# Physics for Artists
Fall 2012

Valerio Paolucci
Michael Leitch

# The Mathematical Beauty of Nature

Fractal geometry, and L-systems in the Visual Effects Production

"Mathematics, rightly viewed, possesses not only truth, but supreme beauty — a beauty cold and austere, like that of sculpture, without appeal to any part of our weaker nature, without the gorgeous trappings of painting or music, yet sublimely pure, and capable of a stern perfection such as only the greatest art can show. The true spirit of delight, the exaltation, the sense of being more than Man, which is the touchstone of the highest excellence, is to be found in mathematics as surely as poetry."

**Bertrand Russel**

## Introduction

The natural world in which we live is as beautiful as intricate and incomprehensible to us men, but it is perhaps it's infinite complexity that makes it the perfect playground for the human mind, and that fuels our will to discover nature's most obscure truths and hidden laws.
Mathematics stands at the core of the eternal struggle for knowledge that is so characteristic of the human specie, and strive to seek out patterns and formulate new conjectures, to provide us with a language, and the tools to ultimately understand and even predict natural phenomena.

The aim of this essay is to provide visual effects artists of any level with an overview on fractal geometry, and a basic knowledge of the logic and grammar of L-systems, as well as an understanding of the possibilities offered by a procedural approach to 3D modeling. Mathematical technicalities, and extremely hard concepts such as fractal dimension are here completely ignored, for I myself am still far from being capable of understanding them. Perhaps half of the power of fractals comes from the fact that they can be approached entirely heuristically, and therefore anyone who is interested in this fascinating topic is invited to read along.

## A qualitative overview of fractals

The challenge of Computer Graphic, and specifically the pursuit of realism, is largely a problem of reproducing the beauty of Nature and its visual complexity with synthetic images. Fractal geometry is a language of visual complexity that is particularly suited to describe the kinds of forms found in Nature, and is utilized in CG to model plants, terrains, and other effects  such as smoke, fire, and clouds, as well as for creating textures.

The first time I got my hands on a 3D program I was maybe ten years old, and only now, 13 years later, I realize that was my first experience playing with fractal geometry. The software was Bryce, a 3D modeling, rendering, and animation package specialized in fractal landscapes developed by Ken Musgrave, a student of Benoit Mandelbrot, the father of fractals.

But what exactly is a fractal? To use the words of K.Musgrave "a fractal is a geometrically complex object, the complexity of which arises through the repetition of form over some range of scale". [277]
Here the concept of Self-Similarity is introduced, and to explain this further Musgrave writes of the existence of two kinds of complexity in our world: fractal, and non fractal. Non-fractal complexity is generated by the combination of a variety of features through distinct and unrelated  events over time, like holes and stains on a pair of shoes, while fractal complexity can be achieved simply by repeating the same thing over and over, at different scales.
Fractals are self-similar patterns, and many objects in the real world are statistically self-similar. What is also fascinating is the symbiosis that exists between fractal geometry and computer graphic. The latter, developed in the 60s, was fundamental for further advancements in the field of fractal geometry, for it provided us with the ability to quickly create visual representations of complex mathematical algorithms. What's better than feed data to a computer, let it process the data, and get the results back in the form of an image? Fractals on the other hand have helped pushing the boundaries of CG by providing much of the visual complexity in realistic computer graphics.

## Procedural modeling and texturing

The term procedural (procedural generation) refers to content generated algorithmically rather than manually.  Any person who has experience modeling with a 3D software like Maya, or 3dsMax knows very well how challenging it can be to build visually complex objects by hand, namely by moving vertices around  the 3D space.
A 3D character model won't look good until a satisfactory level of detail (complexity) is reached. That is because our eyes are so used to seeing complex forms around us that they will immediately notice, and even be disturbed, by lack of complexity.
That can be achieved in many ways, like by adding accessories, surface detail, and textures to a model. In the case of a character this has to be done manually, for the exception of procedural textures that may come in handy for quickly adding surface detail, but more on this later. While in the case of a terrain, or plants, the modeling process can be almost entirely procedural, which offer several advantages.

