Coding L-Systems

Rewriting Rules

For example:

Axiom: F
Rule 1: F + F - - F + F

Applying rule 1 twice on the axiom gives us:

F + F - - F + F + F + F - - F + F - - F + F - - F + F + F + F - - F + F

F means move forward one unit and draw a line segment.

$+$ means turn left by an angle $\pi/3$,

$-$ means turn right by an angle of $\pi/3$.

So the string

- F - - F - - F

is a Koch snowflake.

Code the rewriting rule is as a recursive function.

```
drawbump(i) {
        if (i==0) {
              draw_line()
        } else {
              drawbump(i-1)
              turn_left()
              drawbump(i-1)
              turn_right()
              turn_right()
              drawbump(i-1)
              turn_left()
              drawbump(i-1)
        }
}
```

And an initial triangle is the function that calls the recursion:

```
drawflake(i) {
            initialize()
            turn_left()
            drawbump(i)
            turn_right()
            turn_right()
            drawbump(i)
            turn_right()
            turn_right()
            drawbump(i)
}
```

Branching structures

$$\begin{gathered} L \\ L \rightarrow F[-L][+L] \end{gathered}$$

The single L is the axiom, and there's one rule. Left "[" means "push" and right "]" means "pop". Say L is a leaf, and F is a branch. Then we can interpret the L-system graphically as a primitive plant.

To make it look nicer, we made the trunk get taller as the plant grows. A flexible way to incorporate scale is to use a function that scales the object coordinate system.

$$F \rightarrow RF$$

$$\begin{gathered} \textbf{Axiom}\text{: } L \\ L \rightarrow r\ F[-L][+L] \\ F \rightarrow RF \end{gathered}$$

Let's write this system in pseudocode. Here is the rule for L, the rule for F and the axiom:

```
drawleaf(i) {
        if (i==0) {
                actually_draw_leaf()
        } else {
                shrink()
                drawbranch(i-1)
                pushState()
                        turn_right()
                        drawleaf(i-1)
                popState()
                pushState()
                        turn_left()
                        drawleaf(i-1)
                popState()
        }
}

drawbranch(i) {
          if (i==0) {
                  actually_draw_branch()
          } else {
                grow()
                drawbranch(i-1)
        }
}

drawplant(i) {
        initialize()
        drawleaf(i)
}
```

Notice that the "actually_draw_branch" procedure changes the turtle position, while "actually_draw_leaf" procedure does not.

Various Stack Implementations

OpenGl Stack

```
    pushMatrix()
            turn_right()
            drawleaf(i-1)
    popMatrix()
```

with a call to a new function, for instance

```
    drawrightleaf(i)
```

The recursive function "drawrightleaf(i)" using the program recursion stack should look something like:

```
drawrightleaf(i) {
double[9] savedMatrix;
            copy(C,savedMatrix)
            turn_right()
            drawleaf(i-1)
            copy(savedMatrix,C)
}
```

Here we've designed "drawrightleaf()" *not* to change C. Notice that some L-system procedures *do* change C; in particular, "turn_left", "turn_right", "shrink", "grow" and "drawbranch", which cause a translation.