L-Systems

GONZALO GARCÍA ALARCÓN ESTRADA, 1004229676
Jan / 9 / 2018
University of Toronto, Shameless Mathematica

L-Systems, being susbsititution based, seemed like a good idea to implement in mathematica using *rules* and *replacements*. Worked fine, I just got stuck when implementing the variation for trees.

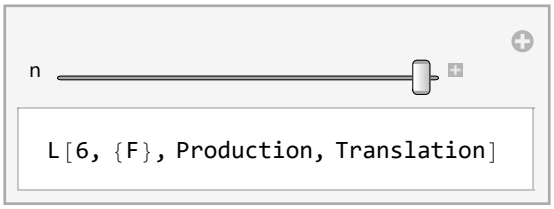
I got most of the rules and starting axioms from here:
http://algorithmicbotany.org/papers/abop/abop-ch1.pdf

```
(* L-System plotter, using "Turtle" (AnglePath) Plotting *)

L[n_Integer, axiom_List, Production_List, Translation_List] := Module[{i = 1, a},
  a = axiom;
  While[i ≤ n,
   a = a /. Production // Flatten; i++
  ];
  Graphics[a /. Translation // AnglePath // Line, ImageSize → Medium]
 ]
```

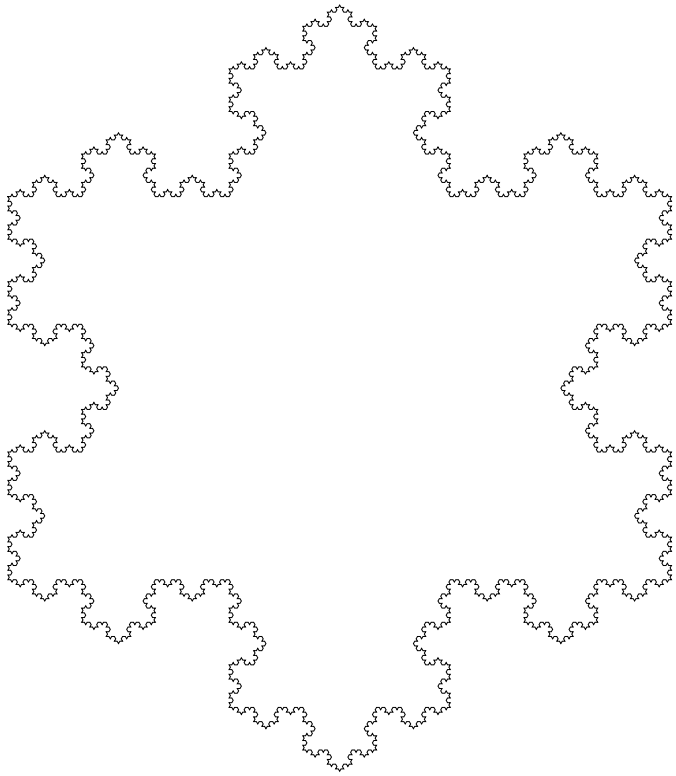
Kock Curve and Snowflake

```
(* Definiton of L-System *)

a = {F}; (* axiom or start *)
Production = {F → {F, r[60], F , r[-120], F, r[60], F}}; (* recursion rules *)
Translation = {F → {1, 0}, r[x_] :→ {0, x Degree}}; (* drawing (turtle) translation *)
```

```
Manipulate[L[n, {F}, Production, Translation], {{n, 1}, 0, 6, 1}]
```



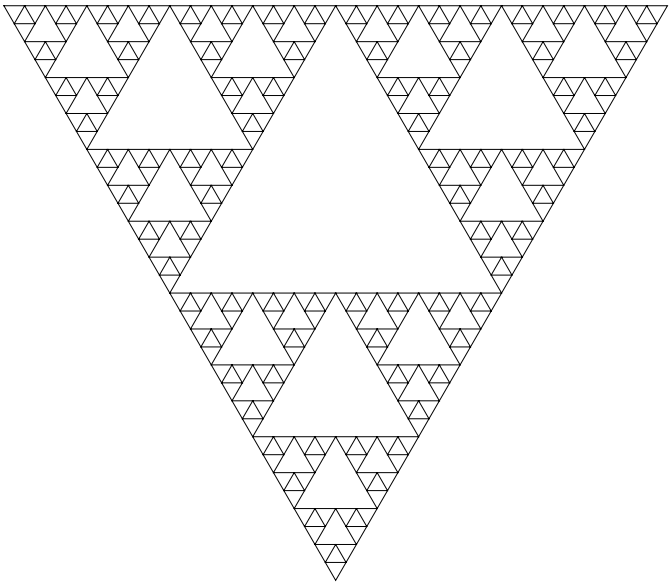
```
L[5, {F, r[-120], F, r[-120], F, r[-120]}, Production, Translation]
```



