Real-Time Procedural Animation of Trees

Jeremy T. Barron
Brian P. Sorge
Timothy A. Davis

Department of Computer Science, Clemson University, Clemson, SC

Abstract

Creating models of living flora, such as trees and grass, has been a challenge in computer graphics for many years. Animating these objects to move realistically in reaction to natural phenomena, such as wind and rain, presents an even greater challenge. In this project, we explore the combination of particle systems with the Lindenmayer model for representing trees to create a realistic simulation of tree movement in response to wind. Our approach, however, is general enough to handle tree animation in response to a variety of other world forces, such as rain, snow, or seismic activity.

1 Introduction

The field of computer graphics has long been interested in creating models of plants such as trees and grass. Animation of these plants further enhances the realism they add to computer-generated scenes. Such animations are useful in a variety of applications including simulators, games, and film special effects, yet the animation of plant movement is rarely approached. Over the past decade we have seen some examples of plant movement in computer-generated movies, but this animation is done using proprietary methods, or is not done at interactive or real-time rates. Lindenmayer-systems (L-systems) are widely used for generating realistic models of trees because the models they generate are based on the actual growth patterns of plants. L-systems do not, however, provide a method for the animation of these models.

In our approach, we add particle systems to the underlying structure of the model in order to create animations with the trees. Using particle systems allows us to animate the trees procedurally, and thus eliminates the need for detailed key-frame animation of hundreds of branch segments. This method also makes it easy to modify the animation by changing a few input parameters, such as wind speed and direction. Modification of these parameters can be used to create a nearly infinite number of animations with the same model. Further, since our approach is based on the physical constraints of branch movement, our system can easily be expanded to simulate the trees' reactions to other phenomena such as rain, snow, or seismic activity.

The rest of this paper is organized as follows. In section 2 we give background on the general use of L-systems and particle systems, as well as discuss the principles from physics used in the animation of our trees. Section 3 highlighs previous work in the area of tree modeling and animation. In section 4 we describe in detail the method used to generate a generalized tree structure, while in section 5 we describe the techniques used to animate these trees in a unique way. Section 6 discusses the input parameters and describes how the user can modify these values interactively. Finally in section 7 we offer conclusions and describe future avenues for our research.

2 Background

To generate a tree structure for animation, our technique incorporates methods from several areas of computer science and physics. These techniques include L-systems for the tree, particle systems for the animation of the tree, and the physics of angular motion and springs to control the particle systems. In this section we provide background for these techniques.

2.1 L-systems

L-systems are named for Aristid Lindenmayer who first developed the idea to provide a mathematical model for plant development in 1968 [LIND68]. L-systems provide a formal method for describing plant growth. This method has become a standard for creating computer-generated plants largely because it is based on empirical data gathered on the development of actual plants. For our system we use context-free parameterized L-systems as described in [PRUS88] and expanded in [PRUS90].

A LOGO-style turtle [ABEL82] interprets the strings produced by these grammars in the following manner. As the turtle reads each character, it checks to see if the character affects its state.

2.2 Particle Systems

Reeves in 1983 [REEV83] first introduced particle systems as a method for generating procedural animations. These systems have been used to model smoke, fire, grass, water, rain, and any number of other phenomena. The particles may be rendered in whatever manner is needed for the application. The great advantage of using particles is that many particles may be updated simultaneously in a procedural way to generate animations that would be incredibly difficult, if not impossible, to create manually. We represent our tree as a particle system to animate it procedurally.

2.3 Mechanics

The forces in our simulation are based on the mechanical concepts of rotation of rigid bodies and properties governing the behavior of springs. In this section we will address these two areas. This material is derived from [SERW92].

2.3.1 Angular Motion

The branches of our tree are represented as cylindrical rods that are fixed at one end and allowed to rotate about that end. Here we describe the physical equations for this motion. First we must address the force acting on the rod. If we assume that the force is acting on the end of the rod, then we can use the force to calculate the torque on the axis point:

$$\tau = f \otimes R \qquad (2.1)$$

where f is the force acting on the end of the rod, and R is a vector representing the length and orientation of the rod. The angular acceleration α is related to the torque as follows:

$$\tau = I\alpha \qquad (2.2)$$

where I is the moment of inertia, which for a long thin rod of mass M and length L is expressed as

$$I = \frac{1}{3}ML^2 \qquad (2.3)$$

Angular acceleration is related to the angular velocity ω by the parameterized relationship:

$$\omega = \omega_o + \alpha t \qquad (2.4)$$

Integrating equation (2.4) we get an angle θ:

$$\theta = \theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2 \qquad (2.5)$$

Given angle θ we now know how much to rotate the rod in a given time step.