## L-systems

In 1968 Hungarian biologist Aristid Lindenmayer developed a string rewriting mechanism to visually describe the growth process of various types of algae he was studying, and other simple multicellular organisms. Rewriting is a technique utilized in mathematics, computer science, and logic for defining complex objects by starting from a simple initial object, and successively replacing its components using a set of rewriting rules, also known as production rules.
The core of a Lindenmayer system is called the axiom, which is nothing but the simple initial object mentioned above. Production rules are then applied to the axiom to generate a more complex string of symbols.

| | | | | | |
|---|---|---|---|---|---|
| Example: | Axiom = DOG | Rules | D = G | Output: | generation1 = GOD |
| | | | G = D | | generation2 = GOD |
| | | | O = O | | generation3= GOD |

The next step to understanding the logic of L-systems is to observe what happens when a recursive rule is applied. That is the case of a symbol being replaced with a copy of itself plus something else. A growth pattern emerges:

| | | | | |
|---|---|---|---|---|
| Axiom = DOG | Rules = | D = DGD | Output: | generation1 = DGDOG |
| | | G = G | | generation2 = DGDGDGDOG |
| | | O = O | | generation3= DGDGDGDGDGDGDGDOG |

An L-system is basically a set of rules for strings made of a set of symbols, and can be split into 4 main components:

A set of **variables**: symbols that can be replaced by production rules
A set of **constants**: symbols that aren't affected and remain the same
A single **axiom** string that can be composed of variables and/or constants
A set of **production rules** that always consist of 2 strings, predecessor and successor
[ D = DGD ]

Recursive L-systems like the one shown above often produce intricate patterns that are self-similar across multiple scales. Nevertheless it'd be very hard, and simply impossible for most people, to perceive these patterns just by looking at a string of symbols.

To obtain a visual representation [or geometric interpretation] of a string of symbols we need to apply a 'drawing instruction meaning' to each symbol in the system. The most common graphic interpretation applied to L-systems is based on **turtle graphics**, a method of programming vector graphics using a relative cursor upon a Cartesian plane developed in the 60s by Seymour Papert as an addition to the Logo programming language.

The **turtle** [cursor] has 3 attributes:
- its location on the Cartesian plane, therefore represented by x and y
- its orientation, or heading
- a pen: which symbolizes the turtle's ability to add qualitative elements such as color and width to its wake

The state of the Turtle at any given point of the plane is defined by [**x**,**y**,$\alpha$] where **x**,**y** define its location and $\alpha$ its heading. Given a step size **d** and an angle increment $\delta$ the turtle can respond to commands represented by the following symbols:

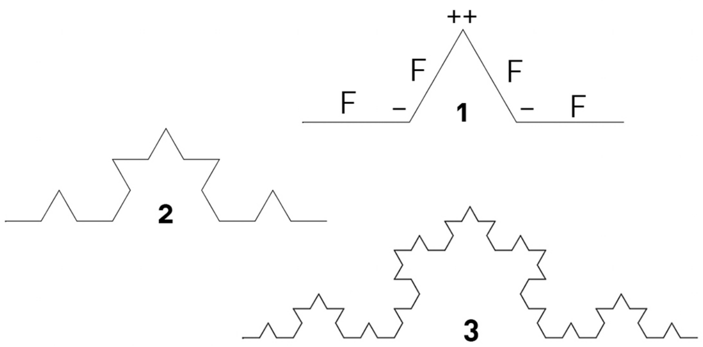
F Move forward a step of length **d** drawing a line. The turtle moves to $(x',y',\alpha)$ where $x' = x + d\cos\alpha$ and $y' = y + d \sin \alpha$.

