LSystemsMaya

Basic L-System rewriting with geometric interpretation in Maya.

Installation

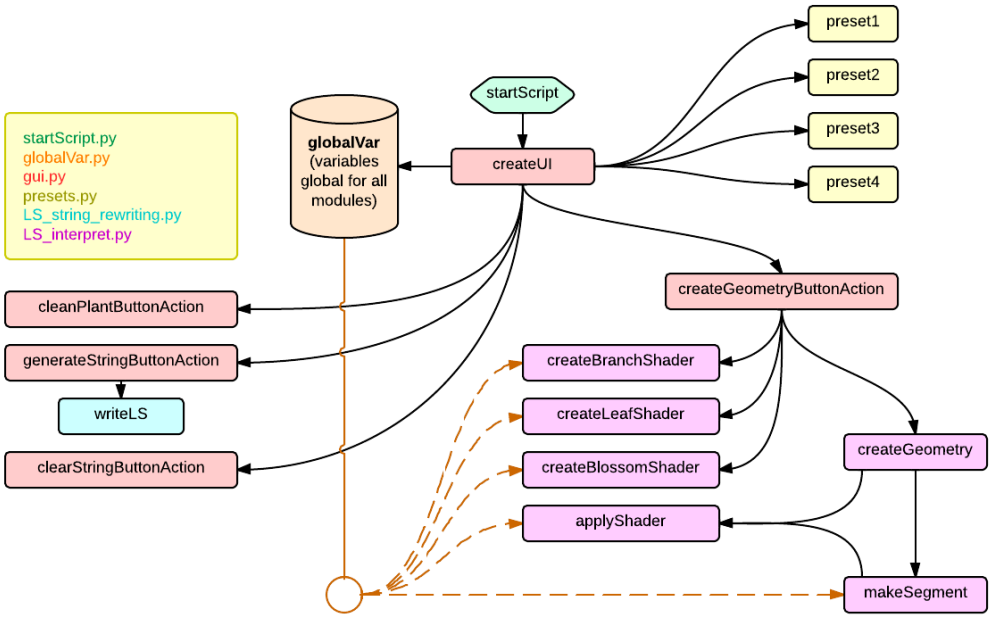
First of all, if you are not familiarized with git and you don't know how the cloning stuff works, just keep it simple: Download all this project as a ZIP file (button on right side of the screen). Then, once you have extracted all the contents on your Desktop or wherever you want..

- Uncompress the ZIP file you just downloaded
- Open `startScript.py` with Notepad or your favourite text editor
- Copy all the content from it.
- Paste it to the Script Editor in Maya
- Execute the code
- You will be asked to point the location where all the scripts are. You must point to the folder called `script` inside the uncompressed folder in which you have uncompressed the ZIP file.
- The UI should be created, then you can click on "Instructions" in order to get started as soon as possible.

This method for including the pointed folder as a sys.path for maya to access the files has been adapted from Jared Auty's wonderful scripting project, which you can download here http://bit.ly/1kDUXhj

Usage

I built up the UI considering standard users. **There's a Help Line**, there's an **Instructions Button**, and so on, I recommend paying attention to them. What I could do is talk a little bit about L-Systems for those ones who don't know what they are. I will attach a diagram that shows how all the modules are connected:



What are L-Systems?

An L-system or is a parallel rewriting system and a type of formal grammar. An L-system consists of an alphabet of symbols that can be used to make strings, a collection of production rules that expand each symbol into some larger string of symbols, an initial "axiom" string from which to begin construction, and a mechanism for translating the generated strings into geometric structures. *(from Wikipedia)*

My turn now... L-Systems consist on rewriting over a string of characters considering a bunch of rules. For instance you can start

with an initial word, for instance `F` (this is formally called **Axiom**), then you have some rewriting rules that tell to replace this `F` by something a little bit more complex, for instance let's says replace `F` with `XF`.

If you apply this rule for each character of the word you would get `XF`, right? Because you iterated over the initial word. So what if we want to iterate again over this string we got? We could! Because we still have the rule `F --> XF`, so if we run the rewriting system first of all we would come across an `X`, mmm.. we don't have any specific rule for it... Well, nevermind, keep going. Next character is `F`, cool we have a rule for it, it says that we have to replace it with `XF`, go on. *(We rewrite)* Now we get `XXF`.