2.3.2 Hook's Law

The use of springs is helpful in particle systems to model interactions between particles. When one particle is moved, it pushes or pulls on the other particles in the system, causing the entire system to move.

In order to model the resistance of branches to movement, we add a spring between the current position and the rest position of each particle in our system. These springs act according to Hook's law:

$$f = -k_s x \qquad (2.6)$$

where f is the force acting on the spring, k_s is the spring constant of the spring and x is the total displacement of the spring. In addition, a dampening force is added to this equation to prevent the spring from oscillating continually once all forces are removed from the system. This dampening force is dependent on a dampening constant k_d for the spring and the velocity of the particle at the end of the spring. Adding the dampening force to equation (2.6) yields:

$$f = -k_s x + k_d v \qquad (2.7)$$

2.4 Splines

We would like our branches to appear as smooth curves rather than straight line segments connecting particles, but representing branch segments small

enough to render as a curve is computationally expensive. We employ Catmull-Rom splines [FOLE90] to draw tree branches as curves.

3 Related Work

A great deal of work has been done in the field of graphical tree generation. Bloomenthal [BLOO85] used a simple procedural method to generate realistic maple trees. Branching patterns were determined by stochastically choosing branching angles and lengths. Prusinkiewicz and deReffye [PRUS88] [PRUS90] [PRUS93] [PRUS94] [DERE88] have used L-systems to generate a variety of different plants, beginning with simple ferns and progressing to full-sized trees. These systems have also been used to animate plant growth, but not their movement. Power [POWE99] employed a method of springs similar to our approach in order to determine the natural orientation of branches at rest by minimizing the energy in the system.

For animation, Perbet [PERB01] used particle systems and pre-rendered images of blades of grass in different positions to animate grass movement in prairies. Stam [STAM97] and Weber [WEBE95] used modal analysis of flexible beams to generate animations of trees swaying in the wind. Many of these earlier efforts used some of the same techniques used here, but relied upon pre-computed animations or pre-rendered images to draw the trees. The primary advantage of our approach is that the underlying particle system provides for fast computation and hence real-time frame rates.

Noser, Thalmann, and Turner [NOSE92] [NOSE97] used particle systems to model forces acting on trees created using L-systems. Their method modifies the characteristics of the turtle states to change the tree based on forces in the system. While the animations produced by this method are similar to those created by our approach, the representation of the trees differs in the continual usage of the grammar. Additionally their method lacks continuity between branch segments, as they model branches simply as linear connections of the turtle positions and make no effort to smoothly move between the radii of connected branch segments

4 Tree Generation

Our tree model is based on L-systems and particle systems. Using the L-systems described in section 2.1, words are generated which are then parsed with the turtle method to generate a particle system that represents the tree. The particles are used as control points for a set of splines that model the tree branches.

4.1 Particle System Generation

Once a string is fully generated from an L-system grammer, it must be interpreted to draw a tree. The first step of the interpretation is to build the particle system representing the tree. This section describes that process.

Before the turtle begins parsing the string, a root particle is created and becomes the current particle. Anytime the turtle's position is moved, a new particle is added. The radius and position of the particle are determined by the current state of the turtle.

At this point the current particle is checked for children. If a first child is found, we walk to the end of the list of children, place the new particle at that position, and update navigational pointers as needed. Otherwise the new particle is placed as the first child of the current particle, and we again update navigational pointers. The new particle's parent pointer is assigned to the current particle, and the new particle's branch value is made equal to the new particle's position minus the current particle's position. If the current particle's stack depth is equal to the depth of the stack when the new particle is added, the current particle's dominant pointer is assigned to the new particle.

Once all particles have been added, the radii of every particle in the tree are scaled by the same value. This value is determined by the smallest radius of any particle in the tree, and a constant defining the minimum radius of any branch in the tree. This scaling allows the grammar to determine the rate at which radii decrease in the tree, but also makes branches thicker as the tree grows.

