CMSC 23700 **Introduction to Computer Graphics** **Project 2**
Fall 2003 **October 30**

Plant synthesis

Due: Monday, November 17

1 Introduction

One of the xenobiology groups at NASA has finally found alien DNA hanging out on some meteor samples. Using Advanced Simulation Techniques^{TM} they have come up with some simple morphology patterns for what appear to be plant like structures that may be created using this alien DNA. Your job is to build a visualization tool to view the morphology of these lifeforms, and assist the xenobotany group in chosing which plant would be the nicest to grow as gifts for important congressional contacts.

2 Description

This assignment will require you to develop two systems. To begin, you will be provided with a system that generates strings[1] based on iterative application of string replacement rules (these are also known as *Lindenmayer systems* or L-systems, and are named after the biologist who originated this formalism for creating plant-like structures). You first need to develop a turtle graphics system that parses the strings generated by the L-system API into a *scene graph* that represents a three dimensional structures. The second step is to create a viewer for the scene graph.

2.1 L-Systems

Your friends at NASA have already provided you with an engine to create the strings you will feed to the turtle graphics system. This engine works by taking a seed string, and applying a set of rules that will replace certain characters with another string. In its simplest form, an L-system is a context free grammar where strings are generated by the *simultaneous* replacement of all nonterminals by the right-hand side of their productions. For example, given the grammar

$$F \quad ::= \quad |[+F]RR|[-F]+F$$

and the start string F, the first two iterations of the resulting L-system are as follows:

[1] Here we are using the term "string" in the formal language sense as a sequence of symbols. In the implementation it is represented as a linked list.

$$
\begin{aligned}
F &\rightarrow |[+F]RR|[-F]+F \\
&\rightarrow |[+|[+F]RR|[-F]+F]RR|[-|[+F]RR|[-F]+F]+|[+F]RR|[-F]+F \\
&\rightarrow |[+|[+|[+F]RR|[-F]+F]RR|[-|[+F]RR|[-F]+F]+|[+F]RR|[-F]+F]RR|[-|[+|[+F]RR| \\
&\quad [-F]+F]RR|[-|[+F]RR|[-F]+F]+|[+F]RR|[-F]+F]+|[+|[+F]RR|[-F]+F]RR \\
&\quad |[-|[+F]RR|[-F]+F]+|[+F]RR|[-F]+F
\end{aligned}
$$

Note that the characters "[," "|," and "]" are symbols in this system and not meta-characters. To better model alien plant-life, we extend this simple model of L-systems to allow arguments to be passed into rules, to allow rules to be conditional on the values of its arguments, and to allow probabilities to be associated with rules. The syntax of an L-system specification is as follows:

File
::= *Definition** *Start Rule**

Definition
::= **`define`** Variable **`=`** Expr **`;`**

Start
::= **`start :`** *Module*$^+$ **`;`**

Rule
::= *ModuleName Params*^{opt} *RHS*$^+$ **`;`**

Params
::= **`(`** *Variable* (**`,`** *Variable*)* **`)`**

RHS
::= *Condition*^{opt} **`->`** *Probability*^{opt} *Module**

Condition
::= **`:`** *Expr*

Probability
::= **`(`** Number **`)`**

Module
::= *ModuleName*
| *ModuleName* **`(`** *Expr* (**`,`** *Expr*)* **`)`**

A specification consists of a sequence of definitions, followed by a start string, followed by a list of rules for expanding modules.[2] In its simplest form, a rule has a left-hand-side module and a right-hand-side expansion. This form can be refined by adding parameters, a condition (*i.e.*, the rule only applies if the condition is true), and a probability. Modules on the right-hand-side of a rule can

[2]The term "*module*" is L-System terminology for a grammar symbol.

take arguments, which are written using the following simple expression language:

Expr
::= *Expr* **|** *Expr*
| *Expr* **&** *Expr*
| *Expr* **=** *Expr*
| *Expr* **!=** *Expr*
| *Expr* **<=** *Expr*
| *Expr* **<** *Expr*
| *Expr* **>=** *Expr*
| *Expr* **>** *Expr*
| *Expr* **+** *Expr*
| *Expr* **-** *Expr*
| *Expr* ***** *Expr*
| *Expr* **/** *Expr*
| *Expr* **^** *Expr*
| **-***Expr*
| **(** *Expr* **)**
| Variable
| Number

Expressions are computed using floating-point arithmetic; boolean conditions are represented using `0.0` to represent false. The "^" operator is exponentiation.

