Generating Trees in the Presence of Wind

by

Everett Morse

A semester project, submitted to Dr. Mike Jones of Brigham Young University in partial fulfillment of the requirements for CS 658.

Department of Computer Science

Brigham Young University

April 15, 2010

Abstract

Trees have been modelled in phenomenological fashion to model competition for light and space. They have been simulated and modeled to show animation in the wind. There is little about modelling trees as if they grew in the presence of wind, especially where trees facing the wind would shelter neighboring trees. This work seeks to copy the algorithm used by Palubicki to model trees growing around each other and other objects in light competion and add in the effects of wind with similar computations.

1 Introduction

There has been much work on modeling trees. Some approaches focus on simulation of natural growth. Others focus on directability. There are several good approaches that generate trees by modeling competition for space and/or light [3] [5]. However, there is not much work on modeling tree growth in the presence of a prevailing wind.

Trees should grow away from the wind. Trees in a pack should show those on the leeward side less affected. In a simulation of one tree, leeward branches should be less affected, shielded by the windward ones. It may also be appealing to simulate random gusts that can break off branches as they get older, simulating the fact that older branches are more brittle, and possibly giving the effect of fewer branches down low. This could produce interesting results that may give better tree models for windy areas, like hill tops or the edge of a forest and a field.

If it were an accurate bio-mechanical simulation, various forest ecologists would be interested. However, this work aims to be more of a phenomenological approach. This work can still be of interest to movie and game artists who are looking for better tree models to place in scenes where wind would have an effect. It would be most useful for generating accurate-looking forests when combined with previous work, and thus it may be used to generate larger scenes. It could also be used to generate a single tree or a few trees next to each other. This would add just a little bit more realism to these artistic mediums.

Others have simulated tree growth in competition for space and light [3] [5]. This is effective, but it doesn't show trees bending away from wind, which is especially apparent on the edge of a forest and a field. Also, there has been work to model trees moving in the wind. Akagi's work uses grids to calculate fluid dynamics of the wind and can do so for groups of trees using varying-sized grids [6]. This is interesting and perhaps his optimizations can be applied to my voxel-based attempt, but this does not model the effects of the wind on growth as it is. Also, if the voxel method works well as-is for light competition then it should work for wind effects without needing optimizations.

This work will generate tree models by modeling growth in the presence of wind using a voxel-based

algorithm similar to Palubicki's [3]. It will model propagation of wind sheltering effects from windward branches and turn buds away from the wind. The resulting models should provide improved realism and artistic freedom.

2 Related Work

The first most relevant work is [3] since it describes Palubicki's method which is the starting point of this work. This project duplicates what they have done with space and light competition for colonization, using the same basic algorithm and equations. Palubicki's method for space colonization is to have buds with an ocupancy zone and perception volume that grow in the best direction observable. For light competition, shadow propagates from the buds to voxels below. For adding wind, this work uses a similar computation but in the direction of the wind. It will cause buds to grow away from (rather than towards) the wind. If there are two trees, the one behind will receive less wind. The crowns will avoid each other by combining with the space colonization rather than using only wind. Palubicki's method was effective in showing trees that grew in a manner that affected neighboring trees and produced good results.

Runions et al did something very similar to Palubicki in [5]. This technique is like the venation algorithm they presented earlier, which has target growth points scattered around and the branches grow node-by-node out towards them. This one is also an easy to understand model of space colonization. It doesn't do anything with wind. Perhaps it could be modified somehow, but it is not quite as amenable to such modification as Palubicki's method. In addition to modelling trees, it can be used to model vines. The results are also good.

[4] is Runion's original paper on modeling venation patterns in leaves, which was later applied to modeling trees. It is a space colonization algorithm similar to what Runions et al did later and related to this work. It attempts to model auxin sources which are one biological explanation for how venetion patterns arrive, though controversal.

