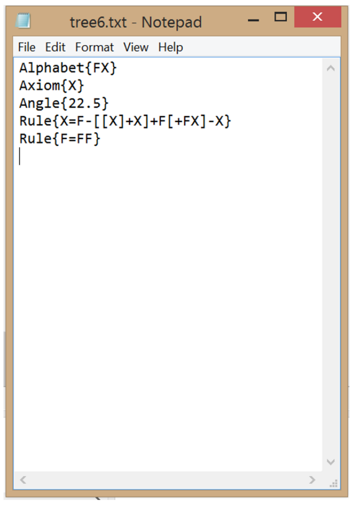


FIGURE 1.4: Schematic representation of the classes used for the implementation

The text reader class uses the *STD ifstream* library to access the file. The Octet Framework has a *getUrl* function in *appUtils* that does the same job in a faster way, but there are two major reason for choosing the first one: the files are quite small so practically there is no sensible difference, computationally speaking; the *ifstream* let you to read the file line by line without having to take care of *EOF* character or having to parse the information *char* by *char* thus incrementing the simplicity and readability of the code. This class is handled by the L-System manager which instantiate it once during the application initialization. To populate the *TreeContext* the manager (L-System) need to pass it to the *TextReaader* as a reference when calling the method to open the file. An example information for the input file can be found in Fig. 1.5. In the end, given a string path this class open the file related to the path, read it and at the same time populate the *TreeContext*, as soon as this operation is finished it closes the file.

1.4.1 Tree Context class

The context class is the one responsible of having all the information about the current L-System. It also calculate the next step of the L-System evolution given the previous one. It uses Octet Framework containers to store the alphabet, the axiom, the angle and a list of Rules. The alphabet is a set of letters used to describe the L-System behaviour, the rules how the L-System evolves in each iteration. Detail of how the file look like can be found in Fig. . The *Tree Context* has also collection that tracks any new generated iteration. In this way when downgrading/upgrading the iteration step to an already calculated iteration no additional computation has to be performed. When a new L-System is being fetched from a text file a *Reset* clean all the information of the previous system. So there is only one instance of this class handled by L-System.



```
Alphabet{FX}
Axiom{X}
Angle{22.5}
Rule{X=F-[[X]+X]+F[+FX]-X}
Rule{F=FF}
```

FIGURE 1.5: Example of input file parsed by the TextReader class

1.4.2 Tree Renderer class

This class is responsible or the actual draw of the tree. It contains a mesh instance which is used to draw any change on the current L-System. Basically the drawing operation consist of two different operations: getting the current evolution and for each symbol decide what operation needs to be done. To open/closing L-System branches a stack of 4x4 matrix stacks is used. Each of this matrix represent a model to the world position and rotation. Every time a new branch has to be created a copy of the last matrix is put in the structure. The stack is used to create a set of vertices which represent the center of the prism used to draw the L-Systems. Given the center to calculate the vertices for the face a simple trigonometric is applied [1]. Indeed the center is considered as the center of a circle on the XZ plane (Octet Framework is right handed with y-axis pointing upwards). To connect the point between the two faces six triangles are then created indicating to the gl resource which is the ordering for connecting the vertices.

1.5 Drawing result

In this section the drawing result for each of the L-System are showed. The last two L-System has been taken from [2].

