L-Systems Mathematics Assignment

(code found in example_box folder)

Youtube Link:

https://youtu.be/0PgHmYq2p7M

Configuration

The config files specify:

- Length (length of any branch of the tree).

- Width (width of any branch of the tree).

- Axiom (the starting symbol for the parser to begin work on).

- Inc (the maximum number of increments that should be stepped through).

- Rules (the L-system rules that are applied in succession to the axiom).

Loading - (load_xml function)

The config files are loaded using the TinyXml 3rd party.

The library looks at the "type" attribute of each parameter in the config file and uses it to decide what to do with it. The majority of them are loaded in to memory as class variables.

The most interesting section of this function is the structure upon which the rules are placed; I use nested maps and vectors architecture to enable stochastic implementation as you will see later:

```
{
	F:[
		{
			F: 'FF'
		}
	],
	X:[
		{
			X: 'F[+X][-X]FX'
		},
		{
```

```
                X: 'F-[[X]+X]+F[+FX]-X'
            },
            {
                X: 'F[+X]F[-X]+X'
            },
            {
                X: 'X->FF'
            }
        ]
}
```

Parse

The parsing function steps through a loop building all of the possible iterations (up to the maximum increment count) and storing them in a vector.

It completes this task of building up the tree structure by looping through the axiom and checking each character against the possible rules.

This is where the stochastic techniques come in to play: If there are multiple possible rules that could be applied to the current character in the ruleset then it is randomly assigned a rule replacement - therefore creating a more dynamic and seemingly unique tree every time.

Render

The rendering function takes the current increment as an index to the increment_vector (which holds all of the possible combinations of tree structure for each increment) and then begins looping through the correct ruleset.

It then uses a switch case statement to evaluate what it should with each command from the ruleset:

- If it is an 'F' then we draw a line at the currentPoint vector. The end of this section saves the co-ordinates of the end of the section for reference when we draw the next branch in a variable called currentPoint. We use the length, width, angle specified earlier in the config unless it has been updated by the controls which we will get to later. We also use the default material unless it is changed.

- If it is a '+' or a '-' then we manipulate the angle and save it in a variable called currentAngle.

- If it is a '[' character then we save the currentAngle and currentPoint in a stack.

- If it is a ']' character then we load the last currentAngle and currentPoint from the stack in to memory and begin creating branches from that point and it is through this that we end up with the beautiful tree branching that we see once rendered.

Camera

The camera uses a simple method of taking the highest point of the tree (sceneHeightY variable) and then translating the camera back in the z direction by the same amount. This works because the fov of the camera is 45 degrees and therefore we know from the special properties of the half-isosceles triangle that the adjacent = opposite in pythagoras theorem

Controls

All controls re-render the tree immediately and are applied for all following iterations of the tree but are reset to the config initial variables when changing tree types.

F1, F2 – Increase and decrease the angle of the branches of the tree

F3, F4 – Increases and decreases the width of the branch segments.

F5, F6 – Increases and decreases the branch segment lengths, essentially growing or shrinking the tree.

F7, F8 – Changes the mode of the materials used; There are 3 present modes:

- Natural; During the render phase the function will check to see if the next character in the sequence is a ']' (the character that tells us that we are about to move back to a previous branch) and therefore knows that we are on a “leaf” section. It colours the leaves green and the rest of the tree brown.
- RGB random; During the render phase it randomly picks a red, green or blue colour and assigns that section a colour.
- Black; During the render phase this displays the simplest of the materials, just a solid black colour throughout.

Text

The text was rendered using the default mesh implemented in octet and simply displays the current tree configuration file and which increment we are currently on.