An Implementation on Real-time Animation of Trees Swaying in Wind Fields

Cansin Yildiz,

Bilkent University, Computer Engineering Department
06800 Ankara, Turkey
cansin@cs.bilkent.edu.tr

Abstract. Trees are one of the most important objects of everyday life. Many studies have been made on simulating tree motion under influence of wind. Most of the time, those studies use physical based approaches to generate the branch motions. In this paper, I propose a hybrid method to create natural motion of branches and leaves using both physical equations and stochastic effects, based on [1]. This method enables real-time animation of branches, leaves and a projective shadow of the tree.[1]

Keywords: tree, motion, wind, spring-force model, real-time animation.

1 Introduction

An accurate representation for vegetation is one of the most important topics in computer graphics field. Starting with late 80's, a lot of research has been done concerning this issue. However, most of those studies [2] [3] [4] are solely focused on modeling the shapes of trees and vegetation, while ignoring motion. But, fortunately, there are also works which incorporate model with motion [1] [5] [6]. Most of these methods generate branch motion using pure physical based approaches. Although, physical approaches can create very accurate motion of branches under influence of wind and other external forces, their calculation complexity makes real-time execution hard. There is also a completely different approach for plant motion problem; to use motion capture data as guidance for generating new plant motions [10].

To the author's knowledge, there are few methods which can realize real-time execution [1] [7] [8] [9]. Similar to what is introduced by *Ota et al.* [1], this paper proposes an efficient way of creating natural branch, and leaf motion. The method is a hybrid one, which combines spring-force model for general motion and stochastic sine oscillation for randomness and leaf motion. This method can easily handle different kinds of trees, like weeping trees (see **Fig. 1**).

[1] This work is a two-course project. The branch simulation and rendering part is implemented at CS565 course of Prof. Ozguc at Bilkent University. Similarly, the **leaf simulation, rendering** and **shadow casting** stages are implemented at **CS568** course.

Fig. 1. Some examples of trees; a weeping tree, a palm trunk and a regular tree.

In section 2, a brief description of overall process is given. Section 3 continues with giving the details of branch simulation/rendering step of the method. After giving some examples of branch motions as well, paper continues with explaining leaf simulation/rendering at section 4. Then, at section 5, the projective shadow approach that is used to generate tree shadow is explained. I conclude the paper with some discussion on future work and so forth at section 6 and 7.

2 Overview of the Method

As all animation methods, the approach has two main concerns; simulation and rendering. For simulating trees' branch motion, a physical based approach is used. The general motion of each branch is simulated as a mass-spring system. Using the displacement caused by the external wind force, a displacement angle is calculated for each branch segment, and it is accumulated through their children as well. The resulting simulation is further enriched by adding oscillation behavior of sine-based stochastic force. For that simulation to be as realistic as possible, some other measures are also taken, which is discussed deeply in following section. Similar to branches' stochastic behavior, leaf motion is simulated using a sine-based force. Since, leaves' actual physically correct behavior is no concern for this real-time application, this stochastic simulation was enough and pretty realistic.

The other issue that has to be handled was rendering. Basically, the concern is to have a tree rendering as realistic as possible. To do that some problems are needed to be overcome like skinning the tree, solving breaking of branch segments, etc. Another issue, which is by no ways less important, is to have a supportive environment for tree animation. The details of both rendering concerns are investigated at section 3. Then, at section 4, the details of leaf rendering is given. Basically, a masking approach is used to render alpha-blended leaves. Shadow is a major concern for realistic rendering. Since, the environment had a flat ground surface; a projective shadowing technique was enough to render the shadow.

I strongly suggest readers to watch the implementation videos to better see the results, since this is a tree *simulation/rendering*.

3 Branches

3.1 Simulation

To represent the wind effect on branches, I adapted a simulation method based on cantilever spring model as proposed at [1]. The overall simulation that is obtained from this cantilever spring model is supported by an oscillation simulation using sine-based force functions. The rest of this section is first explains the branch model that is used to represent the tree, and how it is actually simulated.