[1] is a very tedious, numerical, biomechanical simulation of tree growth, accounting for mass of the tree as it grows, competition in space, etc. This work focusses more on phenomological methods thus Fourcaud's work is not helpful as a starting point. The paper does mention the effects of wind, though perhaps does not include them in the final model. It talks about stresses on the tree which included gravity and wind.

[2] was referenced by Palubicki's paper. It is the basis for Palubicki's shadow propogation via voxel models which this work copies to use for wind calculations. It is an early example of work on the sort of algorithms employed by Palubicki and this work.

3 Project Description

This work uses an algorithm with a voxel grid indicating the wind sheltering effects at a location and then grows buds away from wind. The direction of prevailing wind is generally sideways, but could be at an angle. Wind should cause branches to bend away from it, so instead of having the buds turn towards voxels of higher value as in light competition models, it would turn away from them, which is similar to what space competition does, though the algorithm is more similar to the light competition algorithm. In future work it may also include some measure of branch age and a random change that a gust would remove the branch.

3.1 Initial Setup

First I created an environment in which to generate trees and view the results. This environment uses GLUi for a user interface on top of OpenGL. I spent a long time working on tree rendering code that draws tapered cylinders (truncated cones) for branches based on a tree data structure that connects 3-space points and stores the thickness of the tree at each point. There are also alternate rendering methods that just draw lines or that draw lines with spheres at each point to represent the thickness. The final rendering algorithm has some visible errors, possibly due to rounding issues, but is able to draw trees in 3-space effectively enough.

In order to have some tree to look at while building the rendering algorithm, I included a hard-coded, simple tree. Then I built out a general tree generator architecture. Each generator has an initialization step, an iteration step, and a method to run the iterations to completion. Buttons on the GUI activate these features. There is also code that will calculate thicknesses for a tree according to Murphi's equations, which can be read easily in Runion's paper [5]. Those are $t = \sqrt[N]{t_1^N + t_2^N}$ where a good number for N is 3 and t_1 and t_2 refer to the branch's two thickest children. A GUI button will clear the thicknesses and re-calculate them. I implemented a random tree generator which has a branching factor of two and will randomly choose a branching angle for each branch and a random offset in the Z coordinate, producing trees that are mostly two dimensional but have a slightly three dimensional aspect. Figure 1 shows one such tree.

I also implemented load and save methods. The random generator algorithm uses a stack to traverse the tree freshly for each iteration then add to the childless nodes. This can take a while to run after several iterations. Thus it is worthwhile to save the results for later viewing without having to regenerate the trees. This should be useful for the final product tree models too.

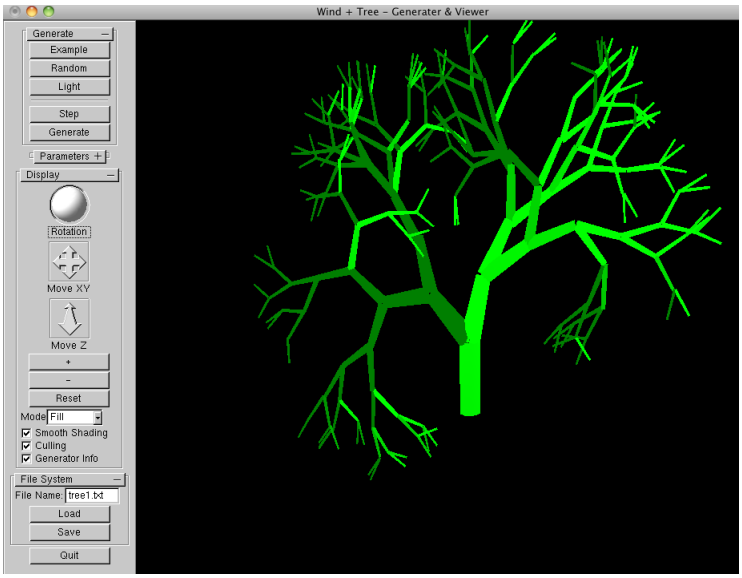


Figure 1: A randomly generated example tree in the GUI.