**f** Move forward a step of length **d** without drawing a line.

**+** Turn left by angle $\delta$. The turtle rotates to $(x, y, \alpha + \delta)$ without moving forward.

**-** Turn right by angle $\delta$.

The following examples of basic fractal curves were described by Swedish mathematician **Helge von Koch** at the beginning of the 20th century, and are shown here using the grammar of L-systems:

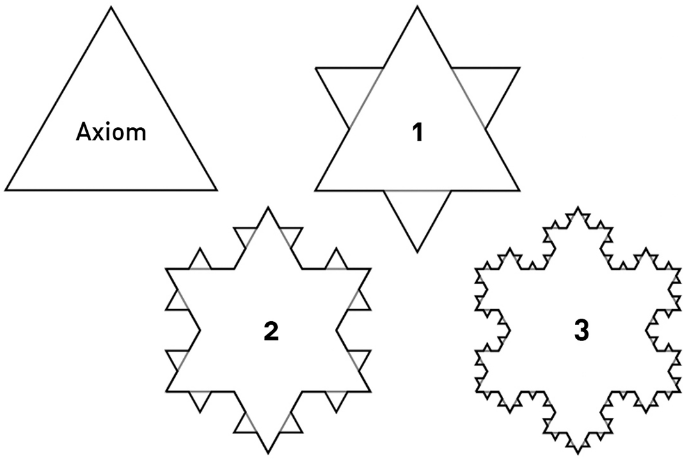


## Koch Curve

**Axiom** : F
**Constants** : +, –
**Production rules:** F = F–F++F–F
**Initial degrees:** 00.0
**Angle increment $\delta$:** 60.0

**First 3 generations: [output]**
**1** F-F++F-F
**2** F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F
**3** F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F
++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F

**Koch Snowflake:**
**Axiom** : F++F++F
**Constants** : +, –
**Production rules:** F = F–F++F–F
**Initial degrees:** 00.0
**Angle increment δ:** 60.0



**First 2 generations: [output]**
**1** F-F++F-F++F-F++F-F++F-F++F-F
**2** F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F+
+F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F

Today several 3D packages like Houdini are capable of implementing the rules of L-systems to quickly create realistic 3D plants. One of the advantages of working with L-systems to generate procedural complex geometries is certainly that of saving time. However creating a procedural tree that closely resembles a specific real tree can take quite a bit, and having to create digital doubles of real objects/characters is a common task for a 3D post-production facility. The greatest advantage is that once the model is generated it is very inexpensive to make changes, whereas it'd be very tedious and time consuming (if not impossible) to apply major changes to a manually built complex geometry.

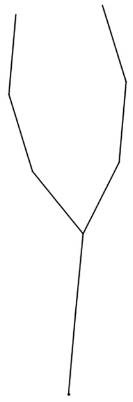
## Building a Tree

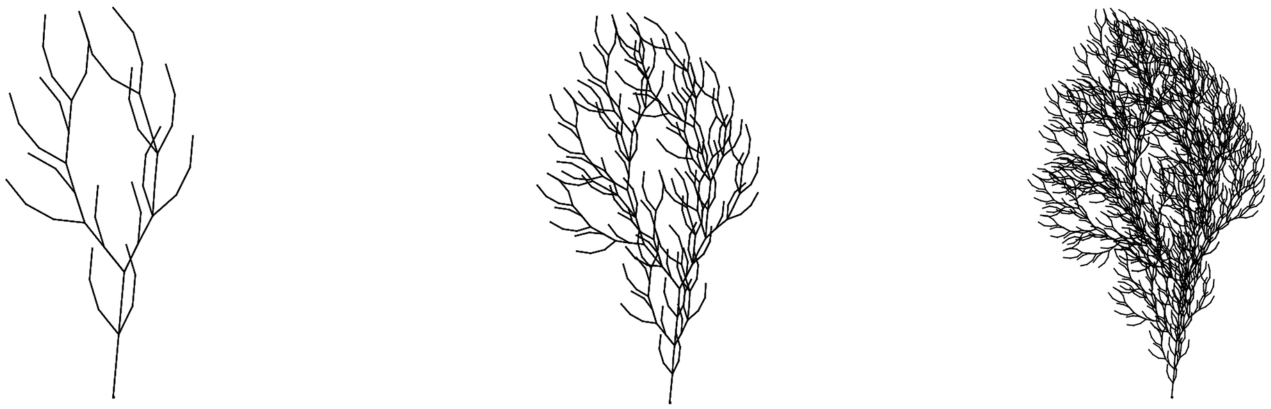
Additional symbols need to be added to the grammar of L-systems in order to obtain more complex shapes and to mimic the branching effect typical of plants.
Different softwares use different symbols to execute the same kind of operations, in general those symbols can behave either as a variable, or as a constant, and can draw or skip a segment.