3.1.1 Branch Model of a Tree

The model for representing a tree is consists of branches. Between each two consecutive joints, there is at least one branch segment (see **Fig. 2**), but there can be multiple segments as well to have a soft motion of long branches; which is most common in weeping trees. Each branch segment is given some properties, namely, *length*, *base width* and *initial angles x/y*. *Top width* of a branch segment is calculated according to its children's base widths.

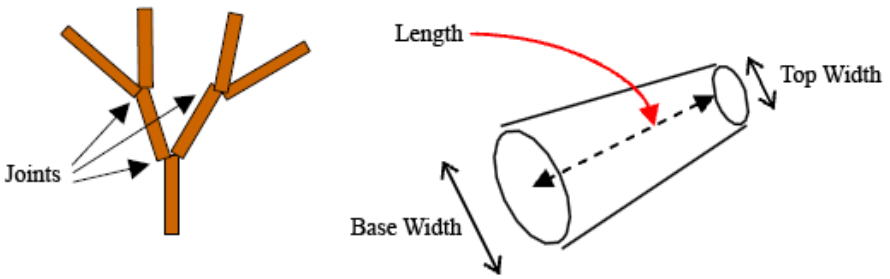

Fig. 2. Branch segments and their properties.

3.1.2 Simulation Based on Spring Model

Each branch segment has its own local coordinate system, which makes simulation easier. The origin of this local coordinate system is set to bottom of the branch segment, and axes are perpendicular to branch direction (see **Fig. 3**).

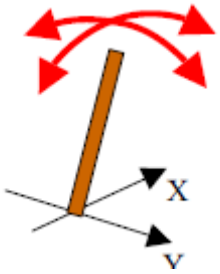

Fig. 3. Local coordinate system of a branch segment.

The simulation method is based on cantilever flat-spring model, as proposed by *Ota et al.* 1. As they described, a branch segment is approximated as a squared non-elastic wood, and the displacement amount is calculated from the wind load given to the branch segment. The pressure that is applied to the branch along x/y directions is described as;

$$P_x(t) = F_x(t) * (1 + a * \sin(t + b)) \quad (1)$$
$$P_y(t) = F_y(t) * (1 + a * \sin(t + b)) \quad (2)$$

Here, the first term *F(t)* is the directional force that can be applied to a branch segment. For current purpose, it is used as the *wind force*. Although the coefficient that is applied to *F(t)* may seem complicated, it is pretty straight-forward. *1* is to represent the wind force's itself. *a*sin(t+b)* coefficient is the so called oscillation behavior of the branch segment. By having a coefficient *a* in front of sine function, the current strength of oscillation is controlled. *a* is a small number, which is derived from the current angle of the branch segment. Basically, if a branch segment is bended much, it will oscillate more as well. The randomness of this oscillation is realized by *b* constant. It is basically a random number, which makes each branch segment to be in different *sine* phases at same time step.

Using those loads, displacement amounts of the branches are calculated as;

$$\delta_x(t) = P_x(t)/k \quad (3)$$
$$\delta_y(t) = P_y(t)/k \quad (4)$$

Here, *k* is the spring constant of branch segment. It is determined by;

$$k = \frac{Ebt^3}{4l^3} \quad (5)$$

E, the elastic modulus, is specific to each tree. Whereas, the width *b*, thickness *t* (base width - top width), and length *l* is specific to each branch segment.

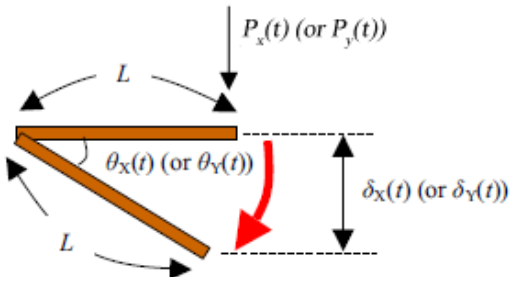

Fig. 4. Simulation of branch motion.