4.2 Tree Geometry

We can get a basic outline of our tree by simply connecting the points in the particle system and drawing a line from every particle in the system to each of its children (see Figure 4.1). We would prefer, however, to show the tree with smooth curves along the branches and with thickness added to the branches. The following two sections describe the process used to generate splines from the particles in the system and how those curves are converted into trunk and branch segments for the tree.

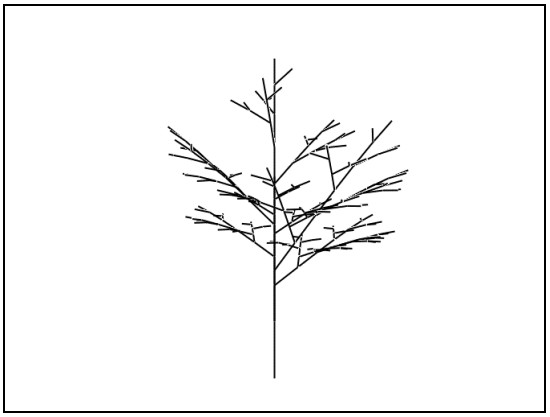

Figure 4.1 *Linear Connection of Particle System*

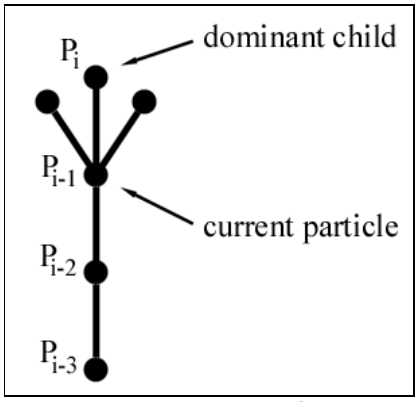

Figure 4.3 *Curve Control Points*

4.2.1 Splines

Since we would prefer to represent branches in our tree as smooth curves rather than straight line segments between particles in our system, we use Catmull-Rom splines as mentioned in section 2.4. A tree skeleton drawn without using splines and using this method of spline generation is shown in Figure 4.2. A wind force has been applied to help show the curve of the tree.

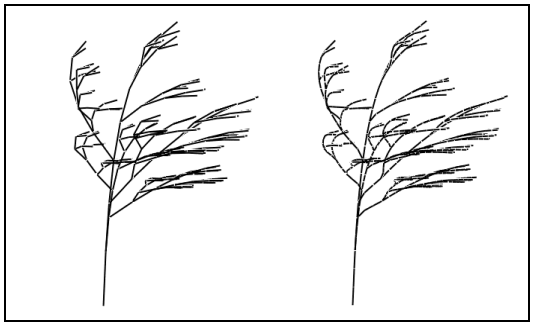

Figure 4.2 *Tree Drawn Without and With Using Catmull-Rom Splines*

The control points for these splines are the current positions of the particles in the particle system. In general, the points used to draw the curve for any branch segment in the tree are as follows: the position of the particle at the end of the branch along with the positions of its child, its parent, and its parent's parent. These four positions, shown in Figure 4.3, give values for P_i, P_{i-1}, P_{i-2}, and P_{i-3}.

4.2.2 Skins

In order to have three-dimensional branch segments, we must generate geometry around each branch segment. We create the geometry by generating vertices around each curve point for a particular branch, and connecting these points to form polygons. Since the curve points change per time-step, these vertices must be created each time-step as well.

Rather than calculating the actual vertex positions, we compute vectors, that when added to the curve points, give the vertex positions. Using these vectors also provides us with a surface normal for the vertex. Note that this normal is not the exact average of the normals for all of the surrounding polygons along the length of the branch segment as shown in Figure 4.4. The value used, however, is close enough that we are still able to achieve smooth shading of the branches, since the error is relatively small. Because the vectors radiate from the center of the branch segment to the vertex positions, they are the proper averaged surface normals about the circumference of the branch segment.