In the following 2 examples the symbols **x** and **y** are added to the grammar in order to build more complex strings. In the first case they are used as a constant, and in the second as a variable. In both cases they don't execute any command, and therefore do not directly affect the cursor [they are ignored by the turtle]. They can only affect the way **F** is rewritten at each iterations.
To mimic the branching effect of a plant we use square brackets **[ ]** . Everything written inside the brackets gets executed separately.

**Tree**

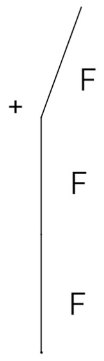
**Axiom :** F
**Constants** +, −, x, y,
**Production rules:** F = FF-[-F+F+F]+[+F-F-F]
**Initial degrees:** 85.0
**Angle increment δ:** 22.0

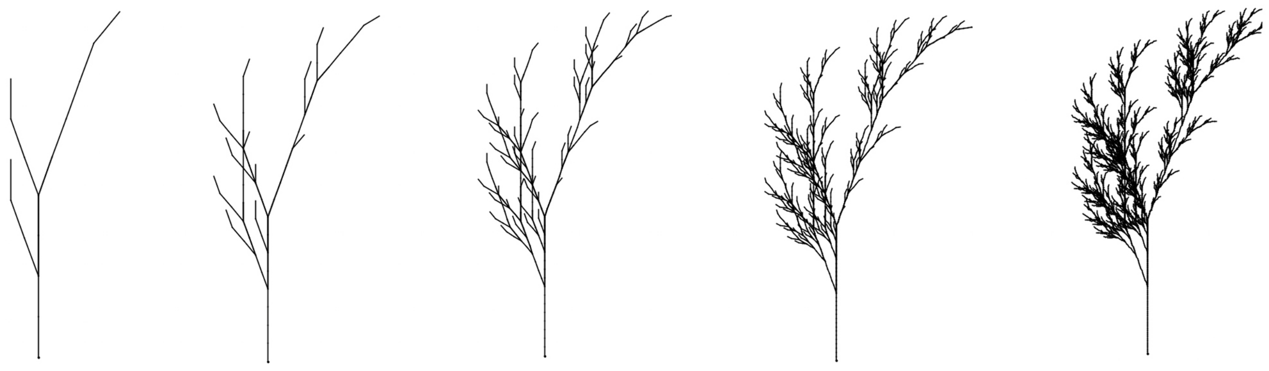




**Plant**

**Axiom :** x
**Constants** +, −, y
**Production rules:** F = FF x = F-[[x]+x]+F[+Fx]-x
**Initial degrees:** 90.0
**Angle increment δ:** 20.0





Many fractals like those shown so far can be thought of as a sequence of segments. You may have noticed that the value of **d**, the step size, was never taken into consideration. That is because the software used to write those systems doesn't support changes to the parameter. **d** remains a constant in every example, and it's simply scaled down at every iterations so that the image can fit the screen. Those systems are also **deterministic**, meaning that there's only one correct answer for each generation. Probability based random variations can be added to the systems to construct more realistic natural shapes.