3.2 Voxel Grid

Next I implemented the voxel grid necessary for both Palubicki's light competition algorithm and the new wind effects algorithm. Figure 2 shows the debugging display of this grid. In this testing screenshot shadow was propagated down from two locations at a depth limit of five. The size of the purple boxes is proportional to the percentage of max shadow or shelter.

The algorithm in Palubicki's paper only described propagating shadow directly down. I copied this algorithm to propagate directly sideways. In future work I could make a more general algorithm to take an arbitrary direction, which would be better for wind and even helpful for light. As it is, wind generally comes from the side and can be effectively modelled that way.

3.3 Light Competition

The first step towards making the final algorithm is to duplicate Palubicki's competition for light algorithm. This involves tracking the buds, calculating their perception volumes, finding the optimal direction based on the voxel grid, and growing in that direction.

Growth occurs at buds and creates shoots. Decurrent growth in my model (using one of Palubicki's choices) is done with proleptic development and sympodial branching. This means that new buds produce shoots in the next season (iteration) and that buds at the end of shoots die (or make flowers, but I'm not drawing flowers). This produces bush-like structures. To make trees some form of apical control is needed [3].

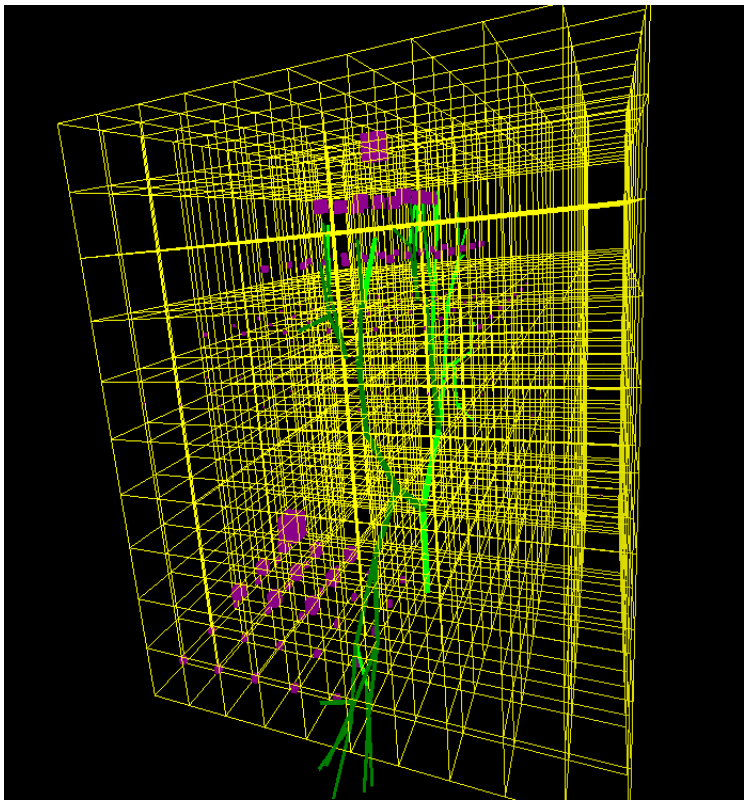


Figure 2: Debugging info shows the voxel grid and shadow/shelter values.

To determine the length of growth I use Palubicki's priority model. This produces the most tree-like results based on Palubicki's figures. Each tree node tracks meta information. In pass one, each bud calculates the amount of light it receives and the total of this and the number of buds are passed down to the axis root. An axis is one straight shoot growth. Nodes store a reference to their axis root to make this pass easy. In the next pass, for each axis, buds and branches are prioritiezed based on the average light per bud and resources are allocated according to this priority as per

$$v_i = v\frac{Q_i w_i}{\sum_{j=1\ldots N} Q_j w_j}$$

for each i up to N where N is the number of buds on the branch, v_i is the amount of resource allocated to the bud or branch, and Q_i is the amount of light exposure for the bud. The weights are done according to a piecewise function from w_{max} to w_{min}, where it goes linearly towards min and then stays there at κN. The equation for light exposure is

$$Q = max(C - s + a, 0)$$

where C is maximum exposure, s is the shadow value in the voxel, and a is the base shadow amount

subtracted because buds do not shade themselves.