We could carry on with this, we did 2 iterations over the string in this example, so the **Depth** would be two (now you know what the depth parameter in my script is). You could expand the rules, you could have some rules for `X` as well. We could even have another rule that said `F --> GG` that could live with the previous one `F --> XF`. *Wait, wait! If you have rules that react to the same character (formally called **predecessor*)*, how can you chose which one to apply?* It's simple, with **percentages**, with probability. And L-System which implements probability is called a Stochastic L-System. That's cool, isn't it?

How do they relate to plants? (Turtle)

So far you could be asking yourself, how does that match to plants? Well... Here the **Turtle Concept** plays such an important role. For those ones who don't know what the turtle paradigm is: imagine you were in the beach and you had a turtle which you could control (I don't know how, maybe mental power!).

For instance you could tell the turtle (let's call it Leonardo, he is a male turtle) to move forward. If Leonardo did that, he would leave a path with his tail in the sand, wouldn't he? You would be able to see his trail. Now you could keep yelling commands to him such as *turn right Leo!*, *turn left!*. This is great but our amazing Leonardo can even do more crazy stuff. Imagine you could tell him: **Leo, remember your current position, save it in your little turtle mind**, then, keep walking forward and turning and more stuff until... when at some point we tell him **go back to that position you saved without leaving trail, teleport yourself!**, then, keep doing other movements from the place I told you before to save in your mind. This concept is called **Pushing a state into the stack** (remembering, saving), and **Popping the state of the turtle** (going back to where we saved the state) // With this concept implemented we could leave trails in the sand that would look like a branching system.

So with that silly explanation I wanted to say that we could give a **meaning** to each character in an L-System generated string. For instance we could say `**F** is Move Forward`, `+ is Turn Left`, `- is Turn Right`... We are just building up a language for the turtle commands, we are giving meaning to a bunch of characters, we are giving them a **Geometrical Interpretation**. THIS IS THE ESSENCE, because L-Systems on their own they are nothing but text characters.

There are some conventions I will be using in this script. The turtle we talked about before it's two-dimensional, try to translate the concept to the third dimension. I don't know, imagine Leo equipped with some sort of jet-pack, being the trail he leaves the smoke emitted by the exhaust pipe... Oh... I am being too creative, pardon me. So bearing in mind that, I collected this grammar:

- `F` Move forward
- `f` Move forward
- `L` Leaf
- `B` Blossom
- `+` Rotate +X (yaw right)
- `-` Rotate -X (yaw left)
- `^` Rotate +Y (roll right)
- `&` Rotate -Y (roll left)
- `<` Rotate +Z (pitch down)
- `>` Rotate -Z (pitch up)
- `*` Turtle rotates 180 (as it was facing backwards)
- `[` Push current turtle state on the stack
- `]` Pop the current turtle state from the stack