After calculating the displacement amounts using spring constant and load, the actual motion angles *Θ(t)* are needed to be calculated (see **Fig. 4**). Those angles are calculated as;

$$\Theta_x(t) = \sin^{-1}(\delta_x(t)/l) \quad (6)$$

$$\Theta_y(t) = \sin^{-1}(\delta_y(t)/l) \quad (7)$$

It must be noted that, the final motion angles are further restricted to not exceed a certain degree *x* (20° in current implementation) to force the natural behavior of tree branches.

Although this method uses a physical approach, it can maintain real-time performance because of the flat-spring model's simplicity.

3.1.3 Integration of Branch Motions

The motions of individual branch segments are obtained as described at 3.1.2. After that they must be integrated to form the final motion of the actual tree. As shown at **Fig. 5**, the branch segments' motion is summed up from parent to children branches.

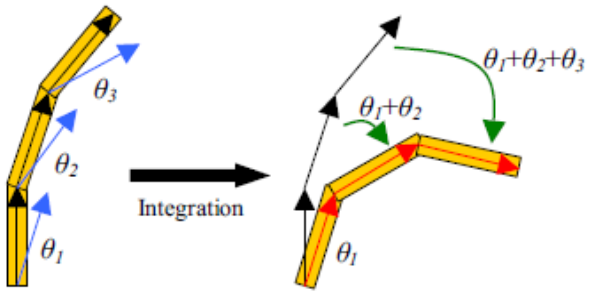

Fig. 5. The accumulation of motion through parent to children segments.

3.2 Rendering

A successful simulation should always be supported with a realistic rendering. There are two main concerns about rendering; rendering a realistic tree and a beautiful scene that the tree can live in. The rest of this section gives the details of those rendering issues.

3.2.1 Tree

Unlike most computer animation problems, rendering an acceptable tree is a straight-forward problem. To achieve that, the tree branches are rendered as cones and covered with a decent bark texture. The only problem about this approach was that there could be some breaking conditions; i.e. when the angle between two consecutive branch segments are larger than a value, it could be seem like the branch is breaking. This notion is overcome by simply rendering spheres of proper radius between each branch segments (see **Fig. 6**).

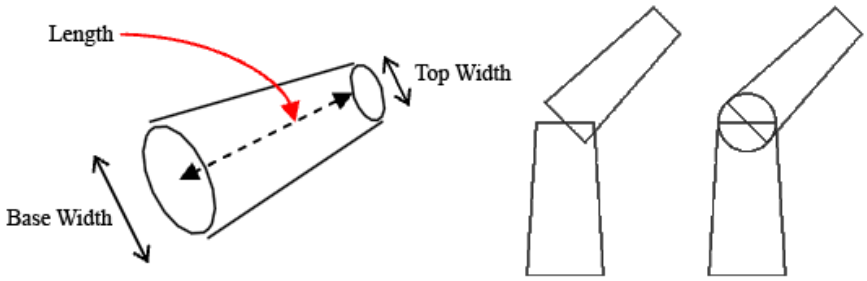

Fig. 6. Breaking problem is handled using spheres of appropriate radius.

3.2.2 Scene

Plate that used is always as important as the food served. Therefore, a realistic scene for tree animation is a top priority. Using only simple OpenGL tricks, a decent scene is created. This scene has some fine details like a sky-dome with a nice texture, a skyline between sky and ground, and even the little grasses that are near root of tree trunk. Those details can be seen at **Fig. 7**.

Fig. 7. Details of rendering.

3.3 Examples

In this section, some examples that are simulated using the actual implementation will be demonstrated. But, since it is a tree *animation*, I urge you to watch the supplementary videos.

The first figure (**Fig. 8**) is a regular tree which sways under effect of wind. You can observe how overall movement is consistent with general wind direction yet has randomness as well.

Fig. 8. A regular tree under effect of wind. White arrows represent wind direction and magnititude.

The second result is more interesting. The system is tested for a different kind of tree; a weeping tree (see **Fig. 9**).

Fig. 9. A weeping tree under effect of wind.

In weeping tree test, it is realized that an important implementation detail was skipped. When the branch segments lays upside-down, the wind force should decrease their angle, rather than increasing it (see **Fig. 10**). After adding that little specification, system is fully suitable for all kinds of trees.