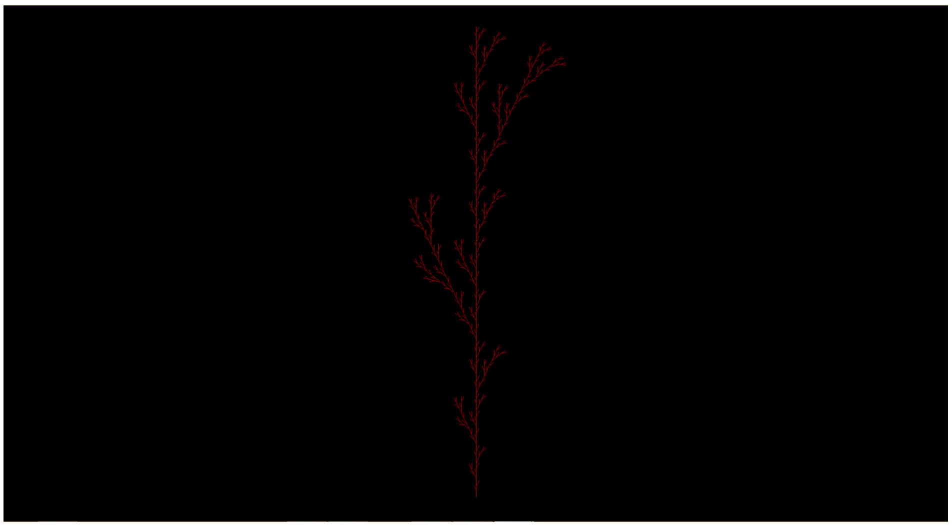


FIGURE 1.6: Example n 1 for L-System tree using just lines.

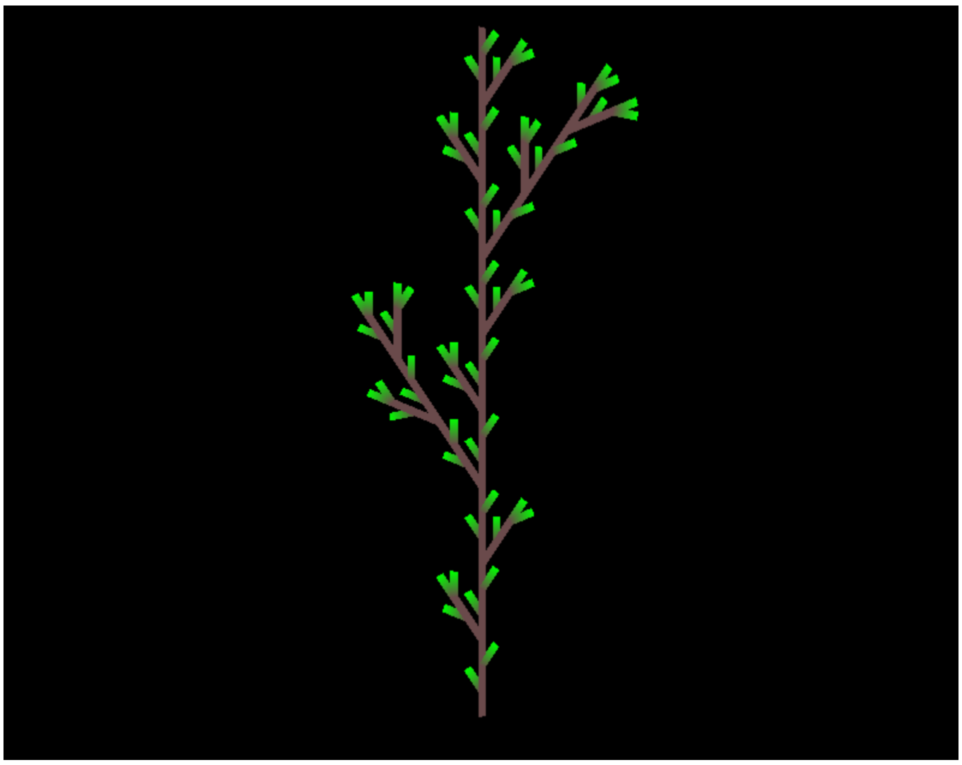


FIGURE 1.7: Example n 1 for L-System tree using prism structure to draw the mesh.