In order to get a more tree-like result, apical control is applied, which takes the terminal bud of a branch and places it on the top of the priority list for distributing resources. The apical control modes I implemented are: full apical control, apical control on main branch only, no apical control, and decurrent (as described above). It is useful to apply apical control early in development and then turn it off or switch to decurrent mode later in development, as this mimics the growth patterns of some trees.

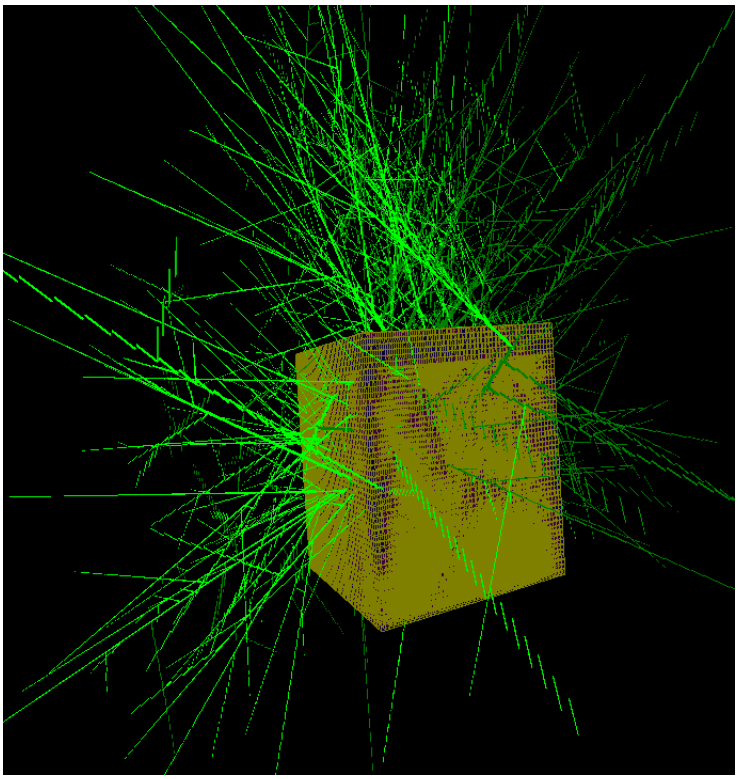


Figure 3: Tree gets less organized if it escapes the voxel grid. Parameters were readjusted to make the grid larger and the tree smaller.

Getting good results also depends on the length of the metamer segments, the size of the voxels, and how big the voxel grid is. Figure 3 shows the problems of the tree escaping the voxel grid. When it does so, the calculated voxel for a bud is the closest in-bounds voxel, resulting in lots of shadow on the edges of the grid and less controlled results for branches outside the grid.

3.4 Wind Effects

Finally wind is added in addition to light competition. The light competition is still needed to keep branches from running into each other. Wind simply tilts the optimal growth direction vector for a bud away from the wind proportional to the amount of wind observed at the voxel in which the bud resides.

This is easily done by extending the LightTree class to add an additional step to determining bud growth vectors. The sub class must also overload the method for propagating shadow to additionally propagate wind in the separate wind grid. Wind amounts in the grid at the point of the bud are taken using the same formula as for shadow multiplied by a factor β. This amount is multiplied by the unit vector in the wind direction and added to the growth vector of the bud, which is then re-normalized.