Sierpinsky Triangle

```
(* Definiton of L-System *)

a = {F, m, G, m, G}; (* axiom or start *)
Production = {G → {G, G}, F → {F, m, G, p, F, p, G, m, F}}; (* recursion rules *)
Translation = {F → {1, 0}, G → {1, 0}, p → {0, 120 Degree}, m → {0, -120 Degree}};
(* drawing (turtle) translation *)
```

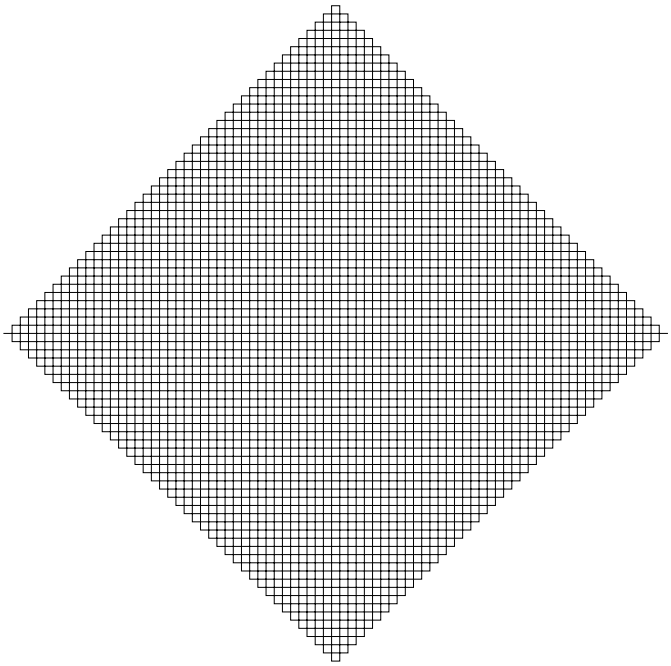
```
a = {F, m, G, m, G};
L[5, a, Production, Translation]
```



A Peano Curve

```
(* Definiton of L-System *)

a = {F}; (* axiom or start *)
Production = {F → {F, p, F, m, F, m, F, m, F, p, F, p, F, p, F, m, F}};
(* recursion rules *)
Translation = {F → {1, 0}, p →  {0, 90 Degree}, m →  {0, -90 Degree}};
(* drawing (turtle) translation *)
```

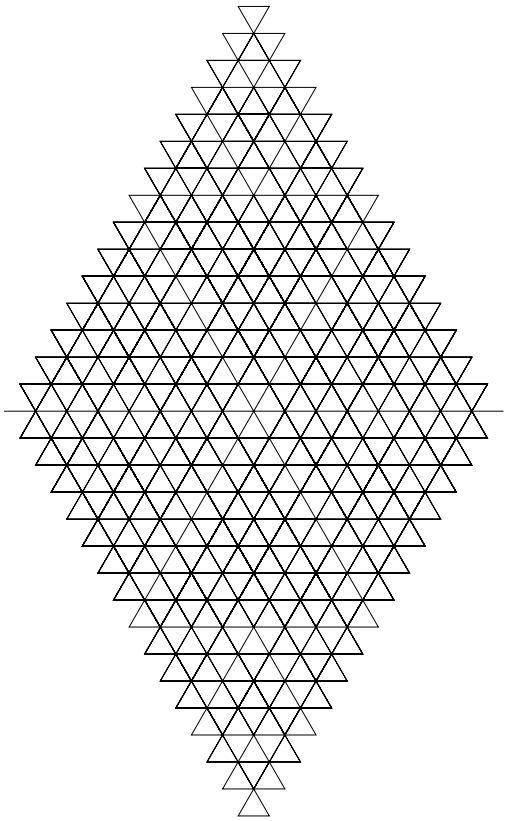
```
a = {F};
L[4, a, Production, Translation]
```



Another Peano Curve (similar to the one drawn in class)

```
(* Definiton of L-System *)

a = {F}; (* axiom or start *)
Production = {F → {F, p, F, p, p, F, p, p, F, F, m, m, F, m, m, F, m, F}};
(* recursion rules *)
Translation = {F → {1, 0}, p → {0, 60 Degree}, m → {0, -60 Degree}};
(* drawing (turtle) translation *)
```