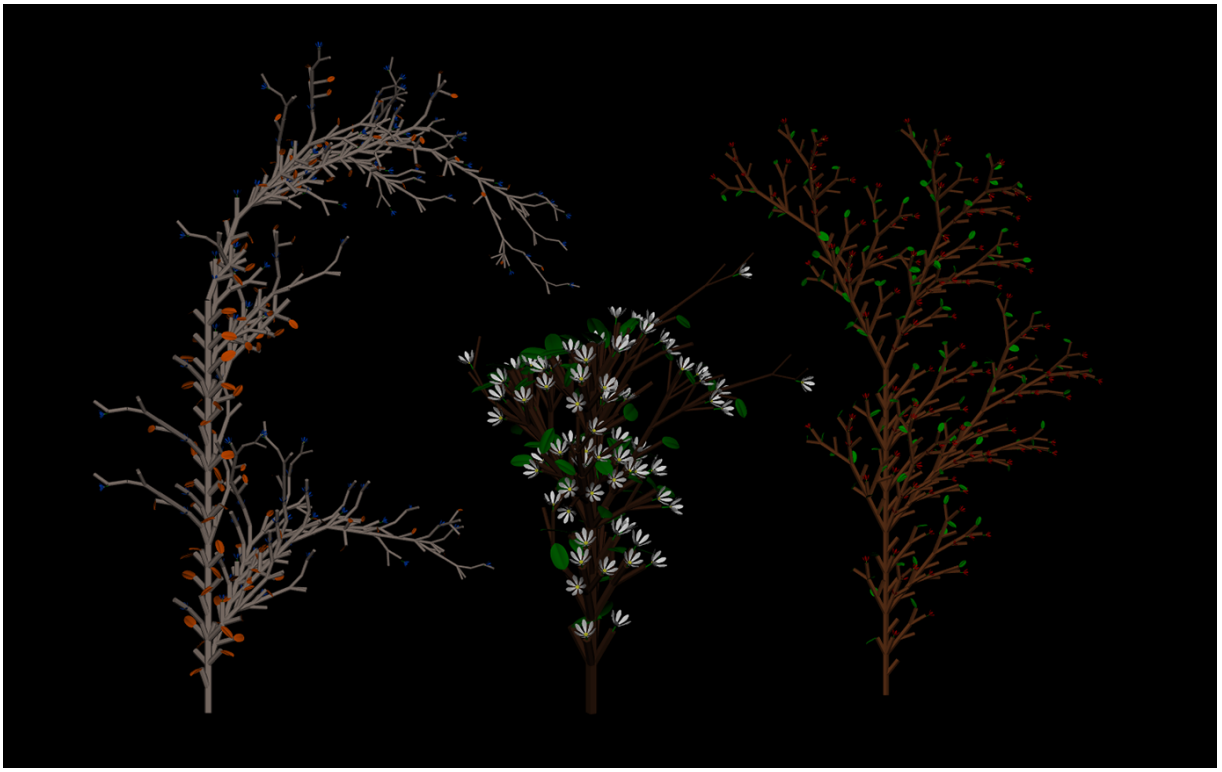
This way we can do complex things for instance having this grammar set:

- Axiom: `F`
- Production Rules: `F --> F[-F]F[+F]F`
- Depth: 3

Then we could say that each step or "F" is a segment of *Length* 2, *Radius* 0.5, and each rotation is done considering a Step *Angle* of 28... We can say as well, each time the turtle *pushes* his state, to slightly *decrease* the length or the radius. This is what I implemented in my script.

Furthermore I added some fancy features such as customizing the *colour for the branches, for the leaves, and blossoms*. And this is pretty much all. You can find all the procedures and modules documented in an **HTML file** inside the documentation folder. Follow the help tips, and instructions of my script, you might need to practise in order to understand better!