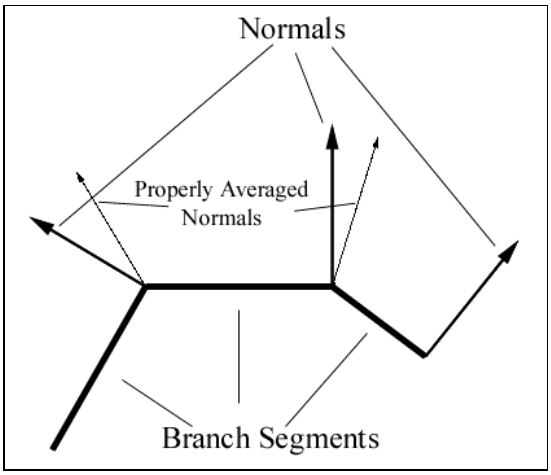

Figure 4.4 *Vertex Normals*

The first step is to calculate the vector from the current curve point to the first vertex. Figure 4.5 illustrates the polygon-twisting problem created by incorrectly positioning the first vertex of a given polygon. In order to keep the polygons from twisting, we compute the first vector of the current curve point $\vec{V}_{current}$ based on the first vector of the previous curve point $\vec{V}_{previous}$. This method assures that the first vector from the current curve point of the particle is parallel to the first vector from the previous curve point, and hence prevents any twisting.

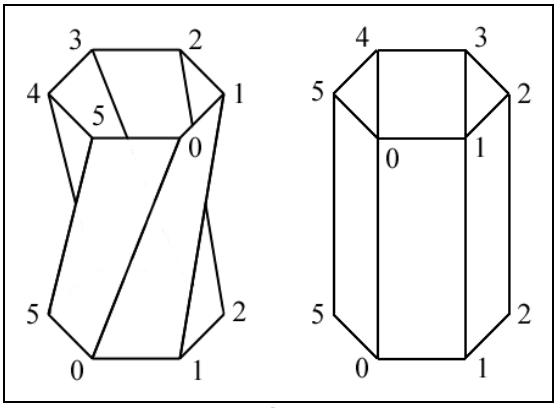

Figure 4.5 *Polygon Twisting*

Once the first vector is computed, the remaining vectors are obtained by rotating the preceding vector about $\vec{C}_{current}$ (the particle's current position) by $2\pi/N$ where N is the number of polygons representing the branch. The vertex generation is shown in Figure 4.6. Figure 4.7 shows the tree from Figure 4.1 in wireframe to illustrate the locations of the vertices.

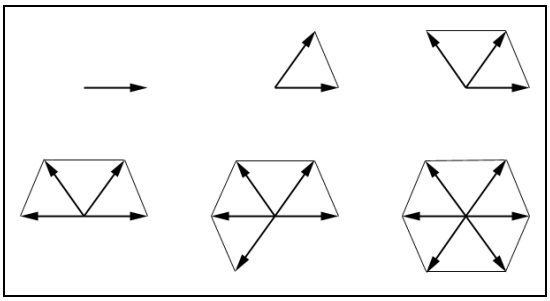

Figure 4.6 *Vertex Generation*

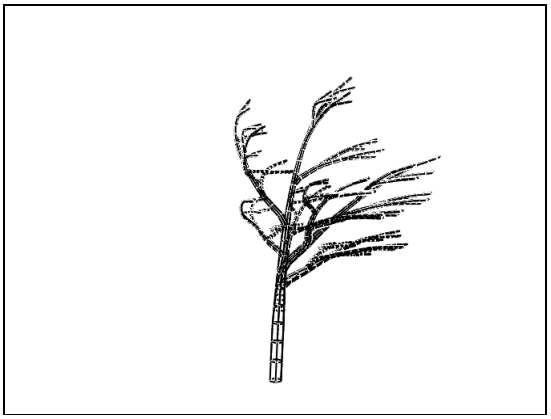

Figure 4.7 *Tree Drawn in Wireframe*

Once the tree has been generated, we desire a method for updating it based on the forces acting upon the particle system representing its structure. In the next section, we will discuss our method for animation in detail.

5 Tree Animation and Rendering

In order to animate the tree in real-time, the particle's positions are updated and drawn at each time step. The spline skeleton described in section 4.2.1 is then used to create three-dimensional geometry, which can be lit, texture-mapped, and shaded. We also add a background and shadows to our scene for realism.