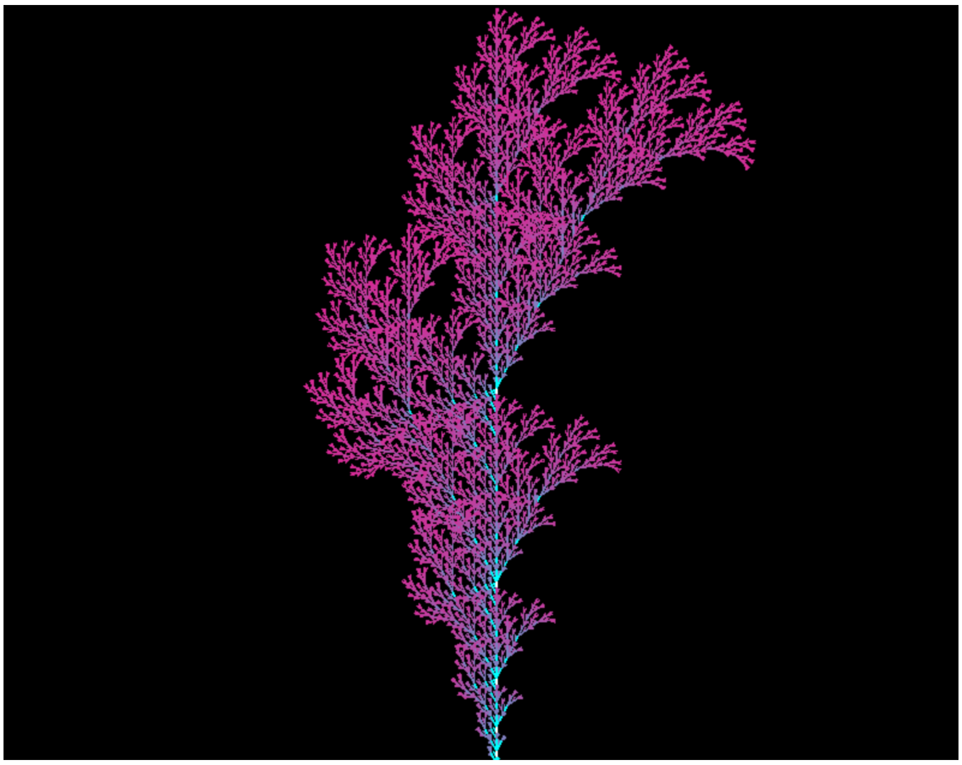


Figure 1.8: Example n 2 for L-System tree using prism structure to draw the mesh and a simple colour shader.

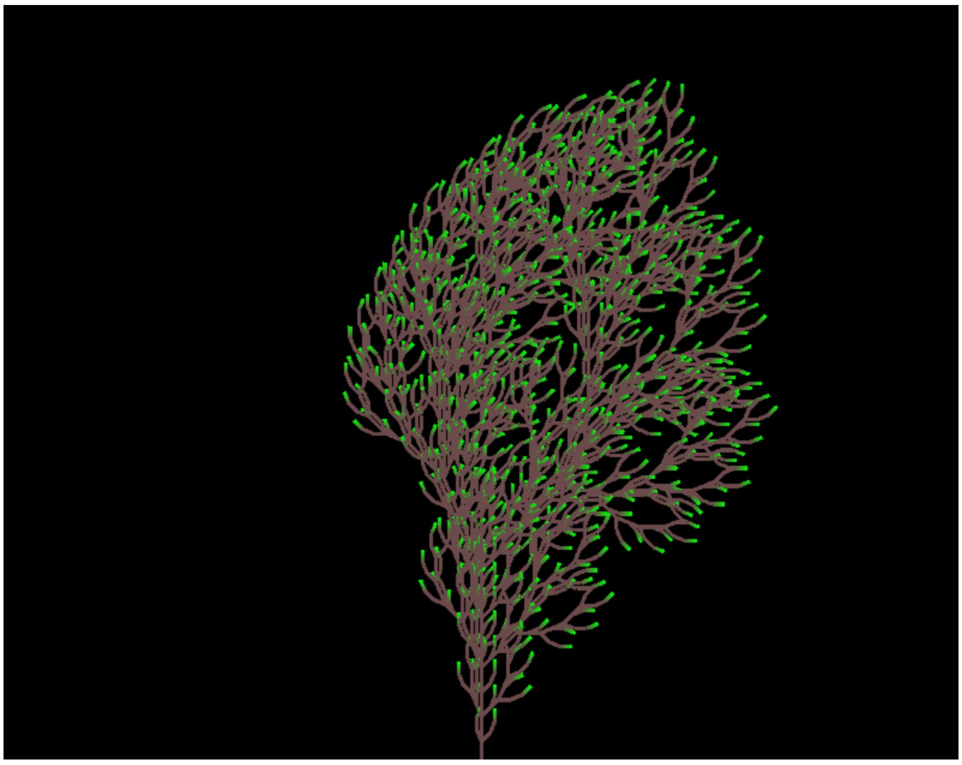


Figure 1.9: Example n 3 for L-System tree using prism structure to draw the mesh.