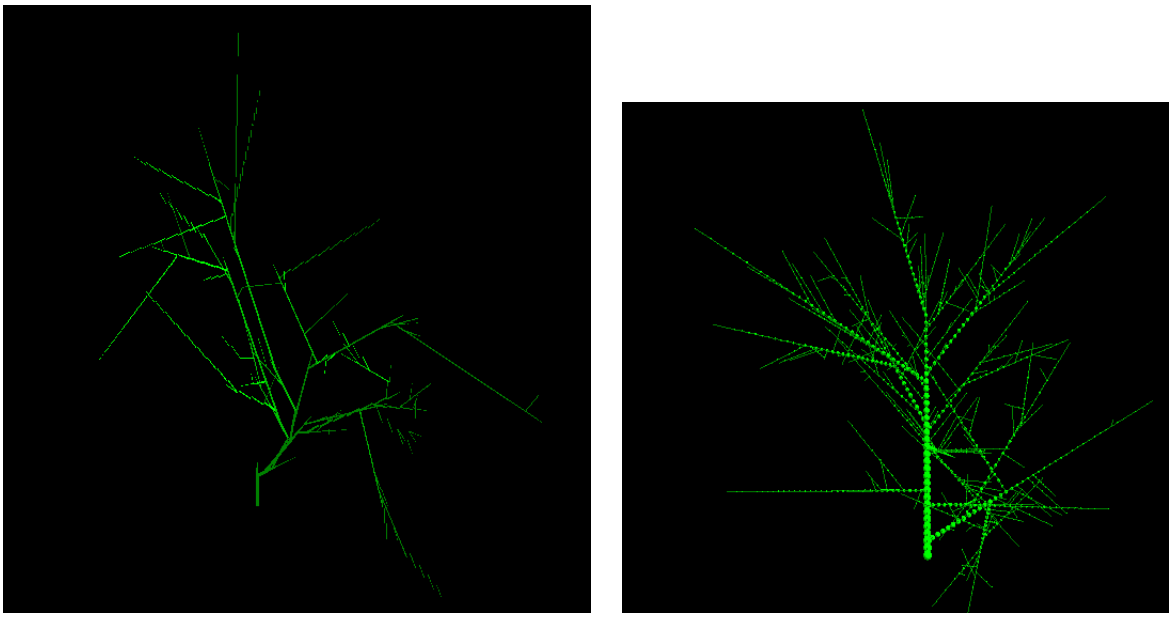
Wind can be applied to all buds, or we can refrain from applying it to terminal buds so that branches stay straight and only the direction of new shoots is affected. When branches bend, a smaller β is needed, but if branches don't bend, then *beta* can be larger to have a noticeable effect.

Figure 4: Debugging information shows wind values in voxel grid.

4 Validation

The only real way to validate the results are to compare the resultant models to real trees to see if the results look realistic and provide the desired wind effects. They could also be compared to models produced by other methods, though other methods do not model wind effects.

The results should demonstrate branches bending away from the wind on the windward side and less effects on the leeward side. Trees that are sheltered by others should show less effects. The algorithm



a b

Figure 5: Trees rendered with Palubicki's competition for light. (a) Decurrent (b) Using apical control.

should run quick enough to be usable, though speed is not the focus of this work.

The results shown below demonstrate the effectiveness of this algorithm. They appear to be satisfactory. Many more trees can be generated by varying the parameters, which can be made to meet various artistic requirements or to approximate real tree examples.

5 Results

Here are the results of the algorithms, to compare against expectations and real trees. This includes just a few examples. Some are rendered as line segments for a metamer and shperes to represent the thickness of the tree at the point. Some are rendered as tapered cylinders, but that algorithm still has a few bugs in it. These pictures should give a good idea of the effectiveness of this work.

Firstly, Figure 5 shows trees rendered using only Palubicki's conpetition for light algorithm. This algorithm is fundamental to the wind competition algoritm. Trees with wind effects are just trees with light competition that have been blown a little. The same kinds of trees are possible in the wind algorithm, including decurrent and excurrent (with apical control) forms.

Figure 6 shows a tree using competition for light with apical control that has wind effects added. The branches were allowed to bend as they grew (though shoots in an iteration are straight). The result is quite pleasing.