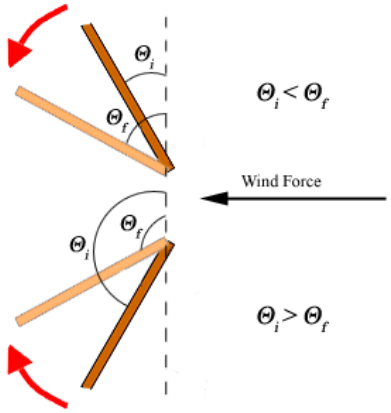

Fig. 10. Upside-down branch vs. normal branch.

Last tree that is simulated is a trunk of palm tree (see **Fig. 11**). This run nicely represent the notion that a trunk can consist of several branch segments, to simulate the flexible behavior.

Fig. 11. A palm tree under effect of wind.

4 Leaves

4.1 Generation

Since adding thousands of leaves to a tree manually is not a feasible idea, I have implemented a generation process for the leaves. It is a simple procedure that follows the idea: *If branch is thin enough (i.e. it's a twig rather than trunk), generate some number of leaves with random local positions and orientations.* At **Fig**. **12**, you can see a randomly dressed tree with leaves.

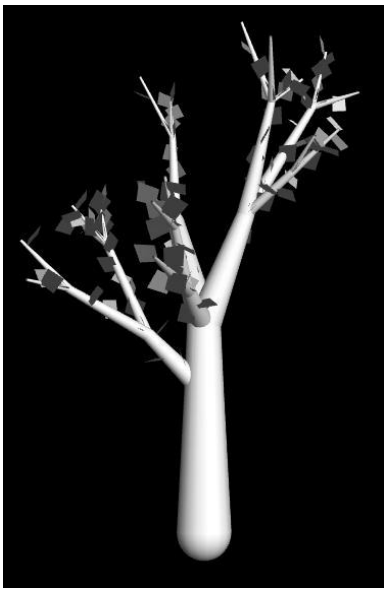

Fig. 12. A tree with randomly generated leaves.

4.2 Wind Simulation

To simulate leaves, I am using a stochastic approach, which is proposed by *Ota et al.* at [1]. To achieve the fundamental simulation, horizontal, vertical and rotational motions are calculated separately (see **Fig. 13**).

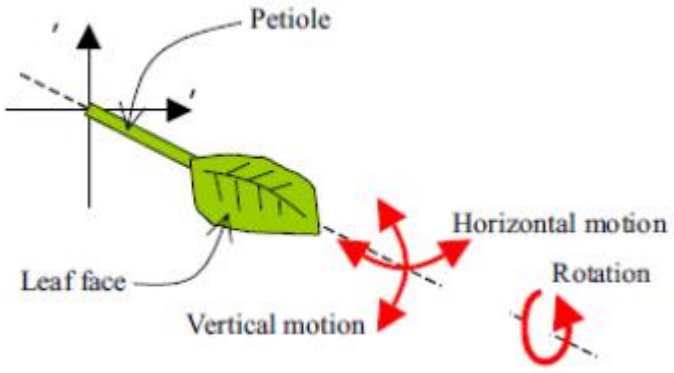

Fig. 13. Leaf simulation procedure.

The calculations are pretty straight-forward. For each motion component, there is a pre-defined maximum angle, and a per-leaf randomly generated parameter. The equations are;

$$\Theta_H(t) = W_H * N_H(t)$$
$$\Theta_V(t) = W_V * N_V(t)$$
$$\Theta_R(t) = W_R * N_R(t) + a * \Theta_X(t)$$

where,

$\boldsymbol{\Theta_{H/V/R}(t)}$: Motion angles at time *t*,
$\boldsymbol{W_{H/V/R}}$: Maximum motion angles,
$\boldsymbol{N_{H/V/R}(t)}$**:** Sine function w.r.t. time *t*.

One thing to note in these equations is that rotation angle calculation has an extra component, related to current horizontal motion. This additional parameter realizes a more natural behavior of leaf bending when wind blows leaf far away. You can better visualize the effect at **Fig. 14**.

Natural **Straight**

Fig. 14. Additional $\Theta_H(t)$ related component at $\Theta_R(t)$ achieves more natural look.

