

SIMULATING PLANT MOTION
WITH LEVELS OF DETAIL

A Senior Honors Thesis

by

REBECCA LYNN FLANNERY

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOWS

April 2003

Group:

Engineering & Physics 1

SIMULATING PLANT MOTION
WITH LEVELS OF DETAIL

A Senior Honors Thesis

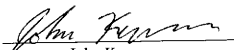
by

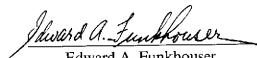
REBECCA LYNN FLANNERY

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOWS

Approved as to style and content by:


John Keyser
(Fellows Advisor)


Edward A. Funkhouser
(Executive Director)

April 2003

Group: Engineering & Physics 1

ABSTRACT

Simulating Plant Motion With
Levels of Detail. (April 2003)

Rebecca Lynn Flannery
Department of Computer Science
Texas A&M University

Fellows Advisor: Dr. John Keyser
Department of Computer Science

Levels of detail are an effective means of reducing computational power in three-dimensional computer animations. Simpler models are used to represent objects farther away from the camera. Without this simplification, the processor may get bogged down with calculations, resulting in a considerable drop in the speed of the animation. This slowdown is undesirable, especially when dealing with interactive animations.

The purpose of this research is to simplify a model of a tree represented using a Lindenmayer system, or L-system. To generate a tree, an L-system takes a set of parameters, such as branch length, and iterates through those parameters a given number of times. The traditional approach to simplifying objects by reducing the number of polygons will not work on such a parameter based model. Further complications arise from the fact that the trees being animated here are not static models; they are constantly in motion due to a simulated wind. A suitable method for simplification must reduce the number of iterations and adjust the given parameters in order to keep both the overall shape, size, and motion of the tree close to the original full size model.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	v
INTRODUCTION	1
PREVIOUS WORK	3
LINDENMAYER SYSTEMS	5
CALCULATING BRANCH MOTION	8
Bend Angle	8
Rotation Angle	10
Force	12
SIMPLIFICATION OF A GIVEN L-SYSTEM	15
RESULTS	17
CONCLUSIONS	18
REFERENCES	19
VITA	20

LIST OF FIGURES

FIGURE	Page
1 Screen shot of plant and wind simulation software	3
2 Branch level L-system parameters	6
3 Tree level L-system parameters	7
4 Comparison of simplified trees with original	17

Modeling every polygon in the aforementioned scene would be computationally very expensive, if not impossible. Levels of detail, or LODs, are a commonly used solution to the problem of rendering complex scenes. An object close to the viewer appears large on the screen, while an object far from the viewer appears small. This farther, smaller object may take up only a fraction of the screen. In this case several polygons may map to each pixel. The farther away the object is from the viewer, the smaller it will appear. Thus, calculating how to display every polygon wastes computing power. LODs display a simplified version of the object, which can be rendered faster.

Each LOD corresponds to a range of distances from the viewer. At close distances, say closer than 1 meter, the object will appear in full detail. At the next LOD, say between 1 meter and 5 meters away, the object will be simplified. At the next LOD, the object will be simplified even more. This simplification continues at each LOD. Note that the programmer determines the number of LODs in a scene as well as the distance ranges.

In this project, we are simulating wind blowing through trees. Our overall goal is to simplify the computation power required to run the simulation on trees farther away from the viewer. On simulations such as this, the biggest slowdowns come from the sheer number of calculations required to run the simulation. The purpose of LODs in simulation, then, is not to reduce the number of polygons but to reduce the number and/or complexity of the equations that must be solved at each time step.

PREVIOUS WORK

Previous work on this project was performed by Jacob Beaudoin and Nathan Monteleone. Monteleone created an application to read in an L-System as a text file and output a three-dimensional model of the tree structure. The software also had the capability to simulate a simple wind blowing on the tree. In this case, the wind was restricted to a constant velocity and direction. The software was written in C++, the graphics were displayed using OpenGL, and the graphical interface was designed using the functionality of GLUT.

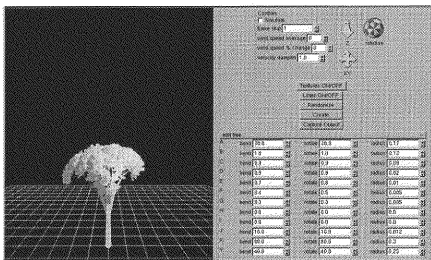


Fig.1. Screen shot of plant and wind simulation software. This application was created by Monteleone. It was used throughout the project to gather data on the various features of each given L-system.