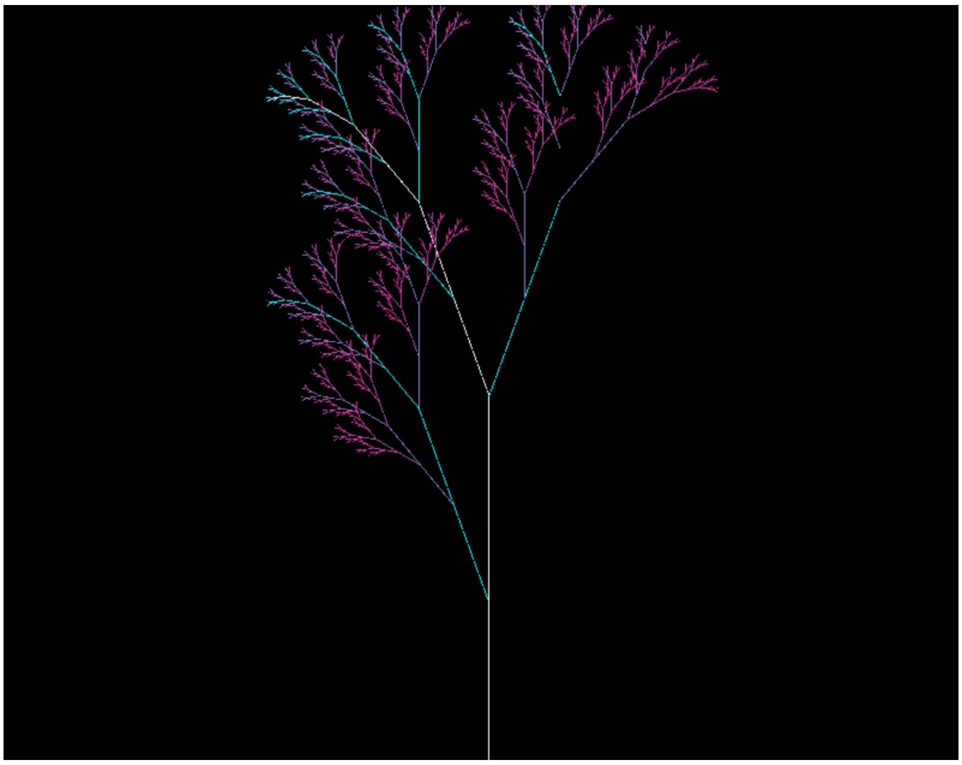


Figure 1.10: Example n 4 for L-System tree using prism structure to draw the mesh and a simple colour shader.