4.3 Shedding Simulation

Although having leaves on top of the branches and letting them move according to the wind added quite realism to the application, it was not enough for a successful tree simulation. Trees shed leaves almost always; therefore leaf shedding is also simulated (see **Fig. 15**).

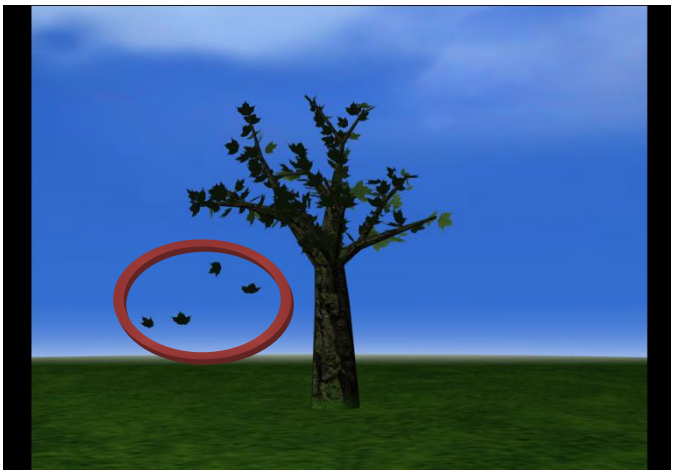

Fig. 15. A tree is shedding its leaves.

The procedure for shedding is a stochastic approach, just like the vast amount of other components of the application. Basically, the idea is to cut off a leaf from its parent branch, according to a per-leaf basis random variable. If the random exceeds a small threshold, the leaf starts falling down. This threshold is proportional to current wind speed, thus the number of leaves falling down gets higher as wind speeds up.

4.4 Rendering

Arbitrary shaped leaves are rendered for a better realism. It is achieved by using an alpha-blending procedure with masking. First, the mask texture is rendered at the scene, with *GL_BLEND* option *glBlendFunc(GL_DST_COLOR,GL_ZERO)* (see **Fig. 16**).

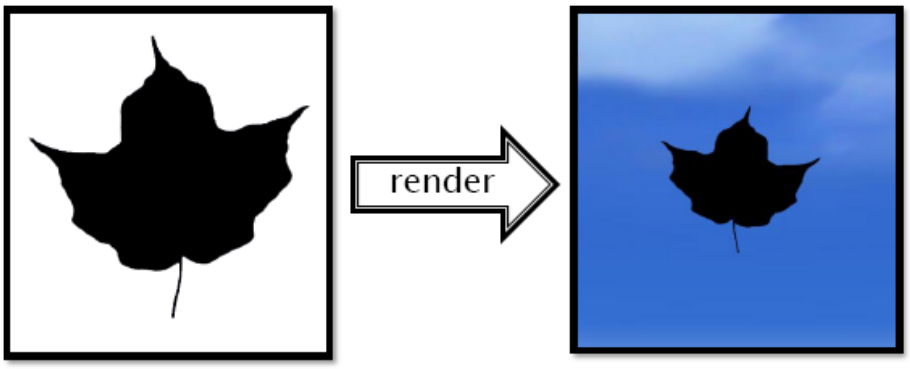

Fig. 16. Rendering mask texture to scene.

Then, actual leaf texture is rendered on top of mask using blend option *glBlendFunc(GL_ONE, GL_ONE)* (see **Fig. 17**).

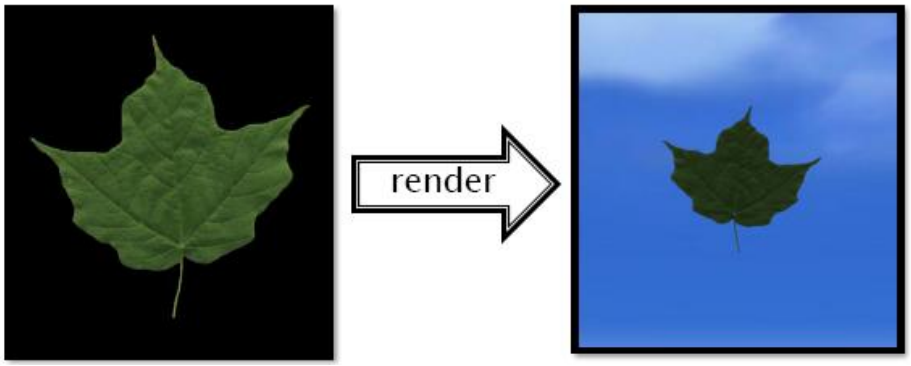

Fig. 17. Rendering leaf texture on top of mask.