Module names are single characters and can either be letters or one of a small collection of special characters.

ModuleName
::= *Symbol*
| *Letter*

Symbol
::= **+** | **-** | **^** | **&** | **** | **/** | **|** | **$** | **[** | **]** | **!** | **'** | ***** | **%**
Letter
::= a | b | c | . . . | x | y | z | A | B | C | . . . | X | Y | Z

As is discussed below, certain module names have special pre-defined meaning. Using this syntax, the example from above might be specified as follows:

```
define maxgens = 3;
define delta = 16.8;
start : F;
F -> |[+F]RR|[-F]+F;
```

The definition of the variable `maxgens` controls the number of iterations and the variable `delta` specifies the default angle for rotations.

NASA has provided a library to read and evaluate these L-Systems. The interface to this library has the following functions:

```
LSystem_t *LoadLSystem (const char *file);
Module_t *EvaluateLSystem (LSystem_t *lsys, int nGens);
void FreeModules (Module_t *list);
```

The `LoadLSystem` function reads an L-system description from a file and returns it. This function returns a null pointer if there is an error. The `EvaluateLSystem` function evaluates the L-system for *nGens* iterations If *nGens* ≤ 0, then the value of the variable `maxgen` is used. starting with the start string given by the L-system specification. It returns a newly allocated linked list of module instances. The final function is used to free the list of modules returned from `EvaluateLSystem`.

2.2 Turtle Graphics in the Third Dimension!

While growing strings alone may interesting to some people, we are not those people! Your mission is to give life to the strings generated by the L-System module by interpreting each character as a command to a "magic" anti-gravity turtle. Not only can our simulated turtle hover and move in arbitrary three space, but much like Sesame Stree characters it can draw in mid-air.

The turtle's state is managed using a state vector that keeps track of the turtle's current position, orientation, branch width, and drawing color. The turtle's position and orientation define its local coordinate system, which is used to interpret turtle commands. In addition to its current state, the turtle has a stack on which it can push and pop its state. You are allowed to manage turtle orientation in any fashion that generates correct visualizations.

The convention is that in the turtle's coordinate system, the Z axis points forward, the X axis points left, and the Y axis points up. Rotations around the Z axis are called rolls (rolling left is a counterclockwise rotation and rolling right is a clockwise rotation). Rotation around the X axis is called *pitch* — pitching down is a clockwise rotation, while pitching up is counterclockwise. Finally, rotation around the Y axis is called *turning*[3] — turning left is a clockwise rotation and turning right is a counterclockwise rotation.

The turtle begins life at the world origin, $(0, 0, 0)$, with its forward direction being along the world's positive Y-axis, $(0, 1, 0)$ and up being along the world's negative Z-axis, $(0, 0, -1)$ (see Figure 1). Thus the turtle's initial Z axis corresponds to the world's Y axis, the initial Y axis corresponds to the world's negative Z axis, and the turtle and world share a common X axis. The initial branch width is 0.1 and the initial color is white.

Table 1 shows how each module in a turtle graphics input string is to be interpreted. The first column gives the L-System module, the second is the name of the command as exported by the L-System API, and the third column is the description of the command. Characters not listed on the table should be silently ignored by your turtle. Note that some modules (*e.g.*, "**F**" and "**+**") have both parameterized and unparameterized forms. The unparameterized forms are just shorthand for default values, which are filled in by the L-System evaluator. Although the turtle does not see the

[3] In airplanes, the angle around the Y axis is called the *yaw*.