Beaudoin then modified the application, giving it a few more features and making the wind simulation more accurate. His work enabled the user to interactively modify the wind speed while the application was running. He also added a feature to

add a bark texture to the branches and a leaf texture to the leaves. In addition to simulating the force on each branch as propagated down from the leaves, he added an additional force from the wind resistance on the branch itself.

Much research has been done on LODs, especially on polygonal mesh simplification. Research on LODs involving animation is scarcer but still available. For example, Schödl et al. [2000] used “video textures” to create animations of repetitive and quasi-repetitive motions like a runner, a pendulum, and a candle flame. They had problems creating more complex motion like grass blowing in the wind and waves on the beach. While there is also a good body of research on modeling individual plants, there has not been as much research on modeling large groups of plants. Deussen et al. [1998] modeled a realistic plant scene, but their method applies only to static images. Weber and Penn [1995] used a method to model the sway of trees and large grasses in light to moderate winds. Their method also incorporates LODs to accommodate a viewer moving through the scene, but it is not real-time for large numbers of trees. In addition, their algorithm for making the trees sway in the wind is not physically based.

LINDENMAYER SYSTEMS

A Lindenmayer system, or L-system, uses a set of rules and symbols to iteratively describe some structure. L-systems were introduced in 1968 by Aristed Lindenmayer with the goal of describing the development of simple multicellular organisms. Later the abilities of L-systems were extended to be able to describe higher plants and complex branching structures. Prusinkiewicz and Lindenmayer [1990] describe the workings of L-systems as “defining complex objects by successively replacing parts of a simple initial object using a set of *rewriting rules* or *productions*,” and they note that at each time step all of the productions are applied simultaneously.

All L-systems consist of an axiom and a finite set of productions. They are commonly implemented using strings, in which case it is easy to interpret the axiom as the basic input string of the L-system. The productions define rules for replacing symbols in the axiom with other strings of symbols. At each iteration, the same productions are run on the newly produced string from the previous step. In this way, a successively longer string, and thus a more complicated structure, is built.

With the addition of special symbols into the alphabet of the system, the strings generated by an L-system can be interpreted geometrically. The regular alphabet symbols would be used to draw a straight line. The special symbol + can be interpreted to mean, “rotate counterclockwise by the given predefined angle.” The complement to this symbol is - which means, “rotate clockwise by the given predefined angle.” Brackets, [and], can also be added to the alphabet in order to create subgroups of

symbols. Using a combination of these special symbols allows us to use a textual representation of an L-system in order to produce a visual representation of a tree.

One important thing to note about L-systems is that in order to be useful, the regular alphabet symbols used in the construction of the L-system must have meaning. In our tree representation, each symbol has associated with it a length. This information is vital or the program would have no idea how long to make the branches. In fact each level of branches has several parameters associated with it. Aside from length, each branch also has a defined radius. Bend strength and rotation strength are also applied to individual levels of branches. These two parameters have no effect on the structure of the tree; rather they affect how the tree reacts to the wind. Bend strength determines how hard a tree resists bending. Lower bend strength leads to a more “wobbly” branch, and higher bend strength leads to a stiffer branch. Similarly, rotation strength determines how strongly a branch resists rotating around its parent branch. These parameters are summarized in Figure 2 below.

Parameter Name	Description
Radius	Radius of the branch
Length	Length of the branch
Bend Strength	Factor defining resistance to bending in relation to the parent branch
Rotation Strength	Factor defining resistance to rotation about the primary axis of the parent branch

Fig. 2. Branch level L-system parameters.

In addition to the parameters for each branch, there are a few parameters defined for the tree as a whole. Derivation length is the most obvious of these parameters. It affects the overall size and complexity of the tree by determining how many times the L-system is iterated through in order to generate the complete tree structure. Branching angle is also applied to the whole tree. Branching angle determines how much each branch is bent off of its parent. Rotation angle is another parameter that applies to the whole tree. It determines how much each child branch is rotated around the axis of its parent. These parameters are summarized in the table below.