5.1 Particle Update

The process of animating the tree is based on updating the tree's underlying particle system structure. At each update step, the newly computed particle positions are used to create a new set of splines, and thus a new set of vertices for skinning the tree. This process is explained more fully below.

5.1.1 Force Calculation

The first step in updating the particle system is to determine what forces are acting on a particular particle. There are three forces currently implemented in our program: gravity, wind, and the stiffness of the tree (modeled as springs).

The gravity force $\vec{F}_G$ is simply a constant force applied in the negative *y*-direction. This force causes the branches of the tree to bend downward slightly, creating a more realistic appearance.

The wind force is determined by the user and may be in any direction and of any strength desired.

Also, to enhance the realism of the animation, a turbulence factor is added to the wind force $\vec{F}_w$, which is also user-specified. The turbulence *T* varies across time-steps to simulate temporal variance in the wind, with a small scalar factor *t* applied per particle to simulate spatial variance in the wind. One of the nice features of our approach is that the wind model can easily be replaced with a more physically accurate model without changing the tree animation method.

The stiffness of the tree at any particle is simulated by the spring force $\vec{F}_s$. This force is determined by the current particle's spring constant, current position and rest position. The displacement of the spring is simply the difference in the current position and the rest position. This value, along with the particle's current velocity, is substituted into equation (2.7) to determine the force of the spring acting on the current particle. The total force acting on the particle is therefore

$$\vec{F}_T = \vec{F}_G + \vec{F}_W + \vec{F}_S \qquad (5.1)$$

Additional forces can also be added without difficulty to simulate snow, rain, or other phenomena.

5.1.2 Branch Rotation

Once the force is calculated, we can determine the torque on the branch whose endpoint is the current particle from equation (2.1). Using equations (2.2) – (2.5) we can determine the angle of rotation θ for the branch in response to the forces exerted on it. The branch is rotated θ degrees about the torque vector, and added to the parent particle's current position resulting in the new position for the current particle. Once the new position of the particle has been calculated, the difference between this new position and the rest position represent the particle's spring vector $\vec{S}$. Figure 5.1 shows the current and new positions of the particle.

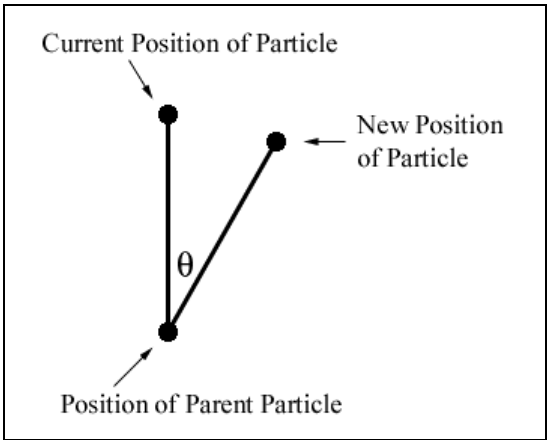

Figure 5.1 *Branch Rotation*

5.1.3 Propagation

Once the new position of the particle is calculated, the rest and current positions of all children of that particle must be updated accordingly. In order to propagate the motion we need

- $\vec{I}_{current}$ - the initial position of the current particle
- $\vec{C}_{current}$ - the current position of the current particle
- $\vec{I}_{child}$ - the initial position of the child particle
- $\vec{R}_{child}$ - the rest position of the child particle
- $\vec{S}_{child}$ - the spring vector of the child particle

Fortunately all of this information is readily available in the data structure for the tree, so the new rest and current positions of each child particle are simply:

$$\vec{R}_{child} = \left(\vec{C}_{current} - \vec{I}_{current}\right) + \vec{I}_{child} \qquad (5.2)$$

$$\vec{C}_{child} = \vec{R}_{child} + \vec{S}_{child} \qquad (5.3)$$

We only have to modify the children of the current particle based on the current particle's movement since the movement of the child positions will be propagated throughout the depth of the tree. Figure 5.2 shows how a child's position is modified based on the movement of the current particle. Figure 5.3 shows four frames of an animation of the tree responding to these forces.