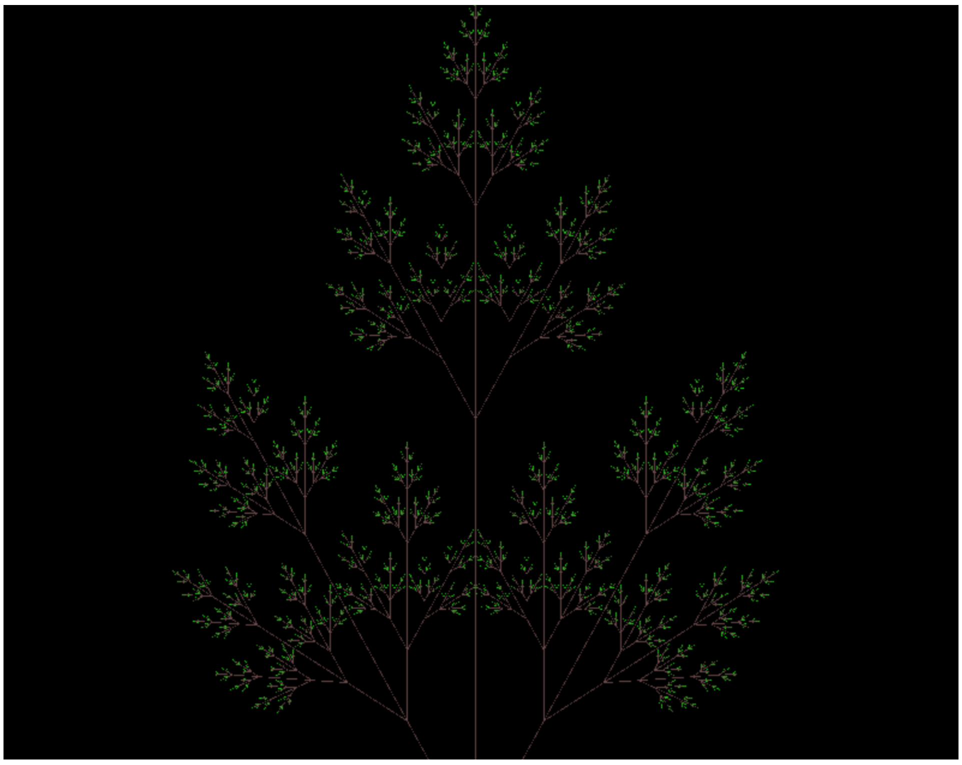


Figure 1.11: Example n 5 for L-System tree using prism structure to draw the mesh.

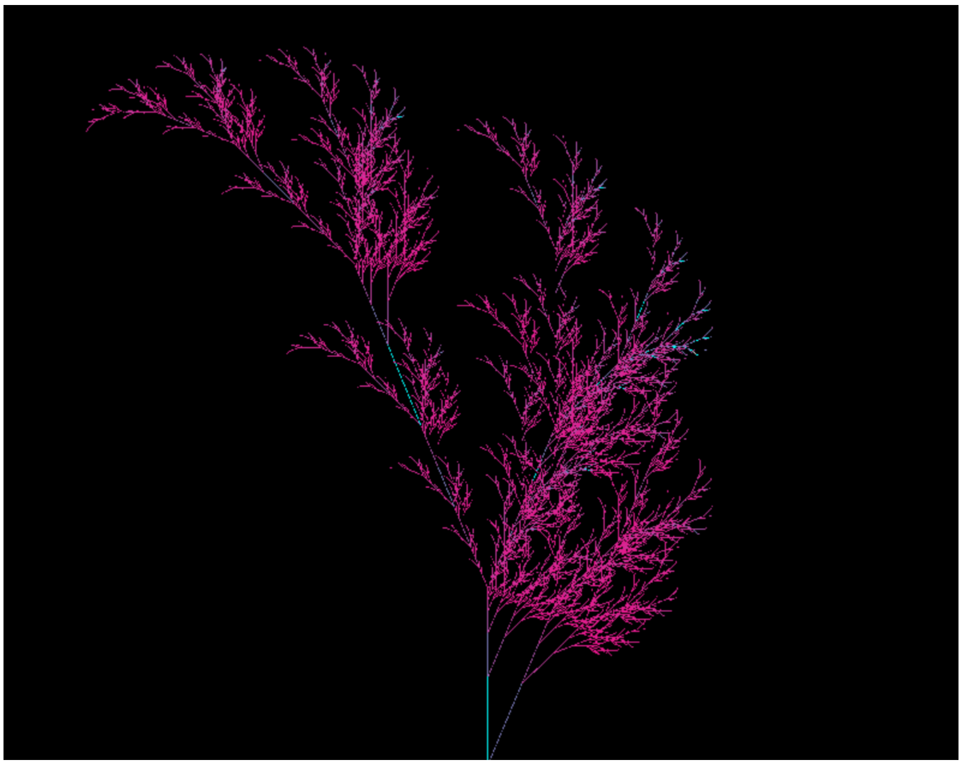


FIGURE 1.12: Example n 6 for L-System tree using prism structure to draw the mesh and a simple colour shader.