Parameter Name	Description
Derivation Length	Number of iterations
Branching Angle	Angle at which a child branch is bent in relation to its parent branch
Rotating Angle	Angle at which a child branch is rotated about the primary axis of its parent branch

Fig. 3. Tree level L-system parameters.

CALCULATING BRANCH MOTION

As stated before, the goal of this project is to simplify tree models by reducing the number of calculations, which must be done. Reducing the derivation length reduces the number of branches, which in turn reduces the number of calculations the application must compute. Simply reducing the derivation length is not enough, however. Not only will the motion of the tree be different but also will the overall size of the tree. A whole new L-system must be constructed to produce the simplified tree model. The challenge lies in trying to figure out how to assign values to parameters of the new L-system in order to make the motion similar to that of the tree generated by the original L-system.

Ideally we would like the swaying of the simplified tree to match up fairly well with that of the original tree. That is, the angles that each branch makes in relation to its parent branch should match visually at each time step. To discover if there is some sort of algorithm for designing the new L-system, we start by looking at the equations that govern the tree motion. The overall angle that each branch makes in relation to its parent is determined by the bend angle and the rotation angle. The equations for determining these values are very similar to each other.

Bend Angle

The bend angle is calculated by incrementing the bend angle from the previous time step. The size of the increment is determined by multiplying the bend velocity by the length of the time step. Bend velocity is simply a measure of how fast the branch is

bending. It can have positive or negative values. The bend angle for a branch at the current time step is given by the following equation:

$$\theta_b = \theta_{b-1} + v_b * \Delta t$$

where:

θ_b = bend angle for current time step

θ_{b-1} = bend angle for previous time step

v_b = bend velocity

Δt = length of time step

Bend velocity changes at each time step, and so must be recalculated each time. Like bend angle, the bend velocity for the current time step is calculated by incrementing the bend velocity from the previous time step.

Here we introduce absorbed force and propagated force. Each branch does not absorb 100% of the force applied to it. Rather, only a fraction of the force is absorbed, and the rest is propagated to its parent branch. Simply put, the absorbed force is what causes a branch to move; the propagated force is what causes the branch below it to move. This propagated force is what causes the whole tree to sway, instead of just the branches with a leaf on the end.

To increment the bend velocity, the x component of the force vector is multiplied by the constant fraction representing the percentage of force absorbed. The resulting number is then added to the previous bend velocity. The bend strength of the branch, as defined in the L-system text file, reduces the bend velocity slightly by making the branch

stiffer. The bend velocity for a branch at the current time step is given by the following equation:

$$v_b = v_{b-1} + F_x * k_{FA} - \theta_{b-1} * \frac{k_{bs}}{1000}$$

where:

v_b = bend velocity for current time step

v_{b-1} = bend velocity for previous time step

F_x = x component of the calculated force vector

k_{FA} = percentage of the force absorbed by the branch

θ_{b-1} = bend angle for previous time step

k_{bs} = bend strength of the branch

Rotation Angle

The equations for rotation angle are very similar to the equations for bend angle.

The rotation angle is calculated by incrementing the rotation angle from the previous time step. The size of the increment is determined by multiplying the rotation velocity by the length of the time step. Rotation velocity can have positive or negative values.

The rotation angle for a branch at the current time step is given by the following equation:

$$\theta_r = \theta_{r-1} + v_r * \Delta t$$

where:

θ_r = rotation angle for current time step

θ_{r-1} = rotation angle for previous time step

v_r = rotation velocity

Δt = length of time step

Like bend velocity, rotation velocity must be recalculated at each time step by incrementing the rotation velocity from the previous time step. To increment the rotation velocity, the y component of the force vector is multiplied by the constant fraction representing the percentage of force absorbed. Notice the difference here between bend velocity and rotation velocity. Bend velocity is calculated using the x component of the force vector, while rotation velocity is computed using the y component of the force vector. The resulting number is then added to the previous rotation velocity. The rotation strength of the branch, as defined in the L-system text file, reduces the rotation velocity slightly by making the branch stiffer. The rotation velocity for a branch at the current time step is given by the following equation:

$$v_r = v_{r-1} + F_y * k_{FA} - \theta_{r-1} * \frac{k_{rs}}{1000}$$

where:

v_r = rotation velocity for current time step

v_{r-1} = rotation velocity for previous time step

F_y = y component of the calculated force vector

k_{FA} = percentage of the force absorbed by the branch

θ_{t-1} = rotation angle for previous time step

k_{rt} = rotation strength of the branch

One other thing to note about these equations is the initial values of the parameters. The force has a value of zero at time t_0 simply because the wind is not blowing at the beginning of the simulation. The values for the following parameters are all set to zero at time t_0 . The values for bend velocity and rotation velocity are also zero in the beginning because of the lack of motion. The other parameters that are initially set to zero are θ_b and θ_r .