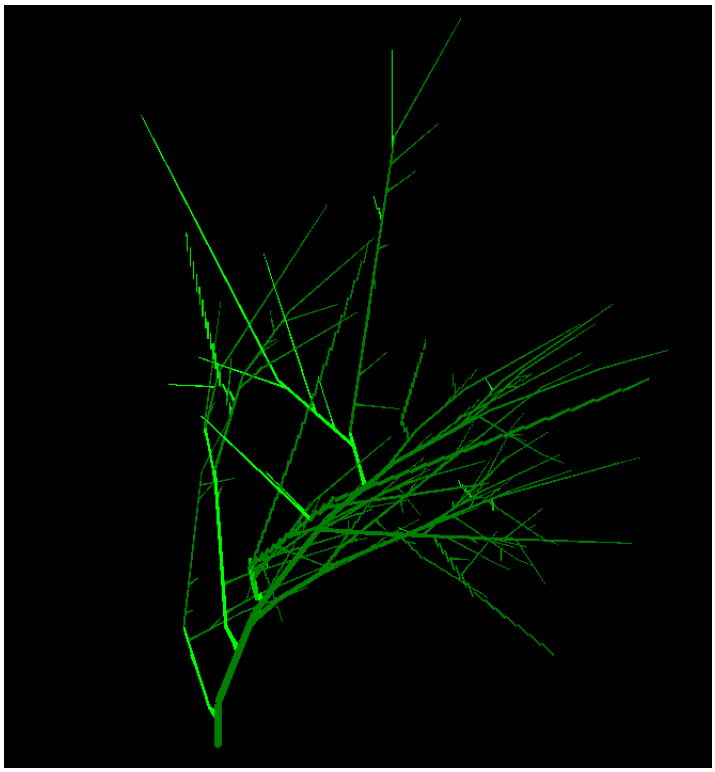


Figure 6: A tree with wind effects added. $\beta = 0.125$, branches can bend.

Figure 7 shows some trees rendered using the wind algorithm where buds on the ends of branches are unaffected by wind, so that branches do not bend. Different values of β are used to demonstate the kinds of trees possible with various wind forces. Figure 7(a) has little wind, but still demonstrates that branches on the windward side have been blown back a little, widening the angles between them, while branches on the leeward side have been blown straigher, lessoning the angle between them.

Reproducing the competition for light algorithm from Palubicki was a challenging task. Once done, however, the results are reasonable. Adding in effects from wind produces immediately visible results which have reasonably good effects. Both algorithms have many parameters that can be tweaked to produce a variety of different trees. The results are thus satisfactory.

.

.

.

.

.