These two steps are not sufficient for a decent rendering. Since alpha-blending is used, order of the rendering is important. For that reason, the leaves should first be sorted according to their distance from the camera position and then they can be rendered; if not, an occlusion problem would occur as in **Fig. 18**.

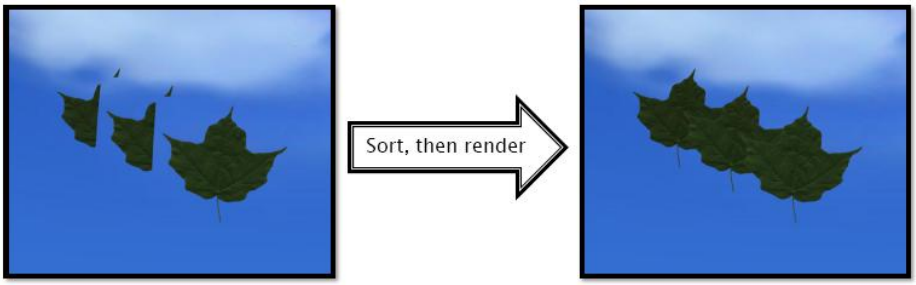

Fig. 18. The leaves first need to be sorted to avoid false occlusion problems.

4.5 Light and Shadow

As a realistic touch, tree shadow is also implemented. Since this is a real-time application, I want to keep shadowing simple for efficiency. Therefore, I have used a projective shadowing technique. It is an easy-to-implement type of shadow where the tree object is simply projected onto a plane, then rendered as a separate primitive. Computing the shadow involves applying an orthographic or perspective projection matrix to the model-view transform, then rendering the projected object in the desired shadow color. According to [11], the needed sequence is as follows;

1. Render the scene, including the shadowing object in the usual way.
2. Set the modelview matrix to identity, then call *glScalef(1.f, 0.f, 1.f)*.

3. Make the rest of the transformation calls necessary to position and orient the shadowing object.
4. Set the OpenGL state necessary to create the correct shadow color.
5. Render the shadowing object.

Fig. 19. Tree casting a shadow.

In the last step, the second time the object is rendered, the transform flattens it into the object's shadow (see **Fig. 19**). Obviously there are some trade offs of this technique. First of all, it is not easy to cast shadow on multiple planes; but since the scene had a flat surface, it did not cause any problem. Second, it is not very easy to control shadow color. Since the shadow is an actual object flattened to a surface; when you enable blending, there appears some artifacts at intersection regions of the shadowing object. In my case, this problem actually added some color differences in the shadow, which made it even more realistic.

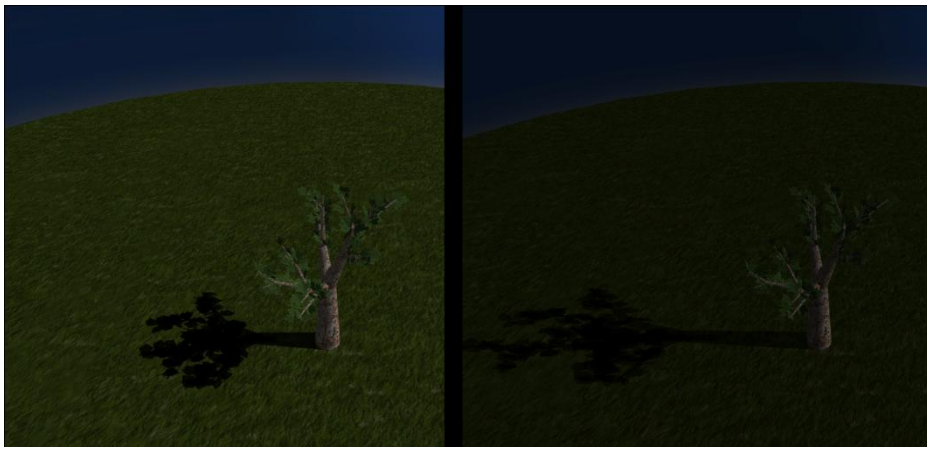

Fig. 20. As time gets late, the scene becomes darker and the shadow becomes more transparent.