Force

The driving factor in the simulation is the force on the tree generated by the wind. In our simulation, force is calculated for both leaves and branches. The equation for force on a leaf is fairly simple. It takes into account the surface area of the leaf, the speed of the wind, and the direction of the wind. Obviously a leaf with a larger surface area is going to have more force on it. The same goes for faster wind speed. The direction of the wind relative to the leaf also makes a big difference in how much force the leaf gathers. If the wind is hitting the leaf head on, in other words if the normal to the surface area of the leaf is parallel to the wind direction, then the leaf will absorb maximum force. If the leaf is at an angle to the wind, it will absorb less force. The force on a leaf is calculated by the following equation:

$$\vec{F} = \hat{N} * (\vec{V} \bullet \hat{N}) * A$$

where:

\hat{N} = unit vector normal to the surface of the leaf

\vec{V} = wind vector in the local coordinate system

A = surface area of the leaf

The force on a branch comes from two places. A small part of the force comes from the force of the wind blowing directly on the branch. The majority of the force is the force propagated from its children branches or, if the branch has a leaf attached, the force propagated from its child leaf. At with the force on the leaf, the force on a branch depends on the surface area of the branch and the angle at which the wind is blowing relative to the branch. Since the branch is round, not flat like a leaf, it absorbs much less force than a flat surface would. A coefficient of drag is introduced to determine what fraction of the force the branch absorbs directly from the wind. For purposes of this project, the coefficient of drag has been set to 0.0015. The equation for force on a branch is as follows:

$$\vec{F} = \vec{F}_p + \hat{V} * v^2 * A * \sin(\theta) * k$$

where:

\vec{F}_p = propagated force

\hat{V} = normalized wind vector in the local coordinate system

v = wind speed

A = surface area of the branch

θ = angle between the branch and the wind vector

k = coefficient of drag

The propagated force is the force that is carried down to a branch from its children nodes. This force is some fraction of the total force acting on the children nodes. If a branch has a leaf on it, then the propagated force comes directly the force acting on the leaf. If the branch has other branches coming off of it, then the propagated force is derived from the sum of the forces acting on all of those children branches. The equation for the propagated force is as follows:

$$\vec{F}_p = k_{fp} * \sum_{i=1}^n \vec{F}_i$$

where:

k_{fp} = percentage of the force propagated from the children

\vec{F}_i = force acting on child node i

n = number of children nodes

SIMPLIFICATION OF A GIVEN L-SYSTEM

The method we have developed to simplify a tree defined by a given L-system is based on a single concept: combine two branches into one. The motion of the new branch must adequately resemble the motion of the old branches. We measure this by comparing the angle the new branch makes with the combined angle. These angles should be the same at any time throughout the simulation. This requirement ensures that not only the angles of the new tree correspond to the original tree but also the period of the tree's "sway."

It should be noted that several assumptions were made about the original L-system before the simplified L-system was created. First, all trees had only one branch coming off each parent branch. Furthermore, no child branch was bent initially in relation to its parent branch. In essence, the original L-system described a stalk that went straight up. In addition each branch throughout the original tree had the same length, radius, bend strength, and rotation strength. The final restriction was on the number of branches, which were limited to be powers of two. For example, we took an original tree model with 8 branches, simplified it to one with 4 branches, and then simplified it even further to a model with only two branches. Doubtless these restrictions will not completely coincide with the simplification method we found. The idea is to extend the current system to encompass the lifting of these restrictions one at a time.