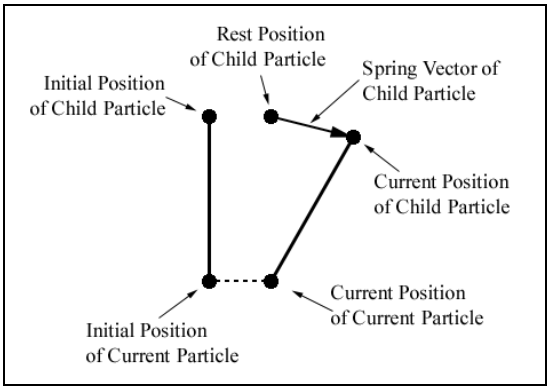

Figure 5.2 *Propagation of Branch Rotation*

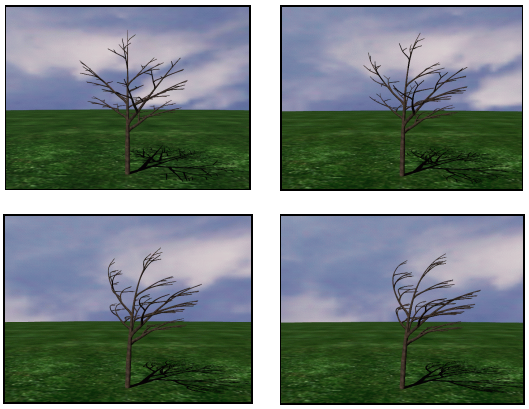

Figure 5.3 *Frames of Tree Animation*

5.2 Drawing the Tree

To draw the tree, we traverse the particles residing at each branching point and draw the branch segment ending at each particle's position. Initially, we make the first child of the root particle the current particle since the root particle has no branch segment ending at its position. Once the segment ending at a particular particle is drawn, each of its children is drawn, followed by each of its siblings.

5.3 Realism

In addition to three-dimensional branches and smooth shading, we would like to augment our model to enhance the realism of our tree. We therefore apply texture mapping to the tree (see Figure 5.4), as well as a background and shadows. A texture of scanned tree bark is applied to each branch segment of the tree. In order to give the tree an environment in which to exist, we add a simple background. A large plane with a grass texture map suffices for the ground, and a large sphere with a sky texture map provides a suitable backdrop. We can more fully integrate the tree into the scene by adding a shadow of the tree on the ground as in Figure 5.5.

Figure 5.4 *Close Up View of the Texture Mapped Tree*

Figure 5.5 *Tree with Shadow*

6 User Interface

All properties of the tree are controlled by the user interface shown in Figure 6.1. This interface has controls for the tree model itself, as well as for the viewer and the scene display method. These controls are explained in the following two sections.

6.1 Tree Model Controls

A tree is generated based on a grammar file chosen by the user with the help of a file browser. Once the grammar has been parsed, a horizontal slider determines the depth of the tree. The user can observe the tree's growth as the tree depth is incremented. The growth of new tree depths slows down exponentially due to the increase in the string lengths generated by the grammar.

The force of the wind may be controlled using three sliders: one for the wind in each of the x-, y-, and z-dimensions. A reset button is provided to stop the wind force completely at any time. A fourth slider allows the user to control the amount of turbulence described in section 5.1.1.

Sliders are also present to determine the curve resolution and step size. The curve resolution slider is used to determine how many spline segments to draw per branch segment. The time step slider determines the length of the time step between updates of the tree animation.

6.2 Viewer Controls

The user may move the eye point along all three axes in a pre-specified range. Additionally the eye point may be rotated about the view direction, the up vector, and about the cross product of the up vector and view direction. Sliders on the interface control these translations and rotations. A reset button is provided to place the viewpoint and eye point back at the position set at the start of the program.

6.3 Scene Controls

Several options are provided in a separate window for the display of the scene. By default the program shows the tree lit, smooth-shaded, and texture-mapped with a texture-mapped background and a shadow. The texture mapping, background, and shadow all may be toggled on or off at the user's discretion. Additionally, the tree may be drawn smooth-shaded, flat-shaded, in wireframe, or as a skeleton. Smooth shading, flat shading, and wireframe views of the tree are all handled directly by OpenGL. The skeleton of the tree is displayed using OpenGL to draw line segments between the tree's curve points rather than drawing polygons based on the tree's vertices.

7 Conclusions and Future Work

In this paper, we have provided the background and details of our method for tree animation. In this final section, we provide some of the improvements made to the system to increase performance. Additionally, we describe future directions for our research, as well as give some concluding remarks on the project to date.

