L Systems

Example simple L System

L-systems are great examples of recursion and using JFLAP they can be used in a very nice illustrative manner.

Unlike Grammars where we replace one variable (non-terminal) at a time, in an L-system all possible replacements are made in one go.

Note: In JFLAP, all unique symbols are separated by spaces. This means that entering "g g" would be considered as representing two unique instances of "g", and entering "gg" would be considered one instance of the unique symbol "gg".

The L systems consist of axioms and replacement rules

The example we will look at will be the following

Rules

$$X \rightarrow gY[+g-X]g$$
$$Y \rightarrow gY$$

Load the file LSystemSimple.jflap or alternately click on the L-system option in JFLAP and then enter that axiom and those rules.

Now each

Refer to the documentation available at http://www.jflap.org/tutorial/lsystem/index.html to understand what the symbols mean.

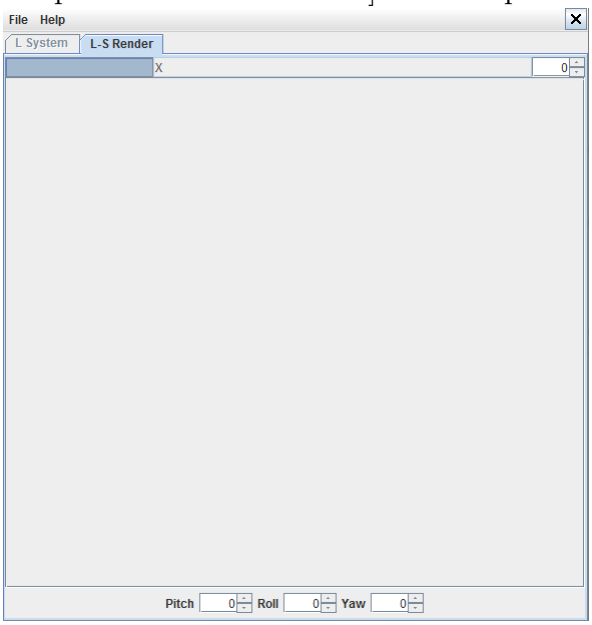
Just for completeness of this example the ones that we need are

- g means move forward with the pen down
- + means rotate to the right by a certain number of degrees. The amount of rotation is specified by the angle parameter.
- - means rotate to the left by a certain number of degrees. Again the rotation amount of specified by the angle parameter.
- [and] are used to save state onto a stack. Since these systems are inherently recursive, the recursive action is simulated using a stack. Everything between the [and] gets pushed on the stack.

Example rendering

The way an L-system is rendered is we begin with the axiom, which in this case is the X and then apply the rules. Remember, as said before, with an L-system you are looking to replace every variable using the rule.

If you are familiar with turtle graphics turtle, the rendering is very similar to that. The rendering can be imagined as a pen passing through the current string and following the instructions. When it encounters a [, it saves state on the stack and pushes things into a stack up until it encounters a]. It then proceeds with the next direction.



Since the first step just has an X, the pen does not render anything.

The number in the upper right corner controls the number of levels of recursion currently being observed. If we increase that to a 1, that means we get the following string to process g Y [+ g - X] g.

Step by step this would mean

1. go forward
2. store current position on a stack. Let us call that position A.
3. push a 30 degree right rotation
4. push a go forward
5. a 30 degree left rotation
6. stop pushing to the stack and get ready to execute the next
7. go forward
8. come back and clean up the stack which in this case involves
   (a) rotating at A 30 degrees to the right
   (b) going forward

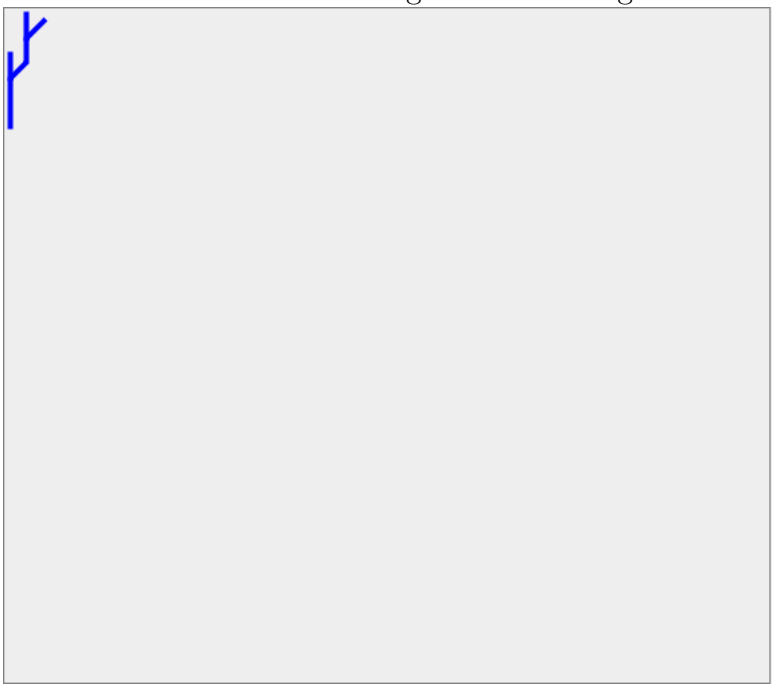
(c) rotating back 30 degrees to the left. no visible effect in this case.

This results in the following

The next step gets more interesting since we not have to use the rule for X as well as the rule for Y.

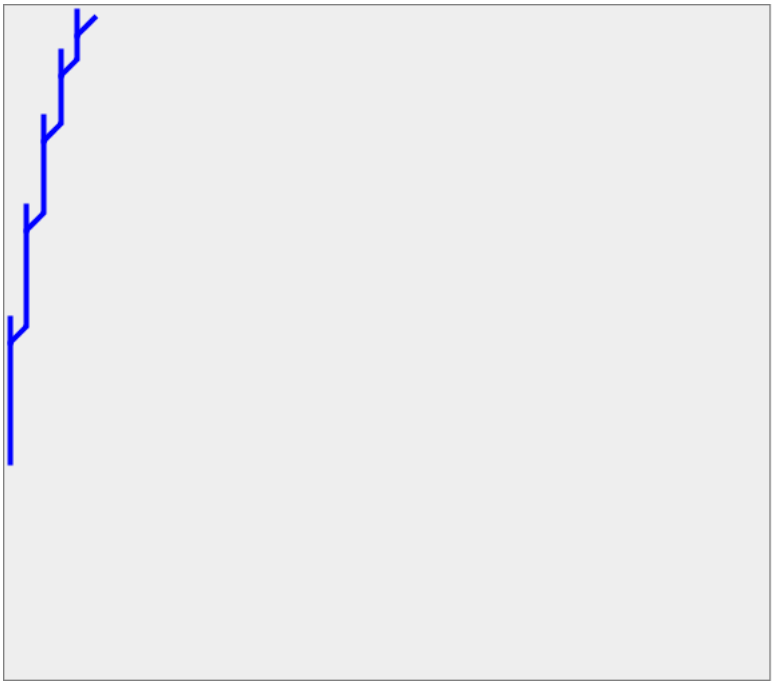
So first of all, let us apply the rules to get ggY[+g-gY[+g-X]g]g.

Now when we render this we get the following



which is basically a recursive application of the initial branching pattern.

he recursion to more steps just produces a more branched tree. Here is the result after 5 steps.



Questions

Think about how you could get the tree to branch on both sides?