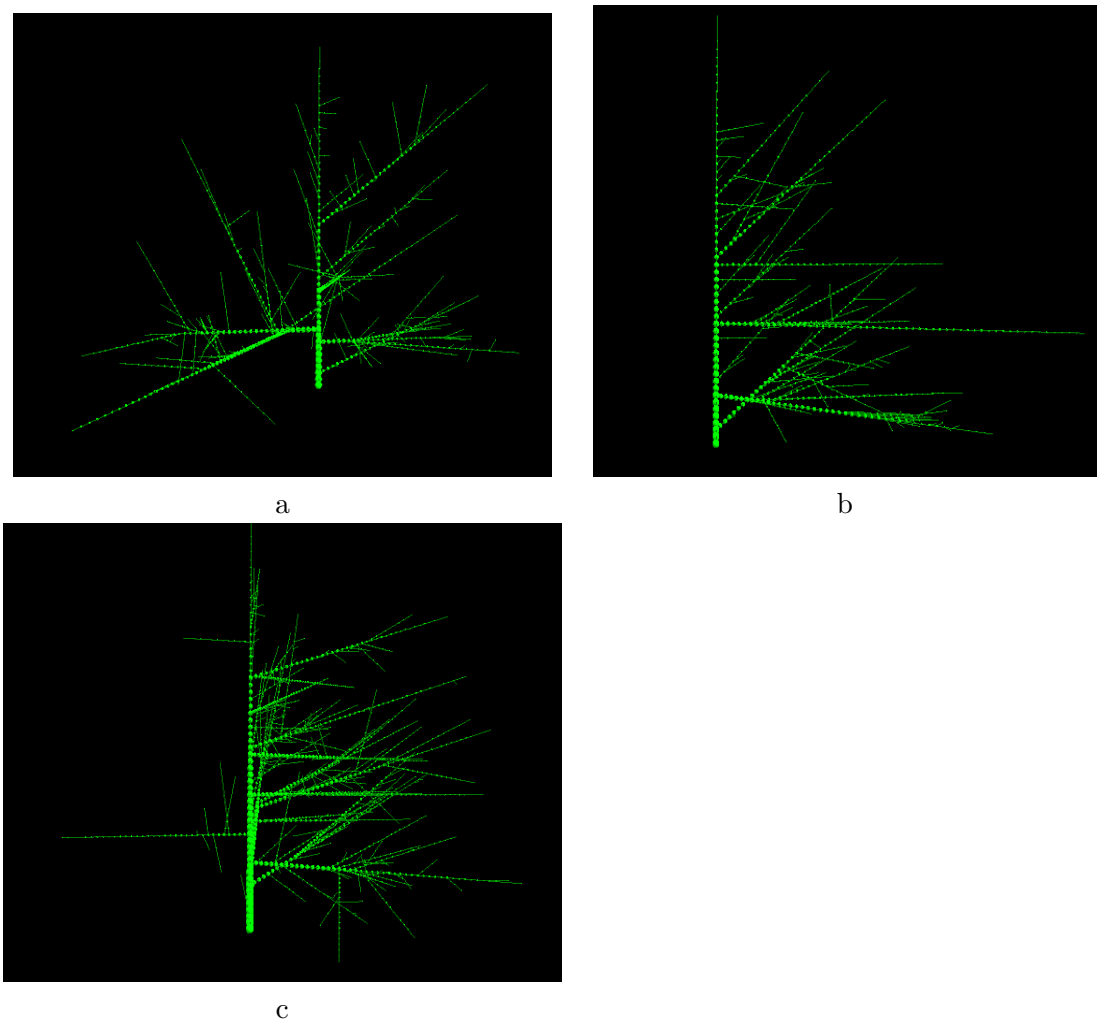


Figure 7: Trees with wind effects. Branches don't bend. (a) $\beta = 0.125$ (b) $\beta = 0.50$ (c) $\beta = 0.25$.

References

[1] Thierry Fourcaud, Frédéric Blaise, Patrick Lac, Patrick Castéra, and Philippe de Reffye. Numerical modelling of shape regulation and growth stresses in trees. *Trees - Structure and Function*, 17(1):31–39, 2003.

[2] N. Greene. Voxel space automata: modeling with stochastic growth processes in voxel space. In *Transactions of Information Processing Society of Japan*, pages 175–184, New York, NY, USA, 1989. ACM.

[3] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch, and Przemyslaw Prusinkiewicz. Self-organizing tree models for image synthesis. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–10, New York, NY, USA, 2009. ACM.

[4] Adam Runions, Martin Fuhrer, Brendan Lane, Pavol Federl, Anne-Gaëlle Rolland-Lagan, and Przemyslaw Prusinkiewicz. Modeling and visualization of leaf venation patterns. *ACM Trans. Graph.*, 24(3):702–711, 2005.

[5] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling trees with a space colonization algorithm. In *Eurographics Workshop on Natural Phenomena*, 2007.

[6] Akagi Yasuhiro, Sanami Sho, and Kitajima Katsuhiro. Computer animation of swaying trees based on physical simulation. *Trees - Structure and Function*, 46(7):1797–1809, 2005.