7.1 Speedups and Performance

A great deal of computation must be done to move the tree for each update. In order to improve the performance of the program, a number of methods were used to reduce the computation required in order to maintain real-time, or at least interactive, rates. In this section we describe the most beneficial methods as well as provide some information about the performance of the program.

A major hindrance to the performance of the system involves the more advanced features used to enhance the realism of the tree. To minimize the effect of these enhancements, the user interface allows for deactivating the background, texture maps, lighting, and shadows.

Another area for performance improvement is in the calculation of vertex positions. When generating the vectors from the curve points of a branch segment, we calculate the position of the first vector, and then rotate that vector about the curve segment for as many vectors as are needed as described in section 4.2.2. As long as an even number of vectors is desired, then we need only calculate the first half of the vectors in this manner. Then, rather than continuing the computationally expensive task of rotating the vector about an arbitrary axis (the curve segment) subsequent vectors are calculated as

$$\vec{v}_i = -1.0 * \vec{v}_{i-\frac{n}{2}} \qquad (7.1)$$

where $\vec{v}_i$ is the current vector, and n is the number of polygons to be drawn about the circumference of the branch. This method resulted in an increased frame rate of ten percent.

The final major performance improvement came from making use of OpenGL's GL_QUAD_STRIPS capability. Rather than drawing each polygon individually, we specify a series of vertices defining a strip of polygons. This method reduces the number of vertices that must be explicitly defined by half.

Drawing a tree with 255 branches and with a curve resolution of 3, and with all background, texture, and shadow elements, the program runs at 6 frames per second on an SGI O2, and at 30 frames per second on an SGI Onyx2. This frame rate makes it quite easy to control all parameters of the tree interactively.

7.2 Future Work

Many open issues remain for future work. Among the avenues being considered for this research are

- leaves and their reaction to world forces
- additional forces such as rain and snow

- a more realistic wind model
- different types of plants, and possibly hair and fur
- parellelization of our method
- multiple trees

7.3 Concluding Remarks

In this paper we provide a method that allows real-time animation of realistic trees. Our method is general enough to animate plant reaction to a number of natural phenomena. By using a procedural method, the tree generation and animation occurs with minimal input from the user, making the method easy to use. Our method is also fast enough to result in real-time, or at least interactive, animations. As processor speeds increase, this performance will only improve.

REFERENCES

[ABEL82] H. Abelson and A. A. diSessa, "Turtle geometry," M.I.T.

[BLOO85] J. Bloomenthal, "Modeling the Mighty Maple," *Proceedings of SIGGRAPH 85* (San Francisco, California, July 22-26, 1985), in *Computer Graphics Proceedings, Annual Conference Series*, 1985, ACM SIGGRAPH, pp. 305-311.

[DERE88] P. deReffye, C. Edelin, J. Francon, M. Jaeger, and C. Puech, "Plant Models Faithful to Botanical Structure and Development," *Proceedings of SIGGRAPH 88* (Atlanta, Georgia, August 1-5, 1988), in *Computer Graphics Proceedings, Annual Conference Series*, 1988, ACM SIGGRAPH, pp. 151-158.

[FOLE90] J. D. Foley, A. vanDam, S. K. Feiner, and J. F. Hughes, *Computer Graphics Principles and Practice, Second Edition*, Addison-Wesley, Reading, MA, 1990.

[LIND68] A. Lindenmayer, "Mathematical Models for Cellular Interaction in Development, Parts I and II," *Journal of Theoretical Biology*, 1968, pp. 280-315.

[NOSE92] H. Noser, D. Thalmann, R. Turner, "Animation based on the Interaction of L-systems with Vector Force Fields," Proceedings Computer Graphics International 1992, 1992, pp. 747-761, Springer-Verlag.

[NOSE97] H. Noser, "A Behavioral Animation System Based on L-systems and Synthetic Sensors for Actors," PhD Dissertation, Swiss Federal Institute of Technology, 1997.

[PERB01] F. Perbet, and M. P. Cani, "Animating Prairies in Real-Time," *ACM Interactive 3D Graphics*, 2001.