Table 1: Turtle commands

Module	Command	Description
F	DRAW	move forward one unit while extruding a cylinder of the current width.
F(l)	DRAW	move forward l while extruding a cylinder of the current width.
f	MOVE	move forward one unit without drawing
f(l)	MOVE	move forward l without drawing
+	LEFT	turn left
+(d)	LEFT	turn left by d degrees
-	RIGHT	turn right
-(d)	RIGHT	turn right by d degrees
|	FLIP	turn around
^	UP	pitch up
^(d)	UP	pitch up by d degrees
&	DOWN	pitch down
&(d)	DOWN	pitch down d degrees
$	REVERSE	Rotate turtle to vertical in the world coordinate system.
****	ROLLLEFT	roll left
****(d)	ROLLLEFT	roll left by d degrees.
/	ROLLRIGHT	roll right
/(d)	ROLLRIGHT	roll right by d degrees.
[	PUSH	Start a branch by pushing the current state on the stack.
]	POP	Complete a branch and restore the state by popping it off the stack.
!	WIDTH	Multiply the current width factor by 0.9
!(w)	WIDTH	Set the current width to w.
'(r, g, b)	SETCOLOR	Set the current color
*****(r, g, b)	MULCOLOR	Modulate the current color
C(r_1, r_2, h)	CONE	Draw a truncated cone of height wh, base radius wr_1, and top radius wr_2 at the current location, where w is the current width.
S(r)	SPHERE	Draw a sphere with radius wr at the current location, where w is the current width.
L(w', h_1, h_2, t)	LEAF	Draw a leaf with width ww', minor height wh_1, major height wh_2, and thickness wt, where w is the current width. Leaves are composed of 8 triangles arranged as described in Section 2.3.
%	*n.a.*	truncate a branch. Occurrences of this module are eliminated in EvaluateLSystem and so should not occur during drawing.

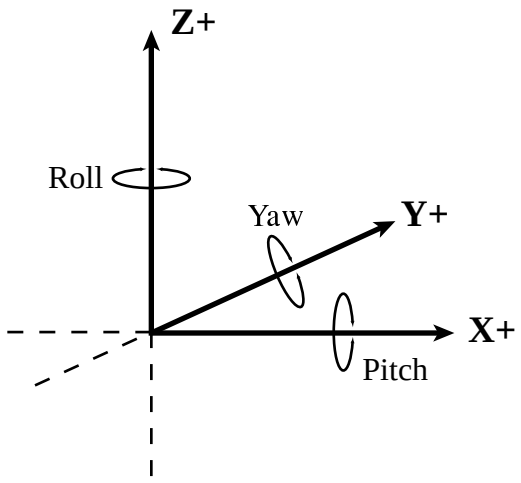

Figure 1: Initial turtle orientation and rotation axes.

default forms, we give the translation for the sake of completeness:

$$
\begin{aligned}
\texttt{F} &= \texttt{F}(1.0) \\
\texttt{f} &= \texttt{f}(1.0) \\
\texttt{+} &= \texttt{+}(\texttt{delta}) \\
\texttt{-} &= \texttt{-}(\texttt{delta}) \\
\texttt{\^{}} &= \texttt{\^{}}(\texttt{delta}) \\
\texttt{\&} &= \texttt{\&}(\texttt{delta}) \\
\texttt{\textbackslash} &= \texttt{\textbackslash}(\texttt{delta}) \\
\texttt{/} &= \texttt{/}(\texttt{delta}) \\
\texttt{!} &= \texttt{!}(0.9)
\end{aligned}
$$

2.3 Graphical objects

The turtle draws three basic shapes: truncated cones, spheres, and leaves.[4] For the cones and leaves, these are drawn with the major axis extending in the forward direction (positive Z-axis) of the turtle's coordinate system). Spheres are drawn with the center at the current turtle position.

Leaves are composed of eight triangles. The size of a leaf is controlled by four parameters: the width w, the minor height h_1, the major height h_2, and the thickness t. Figure 2 shows how these parameters are interpreted.

2.4 Scene Graphs

Great, so we have this wacky little state vector wandering all over three space, what good does it serve? Well, the answer is that your turtle sytem should be generating a scene graph while the turtle is trundling about.

[4]It also draws cylinders, but a cylinder is just a trucated code where the bottom and top radii are the same.