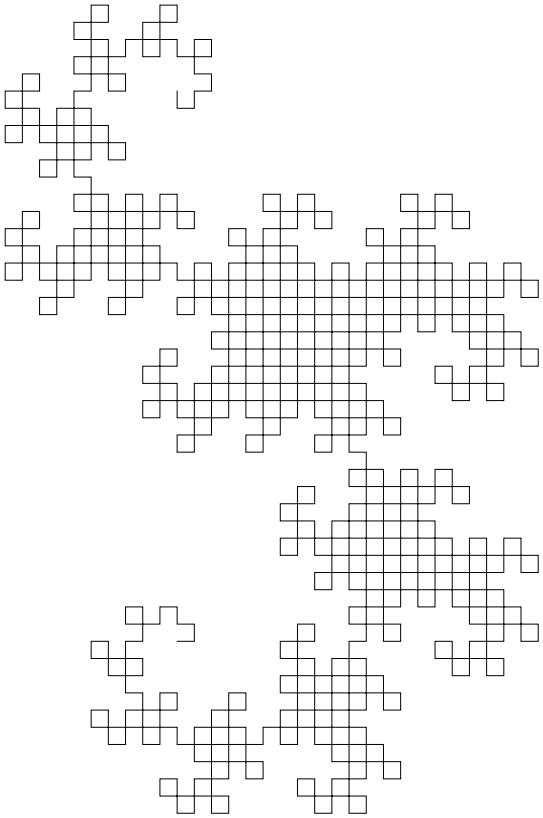
```
a = {F};
L[4, a, Production, Translation]
```



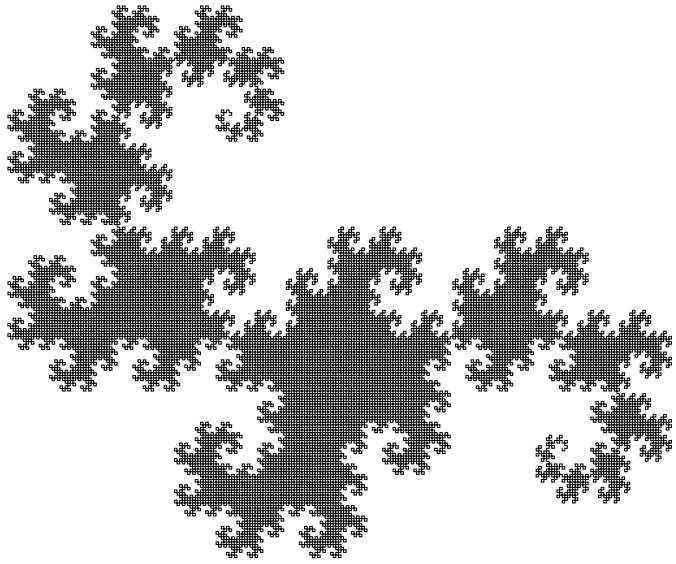
A Dragon Curve

```
(* Definiton of L-System *)

a = {F, X}; (* axiom or start *)
Production = {X → {X, p, Y, F, p},
   Y → {m, F, X, m, Y}}; (* recursion rules *)
Translation =
  {F → {1, 0}, Y → {0, 0}, X → {0, 0}, p → {0, 90 Degree}, m → {0, -90 Degree}};
 (* drawing (turtle) translation *)
```

```
a = {F, X};
L[10, a, Production, Translation]
```



```
a = {F, X};
L[15, a, Production, Translation]
```



Binary Tree (try 1)

This is where I failed..... I could not implement the idea “save state and return” with my implementation. Hence I cannot draw tree-like L-systems since I require to store and return. Perhaps this is not suitable for using AnglePath, and needed to code my own drawing code.

```
(* Definiton of L-System *)

a = {F}; (* axiom or start *)
Production = {F → {T, r[45], F , r[180], R, r[90], F, r[180], R, r[45], R, r[135]},
   T → {T, T}}; (* recursion rules *)
Translation = {F → {1, 0}, r[x_] :→ {0, x Degree}, R → {1, 0}, T → {1, 0}};
(* drawing (turtle) translation *)
```

```
a = {F};
a = a /. Production // Flatten
```

```
{T, r[45], F, r[180], R, r[90], F, r[180], R, r[45], R, r[135]}
```

```
a = {F}
L[2, a, Production, Translation]
```

```
{F}
```