To reach the realism even more, a lighting effect is implemented as well (see **Fig. 20**). There is a directional light source which changes its direction as time passes, just like a sun. And when this light source goes down and down, it diminishes. Also the shadow becomes more transparent. By using such simple tricks, I believe a good result is achieved in the name of simulating day/night circle.

5 Future Work

This system is a fully-functional tree simulation with a decent rendering. An essential extension for the current work could be to implement a tree generation algorithm. So far, tree models that have shown are generated by manual means, which is a long tedious work.

6 Conclusion

Final state of the implementation is really satisfactory, though; there is always room for cleaning and clarifying the code, as all application has. Without adding any new features to the system, only improvement that comes to mind is to port the simulation logic from CPU to GPU, for better performance. Other than that, it is a decent simulation of trees swaying in wind. I strongly suggest you to watch the video to better see the implementation results.

References

1. Shin Ota, Tadahiro Fujimoto, Machiko Tamura, Kazunobu Muraoka, Kunihiko Fujita, Norishige Chiba, "$1/f^{\beta}$ Noise-Based Real-Time Animation of Trees Swaying in Wind Fields," Computer Graphics International Conference, p. 52, Computer Graphics International 2003 (CGI'03), 2003
2. M. Aono and T. L. Kunii, “Botanical Tree Image Generation,” *IEEE CG&A*, Vol. 4, No. 5, pp. 10-34, 1984.
3. P. Reffye, C. Edelin, J. Francon, M. Jaeger, and C. Puech, “Plant Models Faithful to Botanical Structure and Development,” *Computer Graphics (SIGGRAPH ’88)*, 22(4), pp. 151-158, 1988.
4. R. Mech and P. Prusinkiewicz, “Visual Models of Plants Interacting with Their Environment,” *Computer Graphics(SIGGRAPH ’96)*, pp. 397-410, 1996.
5. Shinya, M. and Fournier, A., “Stochastic Motion – Motion Under the Influence of Wind,” *EUROGRAPHICS '92*, pp. C-119 - C-128, 1992.
6. Sakaguchi, Tatsumi and Ohya, Jun, “Modeling and Animation of Botanical Trees for Interactive Virtual Environments,” *Symposium on Virtual Reality Software and Technology (VRST99)*, pp. 139-146, December 1999.

7. Jos Stam, “Stochastic Dynamics: Simulating the Effects of Turbulence on Flexible Structures,” *Computer Graphics Forum (EUROGRAPHICS '97)*, 16(3), pp. 159-164, August 1997.
8. Thomas Di Giacomo, Stéphane Capo and François Faure, “An Interactive Forest,” *Eurographics Workshop on Computer Animation and Simulation*, pp. 65-74, September 2001.
9. Wei, X., Zhao, Y., Fan, Z., Li, W., Yoakum-Stover, S., and Kaufman, A. 2003. Blowing in the wind. In *Proceedings of the 2003 ACM Siggraph/Eurographics Symposium on Computer Animation* (San Diego, California, July 26 - 27, 2003). Symposium on Computer Animation. Eurographics Association, Aire-la-Ville, Switzerland, 75-85.
10. Long, J., Reimschussel, C., Britton, O., and Jones, M. 2009. Motion capture for natural tree animation. In *SIGGRAPH 2009: Talks* (New Orleans, Louisiana, August 03 - 07, 2009). SIGGRAPH '09. ACM, New York, NY, 1-1.
11. Tom McReynolds. Programming with OpenGL: Advanced Techniques. In SIGGRAPH "97 Course Notes, Course No. 9.4. 1997.