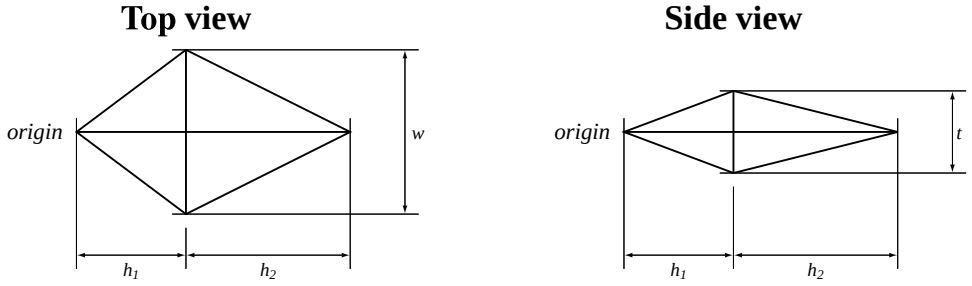

Figure 2: Leaf dimensions

Your scene graph should support creation of the following abstract elements:

- $\mathbf{Union}(elem_1, elem_2)$ — Luckily, our L-systems do not require all CSG operations, and you will only need to provide some Union element for branching from the current point in the scene.
- $\mathbf{Cone}(r_1, r_2, h)$ — Draw a cone in the current coordinate space along the positive Z-axis with initial radius r1, final radius r2, and of height h.
- $\mathbf{Sphere}(r)$ — Draw a sphere of radius r in the current coordinate space, centered at the turtle's origin.
- $\mathbf{Leaf}(w, h_1, h_2, t)$ — Draw a leaf with width w, minor height h_1, major height h_2, and thickness t. Leaves are composed of 8 triangles arranged as described in Section 2.3.
- $\mathbf{Color}(r, b, g, elem)$ — Change the current color of child elements to $\langle r, b, g \rangle$

You will also need to develop transformations. You should remember that the following are abstractions, and your actual implementation may choose to render the following as a single transformation node, or as a property of the current object in the graph.

- $\mathbf{RotateX}(d, elem)$ – Rotate all child elements d degrees about the X axis.
- $\mathbf{RotateY}(d, elem)$ – Rotate all child elements d degrees about the Y axis.
- $\mathbf{RotateZ}(d, elem)$ – Rotate all child elements d degrees about the Z axis.
- $\mathbf{Translate}(d_x, d_y, d_z, elem)$ – Translate all child elements by the $\langle d_x, d_y, d_z \rangle$ vector.

Once you have constructed the scene graph representing the plant, you need to render it. Your viewer can be a modification of the program you wrote for Project 0 and will support simple navigation. It should render the plant on a flat surface (*i.e.*, a rectangle located in the XZ-plane) and it should rotate the plant at a rate of 18° per second (*i.e.*, one complete rotation every 20 seconds).

2.5 Shadows

Your viewer should provide a flat surface to place your plant on (*i.e.*, a rectangle located in the XZ-plane). To make your image more realistic, you should render the shadows it casts. We will discuss real-time shadowing algorithms in class.

3 User Interface

Your program should take an L-System specification-file as a command-line argument. It should load the file and the startup a viewer for the generated plant. Your viewer should support at least the following keyboard commands:

space	toggle rotation on/off
`+`	grow the plant for one more generation
`-`	grow the plant for one fewer generation
`w`	move the camera 0.05 units toward the "look-at" point.
`s`	move the camera 0.05 units away from the "look-at" point.
`a`	rotate the view location to the left 1°.
`d`	rotate the view location to the right 1°.
`l`	toggle the directional light.
`q`	quit the viewer

Feel free to add other commands and mouse-based navigation.

4 Requirements

As with the previous projects, we will create a module in your course CVS repository on the Computer Science CVS server. The module is named `project-2` and contains the implementation of the L-System. Your task is to design and implement a scene-graph representation; write an interpreter for the turtle that translates strings to scene graphs, and implement a viewer for your scene graph representation that supports shadows. As before, submission will be via CVS.

5 Document history

Nov. 19 Fixed UI description.

Nov. 9 Fixed description of the relationship between the turtle's inital orientation and world coordinates.

Nov. 4 Added description of leaf nodes to scene graph discussion.

Nov. 2 Changed grammar for L-Systems slightly.

Oct. 30 Original version.