CS354: Computer Graphics
Lecture Notes on L-systems
Feb 6, 2002

L-systems are a recursive language used to describe plant structures. They were introduced by Arstid Lindendayer, a botanist - hence the L. The classic reference is a book by Lindenmayer and Prezemyslaw Prusinkiewicz, *The Algorithmic Beauty of Plants*.

0.1 Rewriting rules

The main idea of L-systems is *string rewriting*. You may have seen this before in a course covering formal languages. String rewriting is done with *rewriting rules* (also sometimes called *productions*). For example:

$$F- > F + F - -F + F$$

Lets call this rule one. The $+$ and $-$ don't mean plus and minus, we just think of them as characters. The rule means that given a string containing an F, we must replace the F with the string on the right-hand-side, $F + F - -F + F$. So starting with the length one string:

$$F$$

we apply rule one and get

$$F + F - -F + F$$

The first single F is called the *axiom* of the L-system, and we say that the second string is generated from F by rule one. Applying rule one again gives us

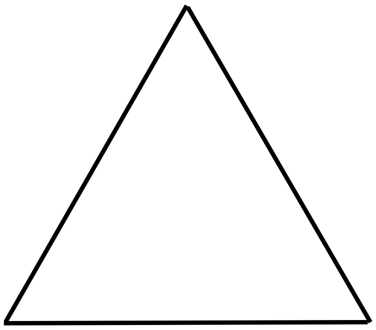
$$F + F - -F + F + F + F - -F + F - -F + F - -F + F + F + F - -F + F$$

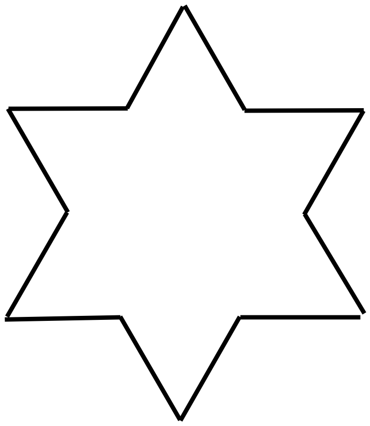
In L-systems, every symbol that can be rewritten gets rewritten simultaneously - we don't keep replacing the first F in the string, for instance. This is different from, for instance, context-free grammars.

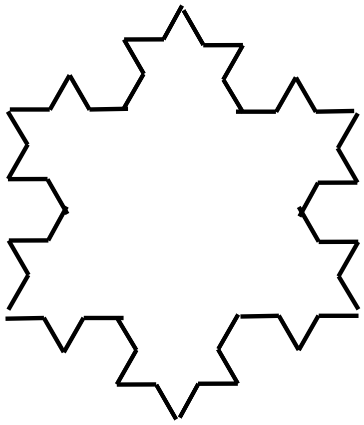
So what does this have to do with computer graphics? We give the string a graphical interpretation. This is usually described in terms of "turtle graphics". The "turtle" is a (hypothetical) cursor moving on the screen, drawing a line. F means move forward one unit and draw a line segment. $+$ means turn left by an angle δ, which we can pick arbitrarily to be say $\pi/3$, and $-$ means turn right by an angle of δ. So the string

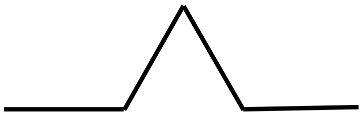
$$+F - -F - -F$$

Draws a triangle. The L-system we get by using rule one with this axiom is already interesting - see the figure below. Rule one replaces one edge - one F - with a "bump". One application of the rule makes a Jewish star. Subsequent applications give us the Koch snowflake fractal.









Another way to interpret the rewriting rule is as a recursive function.

```
drawbump(i) {
        if (i==0) {
              drawline()
        } else {
              drawbump(i-1)
              turnleft()
              drawbump(i-1)
              turnright()
              turnright()
              drawbump(i-1)
              turnleft()
              drawbump(i-1)
        }
}
```

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

```
drawflake(i) {
       initialize()
       turnleft()
       drawbump(i)
       turnright()
       turnright()
       drawbump(i)
       turnright()
       turnright()
       drawbump(i)
}
```

The "turn left" and "turn right" functions turn by an angle of $\pi/3$. The "draw line" function also translates the turtle to the end of the line segment it draws, since our model is that the turtle walks along every edge.