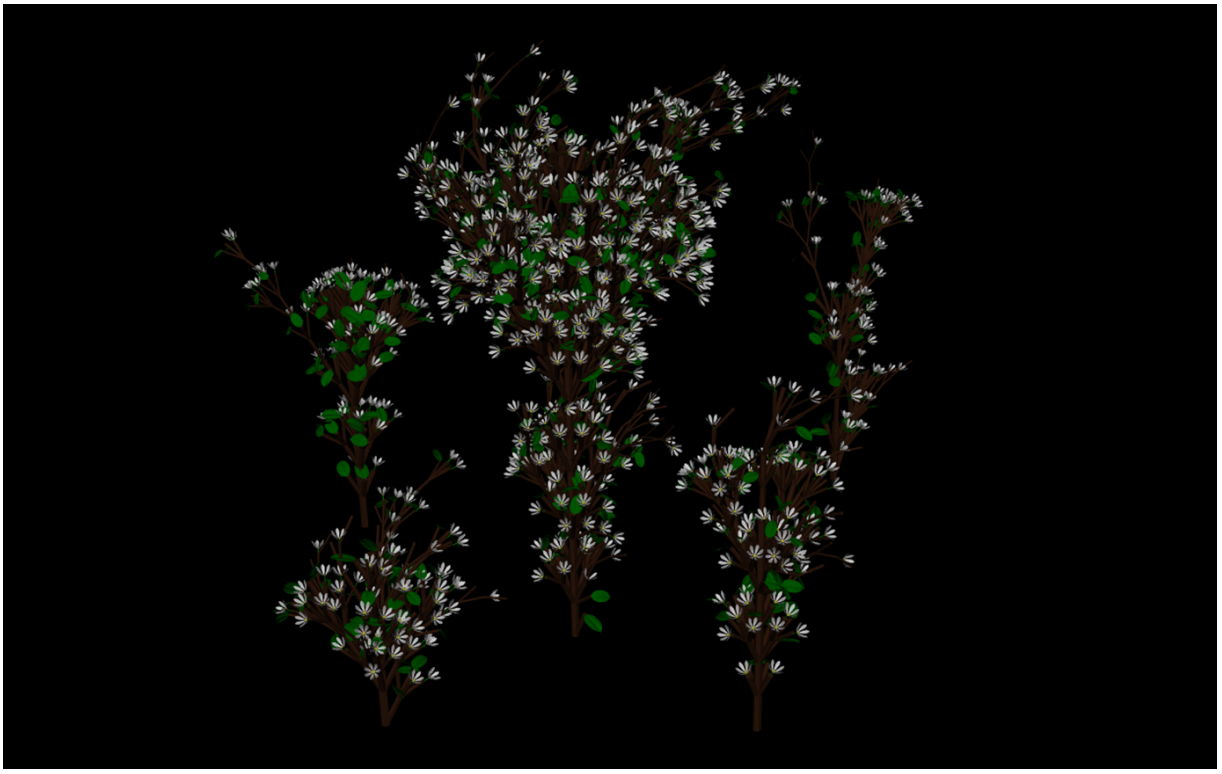
Examples



This is a quick generation using 3 different presets. It's more a general overview. Now I will comment some nice results I got. Major part of them are included in the preset buttons in the UI, try to mess around with them.

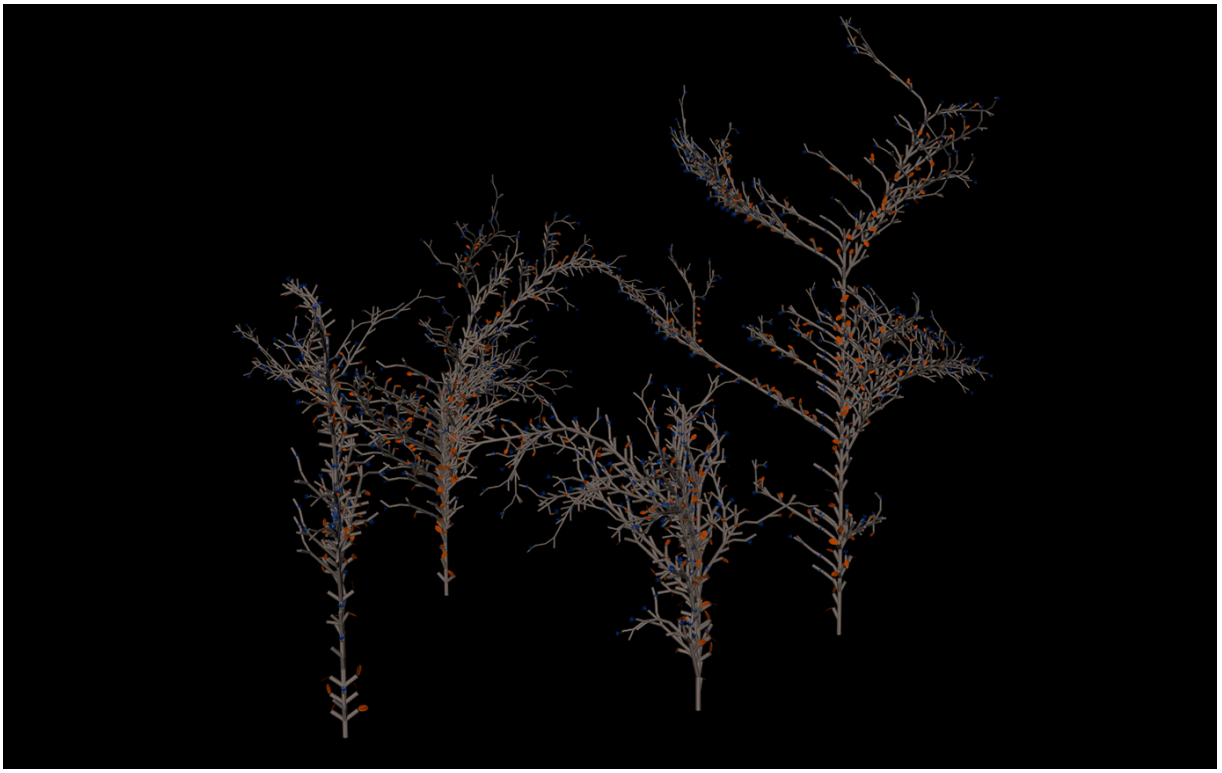
The first preset uses this L-System:

- Axiom: `F`
- Rules: `F --> F[&+F]F[->FL][&FB]`
- Depth: 5
- Angle: 28°



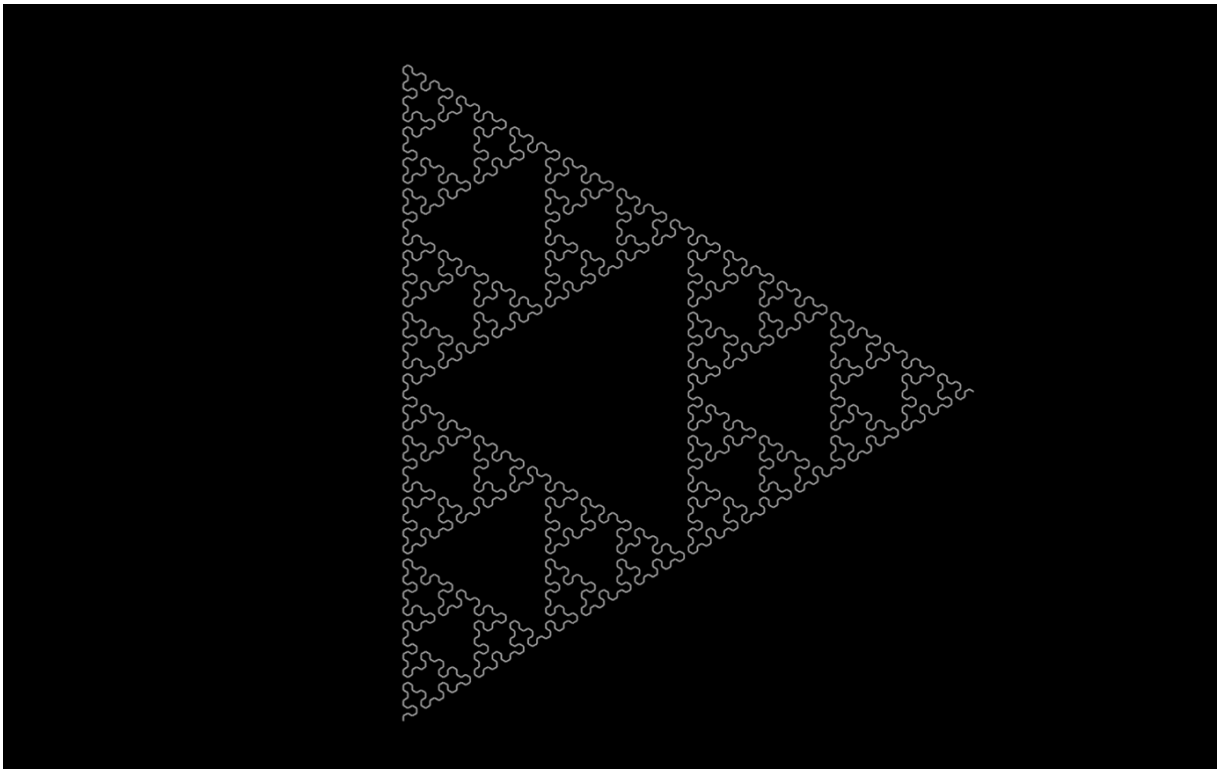
This is a Stochastic L-System example, it's the second preset. I created it by myself, it looks good, I must have been so lucky. As they are Stochastic you can generate a few plants and stick with the one you like most. Use the Outliner and pick the plant group to move the entire geometry, it's more comfortable. The L-System:

- Axiom: `F`
- Rules: **70%** `F --> F[+FL][-FB][&FL][^FB]F`, **30%** `F -- > [-FL]F[F[-FB-&&>F][&>F]]`
- Depth: **3**
- Angle: **25.7°**