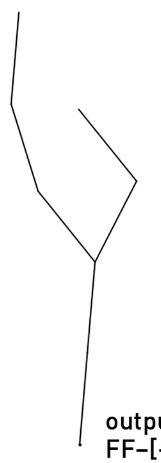
Probability is added to the system by adding alternative production rules as show here:

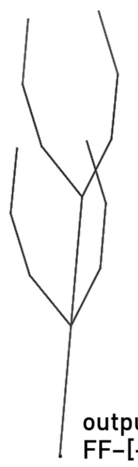
| | | | |
|---|---|---|---|
| **Production rule:** | F = F–F++F–F | or | F+F-F |
| **with probability (sum=1.0)** | 5.0 | | 5.0 |

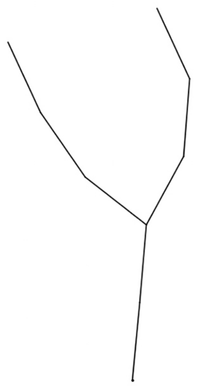
In this case we have a 50% chance that the string F will be replaced by either one of the two strings. Depending on the software capabilities it may be possible to add several other  output strings, each one with it's own rate of probability.

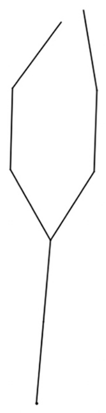
In real life the growth of a plant can be affected by a variety of unpredictable factors both internal organic developmental errors, and external factors such as the effect of wind, sun, and water. These unpredictability can be resembled by intentionally adding **errors** to the system.
Affecting the way the patterns are generated will result in topology mutations [different numbers of segments connected differently] that resemble those determined by internal factors, while external factors can be mimed by slightly changing the angle and length of a segment.
While these effects could be produced manually [to achieve specific results], by writing alternative output strings as shown above, softwares are usually equipped with an 'error editor' that leaves this task almost entirely to the computer.

**Examples of mutations on a tree**


**output:**
**FF–[–F+F+F]+[+F–––F]**


**output:**
**FF–[–F+F+F]+[+F–F–F]FF–[–F+F+F]+[+F–F–F]**





## Conclusion

The fractals we have observed are all considered to be 1 dimensional since they only exists on a plane. 2D and 3D dimensional fractal geometries can be created with the help of specific softwares. L-systems are just one of several languages that can be used to visually interpret fractals.
Procedural textures are another powerful tool in the hands of a 3D artist, and as mentioned above they are a great way of quickly and inexpensively adding visual complexity to a model. [the term inexpensive refers here to the amount of memory required by the cpu to process a certain operation] Algorithms are used to create realistic representations of natural elements like wood, granite, metal, and stone, and the natural look is achieved by the usage of fractal noise.
One of the most popular procedural texture is the Perlin Noise, which secured his inventor Ken Perlin an Academy Award for Technical Achievement in 1997.
The Perlin noise is considered to be a primitive gradient noise, and can be used to generate a wide variety of procedural textures. Thanks to its controlled random appearance it can be used to create effects such as smoke, fire, and clouds.

I personally find computer generated art to be the most fascinating among all , because despite its intangible nature it is by far the most complex, as it incorporates so many different branches of the human knowledge.

**Valerio Paolucci**
[3D Artist]

12 [ 16 ] 2012

## Sources

| | |
|---|---|
| Texting & Modeling A procedural approach | **F.Kenton Musgrave** |
| The Algorithmic Beauty of Plants | **P.Prusinkiewicz A. Lindenmayer** |
| Creating a digital tree and Using L–systems in Production for What Dreams May Come | **David Prescott [Digital Domain]** |
| http://classes.yale.edu/fractals/software/swinglsystem.html | |
| http://www.cs.unm.edu/ | |
| http://en.wikipedia.org | |