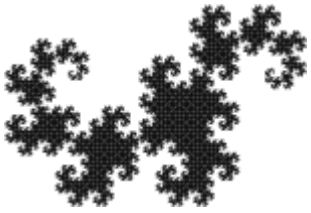
L-Systems in PostScript

The Dragon Curve



The Dragon Curve is a famous fractal. I first saw it on the cover of a (French) book I was reading as a teenager. I then started to draw beautiful dragons, my hand. Later, I decided to write a program to draw the curve. The program was written in Pascal. It worked and drew nice dragons. Unfortunately, I've lost the source code for it. I would be curious to look at it today, because *the program did not use recursion*! I know that for sure because recursion didn't make any sense to me at that time. (The *Towers of Hanoi* was the real eye-opener for me, but it came later.) I vaguely remember that the program was doing repeated iterations over a large array, but I don't remember the details. It must have been an ugly piece of code...

L-systems

Years later, I found on the web a PostScript program that could draw the Dragon Curve. It intrigued me enough to actually make me learn PostScript just to understand the program. I had then some fun with PostScript and prime numbers (programming-primes.html).

But I couldn't understand the algorithm used in the PostScript program. It seemed to rely on two mutually-recursive functions and rotations of 45 degrees (even though all angles in a Dragon Curve are right angles). Once you clean up the code (which was pretty terrible, with global variables shared between recursive calls), it seemed to boil down to this:

```
X -> -FX++FY-
Y -> +FX--FY+
F ->
```

where a form or "turtle graphics" is used and `-` represents a *clockwise 45 degrees rotation*, `+` is an *anticlockwise 45 degrees rotation* and `F` draws a line forward.

I thought this algorithm was remarkable, certainly too good to have been invented by a guy who uses global variables between mutually recursive functions. After searching for more Dragon Curve programs on the web, I found somewhere that exact formulation. It was part of a list of *Lindenmayer systems* (L-systems) that drew everything from classic fractals (Hilbert, Koch) to various plant-like pictures.

L-systems were invented by Aristid Lindenmayer, a Swedish biologist who used them to model the growth of plants and end up publishing about them in both biology and computer science journals. L-systems have been extended to 3D and widely used in plant biology and computer graphics. The reference book seems to be The Algorithmic Beauty of Plants (http://en.wikipedia.org/wiki/The_Algorithmic_Beauty_of_Plants), by Przemyslaw Prusinkiewicz and Aristid Lindenmayer. Although all the equations that are used in this page were found on the web, I strongly suspect they originated from this book. (All I can say is that they are definitely not mine.)

The Dragon Curve as an L-system

Using the set of rewriting rules above (starting from `X`) and applying them iteratively, here is what you get (read a curve from left to right, starting horizontally):

1. `-F++F-`

2. `--F++F-+++F--F+-`

3. `---F++F-+++F--F+-+++-F++F---+F--F++-`

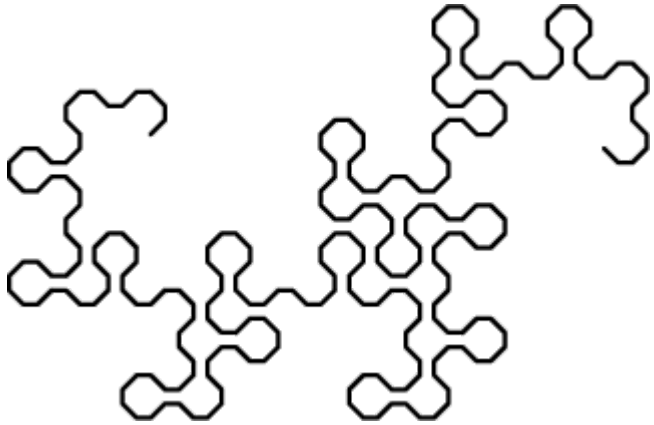
4. `----F++F-+++F--F+-+++-F++F---+F--F++-+++--F++F-+++F--F+----+-F++F---+F--F+++-`

5. `-----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++++-`

6. `------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++++-`

7. `-------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++++-+++----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++++-`

If you “round the corners”, you better see how the curve is drawn. Here it is, at order 7:



If it still looks more like a French poodle than a dragon to you, it's because you have no imagination.

Dragon Curve in PostScript

Now, here's the PostScript program that does the job.

First, tell the printer it's a PostScript file:

```
%!PS
```

Then, specify the order (number of iterations). Beyond 20, it may take forever to draw and print, of course:

```
/N 7 def
```

Here is the core of the computation, functions `X` and `Y`. The order value (or depth) in on the stack and is decreased by one and duplicated 4 times for the 4 function calls. When the order is zero, stop; otherwise, do the recursive calls according to the equation:

```
/X {
  dup 0 ne
  {1 sub 4 {dup} repeat - F X + + F Y -}
  if pop
} def

/Y {
  dup 0 ne
  {1 sub 4 {dup} repeat + F X - - F Y +}
  if pop
} def
```

Function `F` does nothing and just disappears, except at the end, when order is 0, it draws a piece of line:

```
/F {
  0 eq { 10 0 rlineto } if
} bind def
```

Functions `-` and `+` do right and left 45 degree rotations, as expected:

```
/- { -45 rotate } bind def
/+ { 45 rotate } bind def
```

Make sure that pieces of lines are joined nicely:

```
1 setlinejoin
1 setlinecap
```

