Computer Animation Production 1 Assignment (Scripting)

L-System Generator/Interpreter

This scripting project aims to bring the knowledge from *The Algorithmic Beauty Of Plants (1996)*[1] to Maya. My project consists of 4 modules which I will go over in the next section.

- startScript.py
- gui.py
- LS_string_rewriting.py
- LS_interpreter.py

L-Systems on their own are nothing but sequences of characters result of some rewriting rules over an initial string. Imagine you have `ABCD` as your initial word, and you set a rule that says replace B by EFG, `B → EFB`, replace (predecessor) → (successor). The resulting string would be `AEFBCD`. Now we can iterate again over this new string, let's do it. We replace again B and we get `AEFEFBCD`. It is simple, isn't it?

In my script I implemented the possibility to enter percentages for the probability of multiple rules with same predecessor to happen, this allows the script to pick one depending on its chances, this makes then my L-System Generator Stochastic.

Start Script <startScript.py>

This bit was taken from Jared Auty's project and adapted to mine's. It asks the user to point where the script folder is, then, it adds to this path to the sys.path variable temporally so that Python can load the modules without having to move them to the Maya scripts folder.

Graphics User Interface <gui.py>

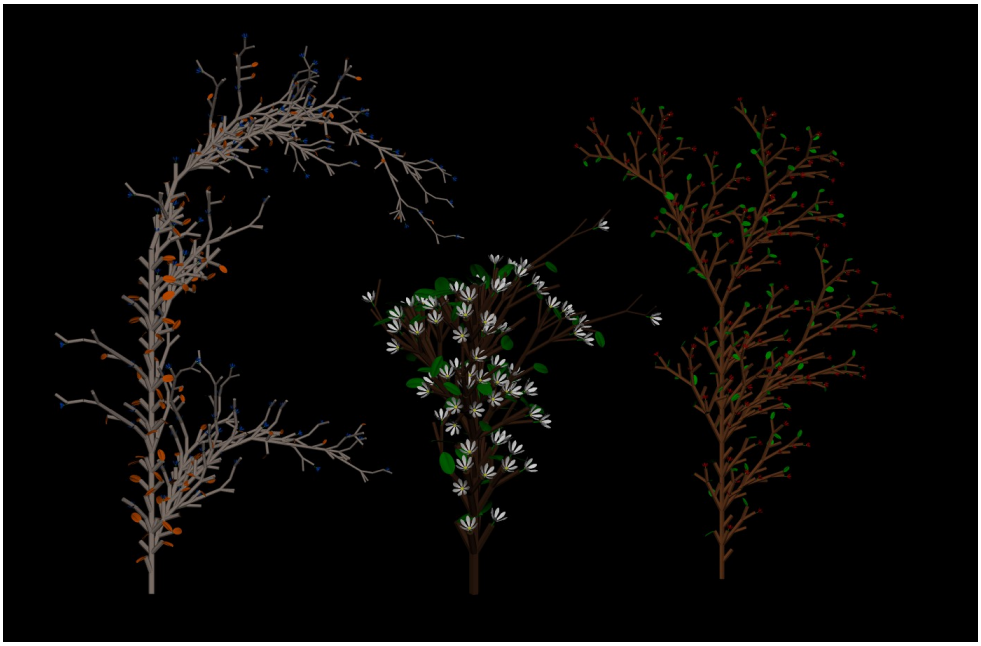
Responsible for the user interface layout. I used some framed tabs so that it doesn't go off screen. Moreover I added an **Instructions** button and a **Help** line underneath so that the user doesn't get lost at the beginning. The structure is not too complicated:

- **Instructions**: quick guide for the users.
- **Rules**: setting the predecessor, successor and probability for the rule to happen.
- **Geometric** interpretation: attributes and parameters for the geometry.
- **Warnings** and helpline

1 Written by Przemyslaw Prusinkiewicz and Astrid Lindenmayer. Available online: algorithmicbotany.org/papers/abop/abop.pdf

Axiom	Initial word.
Depth	Recursive steps. Number of iterations over the same string.
Rule	Each production rule consists of 3 fields: the predecessor.
Angle (in degrees)	The turtle will yaw, roll or pitch by this angular amount each time it finds its corresponding symbol.
Segment Length	The length of the segments left by each turtle's step.
Segment Radius	The radius of the segments left by each turtle's step.
Length Attenuation	Next's segment's length will be **Length Atenuation**'s percent the length of the previous one.
Branches	Sets the colour for the branches.
Leaves	Sets the colour for the leaves.
Blossoms	Sets the colour for the blossoms.
Turtle speed	This option is useful for tracking each step the turtle does. Before proceeding to the next turtle command it will be frozen for this amount of time (in seconds).

Some images with different setup (included in the file as **Presets**):



The instructions for running the script through the GUI is quite simple:

1. Copy all the text inside startScript.py and paste it to Maya's Script Editor, a window will pop up asking you to select the folder which the script files are. Hit accept.
2. Set an axiom (or initial word), depth and rules.
3. Click Generate String. Then you will see the result in the text field below.
4. Set all the “Geometric Interpretation” attributes (Angle, Segment Length...)
5. Click Create Geometry. You will see the result in your scene. If you want to clean the last plant click Clean Plant. If you click Create Geometry again you will get another plant.

6. If you want you can set animation parameters under the tab Animation Settings, they are self-explanatory.

L-System String Rewriting <LS_string_rewriting.py>

This module is in charge to read the axiom and rewrite recursively over it **depth** numbers of times following some production rules. I had the chance to go for the recursion approach or just using loops, I decided to go the recursive way because I really want to understand it by practising.

L-System Interpreter <LS_interpret.py>

This module reads a string of characters and performs actions in function of what they are. This will follow the turtle method as it's the best way to represent a bunch of characters as geometric objects, therefore characters are treated as commands.

The turtle can be considered as an object or instance which moves in the three-dimensional space according to the executed commands. I will write below the interpretation which is given to any of the possible characters we have generated in the previous string (the one generated in LS_string_rewriting).

```
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)
[    Push current turtle state on the stack
]    Pop the current turtle state from the stack
```

The rest of letters (`AaCcDdEeGgHhIiJjKkMmNnOoPpQqRrSsTtUuVvXxYyZz`) will be interpreted as move forward as well.

The position and rotation are stored as parameters of the object instance, so that it is easily accessible. Both position and rotation will be updated each time a character is read.

Areas to improve

- Being able to construct a single geometry instead of hundreds of segments. The best example could be Jared Auty's project, in which he first creates a joints chain and builds a mesh around them afterwards.
- Instead of duplicating I could use instances. For the leaves, blossoms and segments. I think that would make the scene lighter.
- When using 3 variables (F, G and H for instance) with their rules, it has problems, I need to fix this and try to generate the 3D Hilbert Curve for instance.

Please, for more information take a look at the README files I have added to my project. They are under the “LSystemsMaya” folder. It is so worth it taking a look at it. I strongly recommend it.