To continue with the simplification process, we are trying to match the bend angle of the simplified model to the overall angle made by two smaller branches in the original model. Substituting the equation for bend velocity into the equation for bend angle generates the following equation for bend angle:

$$\theta_b = \theta_{b-1} + \left(v_{b-1} + F_x * k_{FA} - \theta_{b-1} * \frac{k_{bs}}{1000} \right) * \Delta t$$

In this equation, k_{FA} , k_{bs} , and Δt are constants. θ_{b-1} and v_{b-1} are based on previous time steps. The only variable left that we have control over is the force. Thus in order to make the new branch have the same combined angle as the two original branches, it must have the same combined force.

Figuring out how to make the force match is a bit harder because there are more parameters to solve for. Following is a summary of the method used to simplify the L-system:

1. Create a new branch with length equal to the sum of the lengths of the two branches being replaced.
2. Adjust the propagated force constant to be $k_{FPnew} = k_{FP}^{n+1}$, where n represents the level of simplification, and the original tree would have $n=0$.
3. Adjust k_{bs} and k_{rs} to correct the period of the branch's swaying.
4. Adjust the force coming from the leaf to correct the amplitude of the sway.

Currently the values in steps 3 and 4 are determined experimentally.

RESULTS

The method developed herein seems to work fairly well. It works especially well at low speeds. As seen in Figure 4 below, it is possible to create an L-system that matches the performance of the original tree almost exactly. It does develop some problems as the wind speed increases, though. Some of the factors affecting the amplitude and periodicity of the swaying motion of the branches that aren't apparent at lower wind speeds start to come in to play at higher wind speeds.

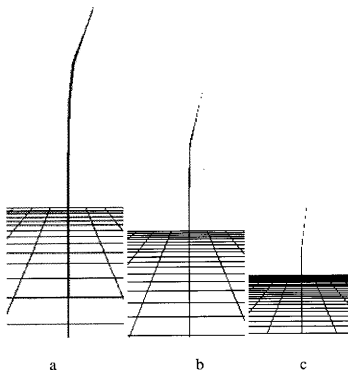


Fig. 4. Comparison of simplified trees with original. The original tree (a) has eight branches. The middle picture (b) shows the tree simplified by one level. It has four branches and is viewed from farther away. The picture on the right shows (c) the tree simplified down to two branches and seen from still further away. At pictures were taken at time $t=4$.

CONCLUSIONS

The method we have developed is useful but only on a limited scale. It does have the possibility of being developed further to handle a wider variety of plants. Specifically, we would like to get it to a place where it handles plants that are commonly recognized as “trees.” There are several possible extensions and improvements of the system described herein that might be implemented in the future. The most obvious one is to extend the system to handle multiple branches coming off of a single branch. This extension would allow more tree-like plants to be simulated, instead of just the stalk like structures implemented here. Calculating the force on each simplified branch becomes a much more difficult task because the propagated force comes from multiple sources.

Before this extension can be implemented, however, work must be done to evaluate how the system reacts with branches that are bent at their “rest” position. Currently, all branches are in the same direction as their parent branch. Multiple branches must be bent by default, or all child branches would occupy the same physical space.

Other extensions that might be investigated are the length of the simplified branch and randomized branch lengths. Currently the length of the simplified branch is the sum of the two smaller branches. Randomized branch lengths present many problems. The randomization algorithm must not be truly random. It must be repeatable if a suitable simplification is to be found.

REFERENCES

- Beaudoin, J. 2002. Wind Motion Through Simple Trees.
- Deussen, O., et al. 1998. Realistic Modeling and Rendering of Plant Ecosystems. *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. 275-286.
- Prusinkiewicz, P. and Lindenmayer, A. 1990. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York.
- Schödl, A, Szeliski, R., Salesin, D., and Essa, I. 2000. Video Textures. *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. 489-498.
- Weber, J. and Penn, J. 1995. Creation and Rendering of Realistic Trees. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. 119-128.

VITA

Rebecca Lynn Flannery

5443 Rutherglenn

Houston, TX 77096

Education

B.S. Computer Science Texas A&M University May 2003

Honors

TAMU Distinguished Student in engineering.

TAMU/Barnes & Noble Academic Excellence Scholarship.

Texas A&M President's Endowed Scholarship.

National Merit Scholar

Experience

Computer Science Helpdesk Technician, Texas A&M University, 2002-Present

Intern/Research Assistant, University of Utah, 2001, 2002

Intern, Compaq Computer Corporation, 2000