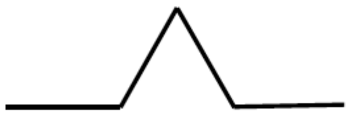
Drawing fractal plants with recursive functions
ECS 175 Spring 2012

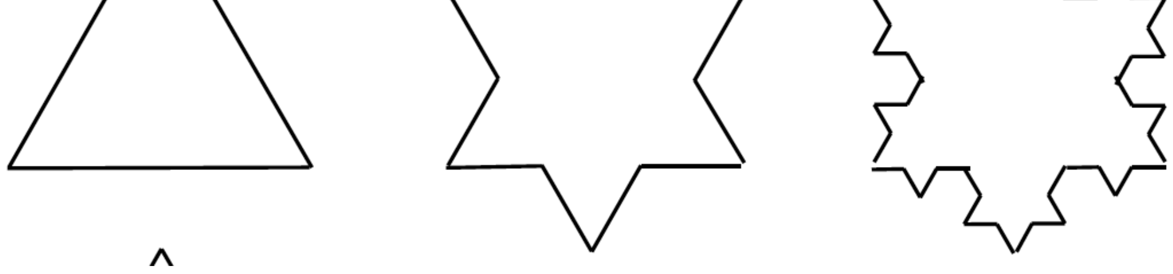
These notes describe a version of L-systems, a recursive language used to describe plant structures. L-systems were introduced by Arstid Lindendayer, a botanist - hence the L. The classic reference is a wonderful book by Lindenmayer and Prezemyslaw Prusinkiewicz (know to everyone as Dr. P), *The Algorithmic Beauty of Plants*.

1 Fractals

We can describe fractals through a re-drawing rule. For instance, consider the rule that replaces a straight line by a straight line with a triangular bump in it:



Starting with a triangle, we can apply the rule to all of its edges, and then do it again:



Applying the rule an infinite number of times produces a fractal called the Koch snowflake.

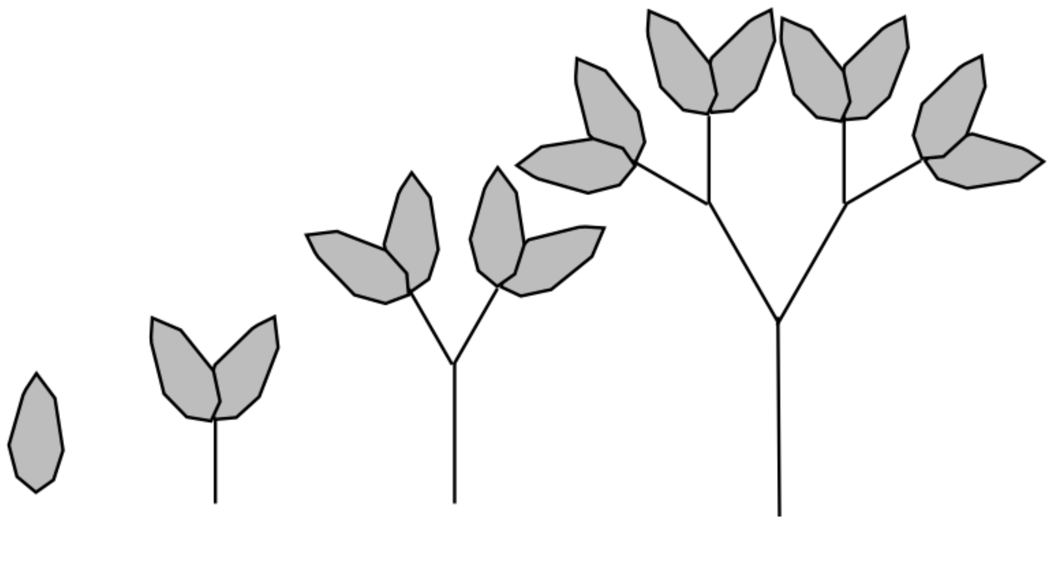
We can write this rule as a recursive function:

```
drawBump(i) {
    if (i==0) {
        drawLine()
    } else {
        drawBump(i-1)
        turnLeft(Pi/3)
        drawBump(i-1)
        turnRight(2 Pi/3)
        drawBump(i-1)
        turnLeft(Pi/3)
        drawBump(i-1)
    }
}
```

We can think of these functions as commands for a "turtle", who trails a line of paint behind itself as it moves along. The turtle has to move forward, turn left and turn right. More concretely, we can think of the turtle as a matrix. To move it forward, we left-multiply the turtle matrix it by a translation matrix. To make it turn, we left-multiply the turtle matrix by a rotation matrix. Every line that we draw gets multiplied by the turtle matrix to place it in the right position.

2 Branching and plants

To describe plants we really need to have branches. For example, in the picture below, we make a plant by replacing every leaf with a twig with two leaves on it. We don't think of applying the rule an infinite number of times for plants!



When we're doing branching, it helps to explicitly represent the turtle matrix, so that we can turn one way, and then go back to the branch point and turn the other way too. We can write something like this to draw the plant:

```
drawPlant(i, t) {
    if (i==0) drawLeaf(t)
    else {
        drawStem(i-1,t)
        temp = t
        turnLeft(t, Pi/10)
        drawPlant(i-1,t)
        t = temp
        turnRight(t, Pi/10)
        drawPlant(i-1,t)
        }
    }
```

You may have noticed that in the picture that the stem gets smaller as i decreases. How could you implement this?