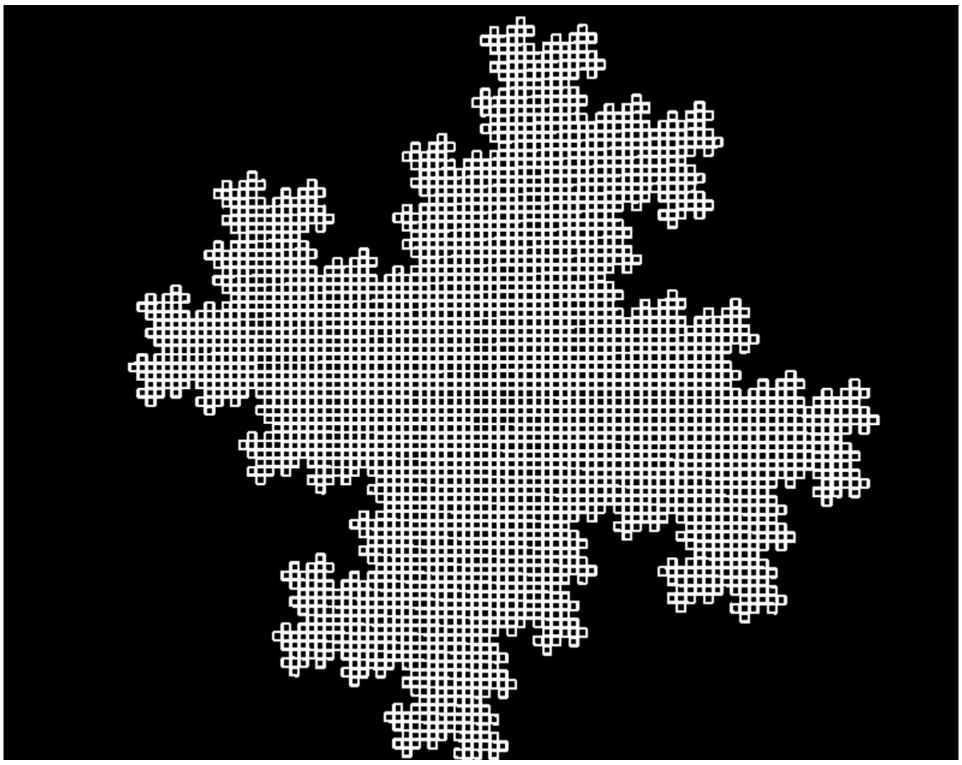


FIGURE 1.13: Example n 7 for L-System.

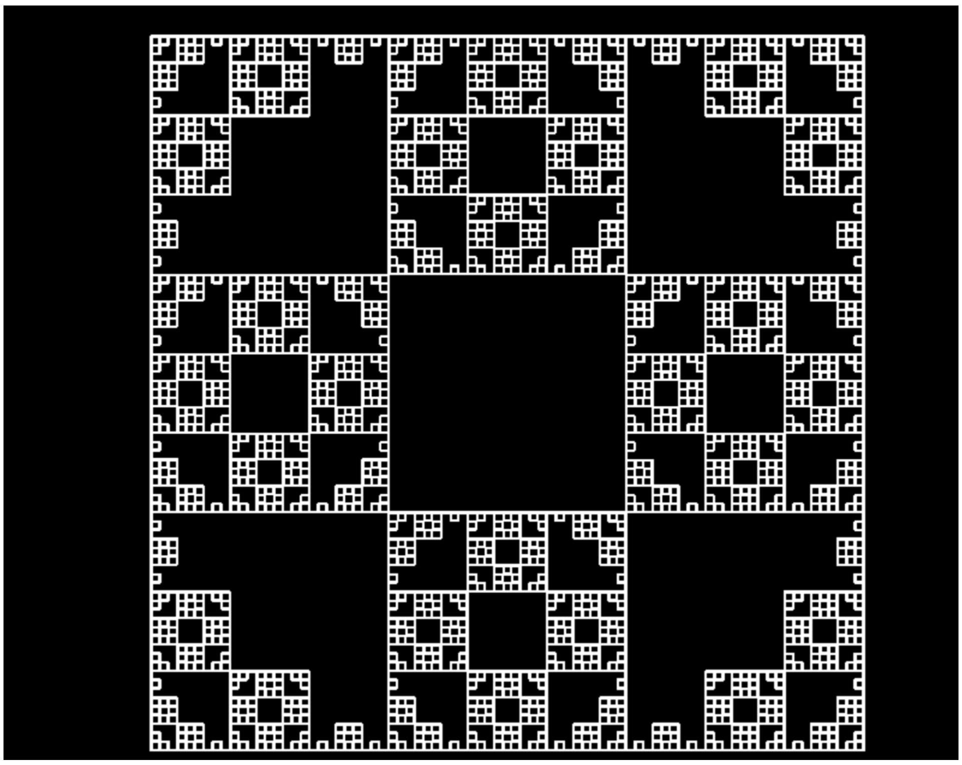


Figure 1.14: Example n 8 for L-System: box fractal

Bibliography

[1] Dunn and Parberry. 3d math primer for graphics and game development.

[2] Przemyslaw Prusinkiewicz. The algorithmic beauty of plants.