Finally, decide where to start and scale the drawing according to the order value so that it's always the same size, no matter what the order is:

```
newpath
220 180 moveto
50 N { 2 sqrt div } repeat dup scale
```

Rotate the drawing to landscape, and start:

```
90 rotate
N X
```

And proudly show the resulting page:

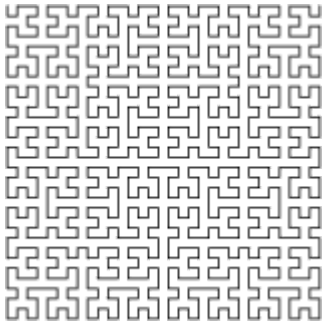
```
stroke
showpage
```

The whole PostScript program can be downloaded here (programming-lsystems-dragon.html).

More L-systems

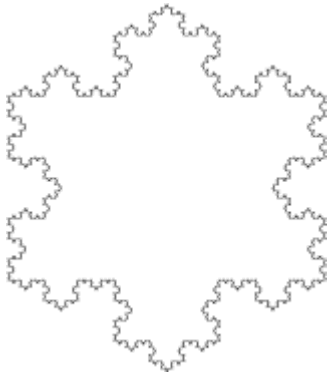
On my way to understand the mysteries of this Dragon Curve program I had found, I encountered a number of L-system equations. Some of them are classic fractals, like Hilbert's curve:

```
angle 90
START -> X
X -> -YF+XFX+FY-
Y -> +XF-YFY-FX+
F ->
```



or Koch's snowflake:

```
angle 60
START -> +F--F--F
F -> F+F--F+F```
```



But the ones I like best are plant-like pictures. (After all, Lindenmayer *was* a biologist.) What I find interesting about them is how different they are from each other while based on similar sets of equations. Some of my favorites are below:

```
angle 22.5
START -> F
F -> FF-[-F+F+F]+[+F-F-F]
```

```
angle 22.5
START -> X
X -> F-[[X]+X]+F[+FX]-X
F -> FF```
```

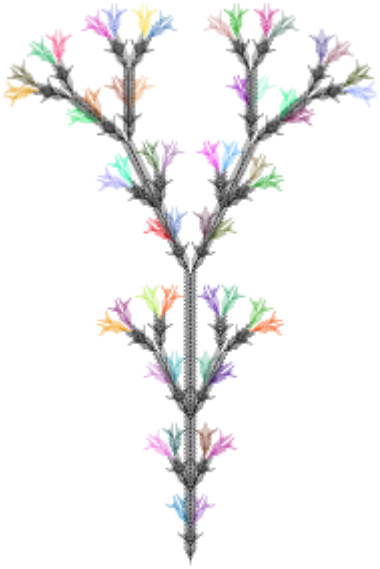
```
angle 20
START -> F
F -> F[+F]F[-F][F]
```

```
angle 25
START -> X
X -> F[+X][-X]FX
F -> FF
```

```
angle 20
START -> X
X -> F[+X]F[-X]+X
F -> FF
```

```
angle 25
START -> F
F -> F[+F]F[-F]F
```

```
angle 25
START -> Y
X -> X[-FFF][+FFF]FX
Y -> YFX[+Y][-Y]
F ->
```



```
angle 30
START -> F
F -> F[+F[+F][-F]F][-F[+F][-F]F]F[+F][-F]F
```

I have not mentioned `[` and `]` which are used for branching. Basically, `]` brings you back in the drawing at the point where the corresponding `[` was issued. They are easily implemented in PostScript using `gsave` and `grestore`, used to push and pop the graphics context on and off the stack.

L-systems in PostScript

The plants on this page were all rendered using this PostScript program (programming-lsystems-plant.html) by simply modifying the equations (`[` and `]` are called `B` and `E` because the square brackets are part of PostScript's syntax). This program is a little more complicated than the Dragon Curve before, but it's mostly because of multiple color schemes and because it's made generic to work with many different equations.

For reasons that were never clear (and probably involve cosmic forces beyond our comprehension), this experiment in programming L-systems in PostScript was published by the *Zpravodaj Československého sdružení uživatelů TeXu* (2012, vol. 1) under the title *Programujeme L-systémy v PostScriptu*. The paper (http://bulletin.cstug.cz/pdf/2012-1.pdf) contains a few more technical details and, in spite of its Czech title, is written in English.

Links:

- Programming L-Systems in PostScript (http://bulletin.cstug.cz/pdf/2012-1.pdf), Zpravodaj Československého sdružení uživatelů TeXu (CSTUG: Czech and Slovak TeX Users Group), 2012.
- Wikipedia page on The Algorithmic Beauty of Plants (http://en.wikipedia.org/wiki/The_Algorithmic_Beauty_of_Plants)
- Adobe page on PostScript (http://www.adobe.com/products/postscript/)
- PostScript language reference (https://www.adobe.com/products/postscript/pdfs/PLRM.pdf)
- another experiment in PostScript programming: prime numbers (programming-primes.html)

Downloads

- *Dragon* PostScript program (programming-lsystems-dragon.html)
- *L-Systems* PostScript program (programming-lsystems-plant.html)
- *Zpravodaj Československého sdružení uživatelů TeXu* paper (http://bulletin.cstug.cz/pdf/2012-1.pdf)

last edited 2016-09-25