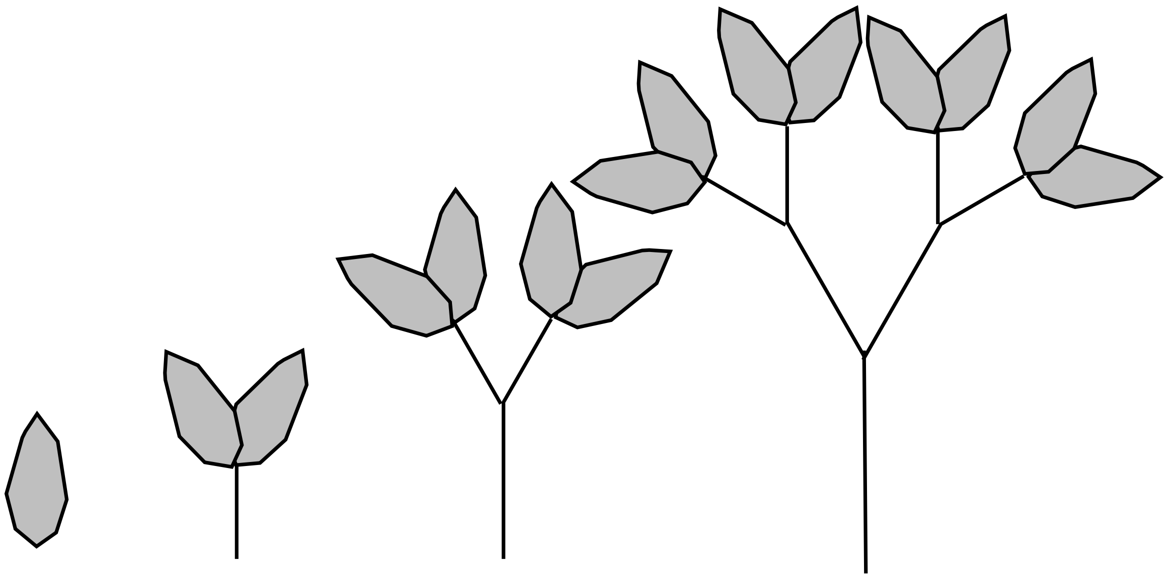
0.2 Branching structures

L-systems like this can make wiggly paths, but since we just have one turtle, moving along a curve, we can't get branching, which is pretty important for plants. To represent branching, we add a stack. For instance, to draw a tree we draw the trunk, push the state on the stack, draw one branch, pop the state, and draw the other branch. We use brackets to indicate pushing and popping in the notation. Here's an example of a simple L-system with brackets:

$$L$$

$$L- > F[-L][+L]$$

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 twig. Then we can interpret the L-system graphically as a primitive plant like the figure below.



To make it look nicer, we made the trunk get taller as the plant grows. Sometimes this is done by adding a second rule like:

$$F- > FF$$

This would mean that the length of every existing segment doubles at every iteration. A more flexible way to incorporate scale is to add a symbol for scaling the object coordinate system by some factor R. We should think of this as also scaling the turtle, so that it takes bigger steps. We can write this rule as:

$$F- > RF$$

In the picture above, R can be interpreted to mean "scale by $\sqrt(2)$", so that after two scalings the branches will be twice as long. But notice we have to scale down again as we draw the smaller branches. We can use another scaling symbol r which means "scale by $1/\sqrt{2}$". The L-system for the tree now looks like this:

$$Axiom : L$$

$$L- > rF[-L][+L]$$
$$F- > RF$$

Let's write this as a bit of recursive pseudocode.

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

drawtwig(i) {
        if (i==0) {
                actually_draw_twig()
        } else {
                grow()
                drawtwig(i-1)
        }
}

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

Notice that the "draw twig of length len" subroutine changes the turtle position (it corresponds to an "F"s in the string), while the "draw leaf" subroutine does not.

0.3 Implementation

Now we consider the implementation of drawing, turning and scaling.

We can think of the turtle as a point with a direction (where she is, where she's facing, and how big she is). This state information determines an object coordinate system in which drawing takes place. The turtle is like a personification of the object coordinate system. So our representation of the turtle is as a transformation matrix C.

OpenGL will take care of applying C to all the vertices of all the polygons that we draw automatically, if we load it into the OpenGL MODELVIEW matrix. The assignment includes some code which puts a 2D matrix into the MODELVIEW matrix properly. So all we need to do is maintain the correct 2D matrix as the turtle moves.

Assume the turtle has a standard initial state, say at $(0,0)$, and directed straight up, at angle $\pi/2$ from the horizontal. Since this is the initial state, we represent it by the identity matrix. Say we want to get her to position $(50,100)$ and have her facing angle $\pi/6$ (from vertical).

There are two equally good ways to think about how to get there: from the user's point of view, watching the turtle on the screen (world coordinates) and from the turtle's point of view, following the direction we give her (object coordinates).

Let's start with the user's point of view. We first rotate her around $(0,0)$ by $\pi/6$, and then translate $(0,0)$ to $(50,100)$. So we multiply every point in the vector at the initial state by the following matrix to get the vector representing the current state.

$$\begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\pi/6 & -\sin\pi/6 & 0 \\ \sin\pi/6 & cos\pi/6 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\pi/6 & -\sin\pi/6 & 50 \\ \sin\pi/6 & cos\pi/6 & 100 \\ 0 & 0 & 1 \end{bmatrix}$$

We'll use a shorthand matrix notation for this:

$$T_{(50,100)}\ R_{\pi/6}$$

Note that we write down the transformations in the order *opposite* that in which they are applied from the users point of view.

Now let's think of it from the turtle's point of view. To give her directions to get where she ought to be, we could tell her, " Walk to $(50,100)$, and then turn left by $\pi/6$." Note that in the turtle's view the transformations occur in the *same* order in which the matrices are written down.

To get a little more practice with the "turtle's eye view", let's consider asking her to more forward by an additional 30 units from her new poisition. This means we should *post-multiply* the matrix C by yet another transformation matrix:

$$\begin{bmatrix} \cos\pi/6 & -\sin\pi/6 & 50 \\ \sin\pi/6 & cos\pi/6 & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 30 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\pi/6 & -\sin\pi/6 & 50 - 30\sin\pi/6 \\ \sin\pi/6 & cos\pi/6 & 100 + 30\sin\pi/6 \\ 0 & 0 & 1 \end{bmatrix}$$

Our new matrix is the combination: $T_{(50,100)}\, R_{\pi/6}\, T_{(0,30)}$. How do we reconcile this with the user's point of view, where $T_{(0,30)}$ occurs before the other two transformations? To the user, the turtle first moves forward in the y direction by 30. Now we can think of the turtle as a new object, 30 units above the origin. Now all we have to do is get this new object to where the old object used to be, which we can do by applying the old transformation $T_{(50,100)}\, R_{\pi/6}$.