[POWE99] J. L. Power, A. J. B. Brush, P. Prusinkiewicz, and D. H. Salesin, "Interactive Arrangement of Botanical L-system Models," *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, 1999, pp. 175-234

[PRUS88] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan, "Developmental Models of Herbaceous Plants for Computer Imagery Purposes," *Proceedings of SIGGRAPH 88* (Atlanta, Georgia, August 1-5, 1988), in *Computer Graphics Proceedings, Annual Conference Series*, 1988, ACM SIGGRAPH, pp. 141-150.

[PRUS90] P. Prusinkiewicz, and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

[PRUS93] P. Prusinkiewicz, M. S. Hammel, and E. Mjolsness, "Animation of Plant Development," *Proceedings of SIGGRAPH 93* (Anaheim, California, August 2-6, 1993), in *Computer Graphics Proceedings, Annual Conference Series*, 1993, ACM SIGGRAPH, pp. 351-360.

[PRUS94] P. Prusinkiewicz, M. James, and R. Mech, "Synthetic Topiary," *Proceedings of SIGGRAPH 94* (Orlando, Florida, July 24-29, 1994), in *Computer Graphics Proceedings, Annual Conference Series*, 1994, ACM SIGGRAPH, pp. 351-358.

[REEV83] W. T. Reeves, "Particle Systems – A Technique for Modeling a Class of Fuzzy Objects," In *Computer Graphics*, Vol. 17, No. 3, 1983, pp. 359-376

[SERW92] R. A. Serway. *Physics for Scientists and Engineers with Modern Physics*. Saunders College Publishing, Philadelphia, 1992.

[STAM97] J. Stam, "Stochastic Dynamics: Simulating the Effects of Turbulence on Flexible Structures," *Computer Graphics Forum, Proceedings of Eurographics '97*, 1997, pp. 159-164.

[WEBE95] J. Weber, and J. Penn, "Creation and Rendering of Realistic Trees," *Proceedings of SIGGRAPH 95* (Los Angeles, California, August 6-11, 1995), in *Computer Graphics Proceedings, Annual Conference Series*, 1995, ACM SIGGRAPH, pp. 119-128.

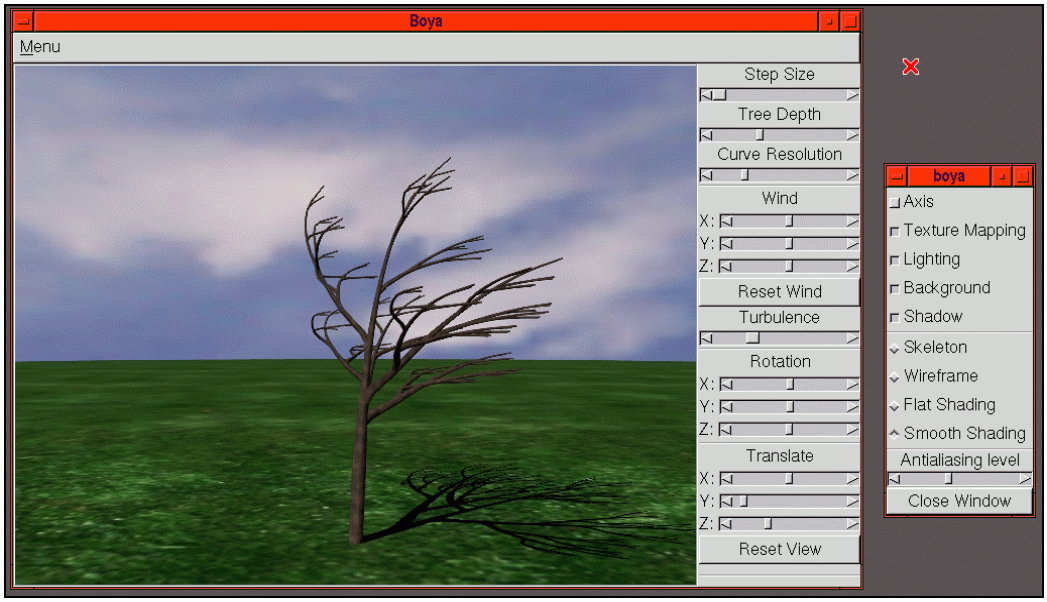

Figure 6.1 *User Interface*