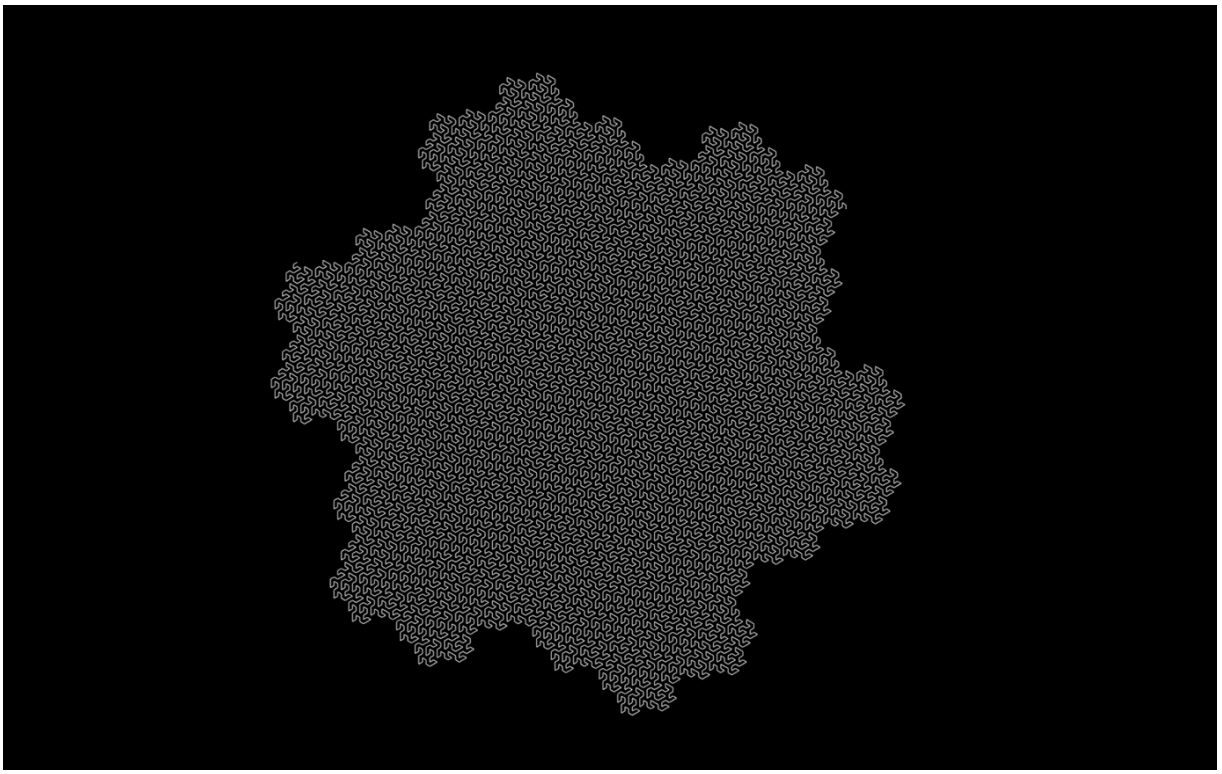
Again another Stochastic L-System preset. In perspective view looks a little bit 2D, but still nice. Tim Burton might love these kind of vegetation.

- Axiom: `S`
- Rules: 33% `S --> S[>>&&FL][>>^^FL]S`, 33% `S -- > S[-FL]F[S[-F-FB-&&>S][&>F][+S]]`, 34% `S --> S[+S[-FB][&>S]]`
- Depth: **6**
- Angle: **26.5°**



Fractal geometry is so beautiful. With this script you can also create them. They can be generated in many ways, and the L-System approach is one of them, our script is valid for it as well. For the Sierpinsky just enter:

- Axiom: `F`
- Rules: `F --> X-F-X` and `X --> F+X+F` (We need no percentages because they don't apply to the same predecessor)
- Depth: **6**
- Angle: **60°**



The Gosper curve is quite famous as well. It's L-System is:

- Axiom: `F`
- Rules: `F --> F+R++R-F--FF-R+` and `R --> -F+RR++R+F--F-R`
- Depth: **5**
- Angle: **60°**

It's normal if you didn't understand exactly what I explained in this readme file, I would recommend you to find a proper explanation of

the L-System concept, because it is so interesting! You can even apply L-Systems to music!

Any questions please drop me an email to *ramon@ramonblanquer.com*, thanks for taking your time reading, and apologies for my crappy English. Love.

Ramon Blanquer (NCCA) ©