EUROGRAPHICS 2009 / P. Dutré and M. Stamminger
(Guest Editors)

Volume 28 (*2009*), *Number 2*

Physically Guided Animation of Trees

Ralf Habel[1], Alexander Kusternig[1] and Michael Wimmer[1]

[1]Institute of Computer Graphics and Algorithms
Vienna University of Technology, Austria

Abstract

This paper presents a new method to animate the interaction of a tree with wind both realistically and in real time. The main idea is to combine statistical observations with physical properties in two major parts of tree animation. First, the interaction of a single branch with the forces applied to it is approximated by a novel efficient two step nonlinear deformation method, allowing arbitrary continuous deformations and circumventing the need to segment a branch to model its deformation behavior. Second, the interaction of wind with the dynamic system representing a tree is statistically modeled. By precomputing the response function of branches to turbulent wind in frequency space, the motion of a branch can be synthesized efficiently by sampling a 2D motion texture.
Using a hierarchical form of vertex displacement, both methods can be combined in a single vertex shader, fully leveraging the power of modern GPUs to realistically animate thousands of branches and ten thousands of leaves at practically no cost.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Natural Phenomena,Vegetation,Tree,Animation—

1. Introduction

Plants, especially trees, are an important element of many interactive applications. While there have been significant advances in the realistic *rendering* of trees, it is probably the quality of *animation* that has most influence on how convincing a tree looks. Visually convincing animation of trees in real time is a difficult problem due to several factors. Trees are geometrically very complex, consisting of thousands of branches and leaves. All these geometric elements are connected in a complex dynamic system that is hard to solve. The characteristics of tree animation are determined mainly by the high-frequency behavior, for example the jittery movement of leaves even in light wind. This cannot be expressed using simple approximations like simple sinusoidal movement of tree branches. As yet, there are only few methods that provide real-time animation of trees while addressing at least some of these factors.

In general, animating a tree involves different components: a *wind model* which describes the characteristics of turbulent wind interacting with a tree; the *dynamic system*, which describes how the interconnected branches and leaves react to the external wind force; the *geometric model*, which describes how the forces resulting from the dynamic system affect the geometry of the tree by for example bending the branches; and the *structural model* which describes the hierarchical organization of the geometry.

The prevalent structural model for trees is skinned skeletal animation. This model has the disadvantage that it places a burden on the CPU because the bone matrix for each branch has to be recalculated every frame. Usually this is combined with a rigid geometric model, i.e., each sub-branch (between joints) is rigid and does not deform (bend). Some approaches introduce limited bending by adding more joints, leading to even more overhead. Furthermore, most real-time tree animation approaches have greatly simplified dynamics, for example by using a static mass-spring model instead of integrating forces over time.

In this paper, we introduce a new real-time tree animation method that is both significantly faster and more physically plausible than previous methods. Quality is improved using both a physically based geometric and dynamic model, leading to realistic bending and swaying motion of branches in the wind. Speed is improved by choosing all elements so that they can be localized to a vertex: in the structural model, skeletal animation is replaced by *hierarchical vertex displacement*, which directly evaluates the branch hierarchy in the vertex shader; the geometric model is expressed using an explicit analytic formula for the *Euler Bernoulli beam model* that has no dependencies on other parts of the tree; and the *dynamics model* is calculated in the frequency domain and evaluated using noise functions.

While deformation of branches is rarely even considered in real-time animation, our geometric model even improves on previous offline methods, which are based on the common assumption of uniform-thickness beams. We show an efficient analytic deformation operator for tree branches, whose thickness vary significantly along the length, causing different bending behaviors. We also show the corresponding analytic formulation for transforming normals and tangents for realistic shading of the deformed trees.

The main observation underlying our dynamics model is that most of the visual effect caused by the dynamic system in a tree can be obtained by looking only at the *decoupled* system, i.e., where branches have no direct influence on each other [CGZ*05]. This is plausible because there is in general little overlap in the vibration frequencies of connected branches. We model each branch as a damped harmonic oscillator, and calculate its interaction with wind in frequency space. For efficient evaluation of the result in primary space, we introduce *2D motion textures*, which provide an extremely efficient way to generate thousands of functions for the whole tree while avoiding periodicity, and which can be evaluated in the vertex shader.

Thus, all computations are done in the vertex shader and are usually hidden by shading computations. Furthermore, we provide a set of intuitive parameters which control the animation and can be changed in real time.

2. Related Work

Previous approaches for simulating the dynamic system of trees can be roughly categorized in full simulation approaches, based on a computationally expensive integration of the equations of motion ([AK06,ZST*06,SF92,ZST*06, SO99]) and heuristic approaches ([SFT*03, WS05, Zio07, Sou07]) which do not consider the physical properties of trees. There are also hybrid techniques [GCF01, WVHR06]. All of these typically make use of stochastic approximations, especially for wind.

Stam [Sta97] carries out the simulation in the frequency domain, reducing computation time by modal analysis. The result is a precalculated stochastic periodic motion. [CGZ*05] uses similar spectral methods to model a large range of natural phenomena. For trees, they consider a simplified, uncoupled dynamic system based on a harmonic oscillator per branch, which is also the basis of our texture based motion synthesis. The basis of this simplification is that an observer cannot judge the correctness of the response function of a highly complex dynamical system because the wind itself is not visible. Thus, the characteristics of the animation are mainly determined by the frequencies and amplitudes of branches. In comparison to these methods, we introduce 2D motion textures to efficiently generate a large number of aperiodic noise functions.

It turns out that for certain sets of tree parameters, the resulting spectral response function has roughly the shape of $1/f^{\beta}$, which has been exploited by [SFT*03] to drive motion directly by $1/f^{\beta}$ noise.

A common geometric model for representing tree structures is based on deformable uniform thickness beams [SF92, CGZ*05], while our method on the other hand correctly takes taper and length into account, which has an essential impact on the deflection of a branch. Interactive methods, on the other hand, usually do not consider deformation of branches [SO99, SFT*03]. Joint segmentation could be used, but would get prohibitively expensive for highly detailed trees. Our example tree would require about 30000 joints with the corresponding hierarchy and smooth skinning.

3. Hierachical Vertex Displacement

The key element for real-time performance is to *localize* all computations in a vertex shader, leading to so-called vertex displacement. This is often used to animate simple vegetation represented as billboards (e.g., grass) or billboard clouds (e.g. simple tree models [Sou07]). For hierarchical, full-geometry tree models, this is not straight-forward.

The main idea is to expose the relevant part of the tree structure (i.e., hierarchy) to every vertex. Thus, the hierarchical deformations of all parent branches can be explicitly performed inside the vertex shader and no information needs to be propagated at runtime. We can achieve this by assigning each vertex an index into a texture that holds all information necessary for that vertex, so every vertex within a branch has the same index. Additionally, since we allow sub-branches to emanate at any position of its parent branch, the relations of each vertex to all parent hierarchy levels are required. For this, we precalculate the normalized local coordinate $x \in [0..1]$ of the vertex, where x is along the principal axis of a branch. We also calculate the x-values at parent-branch connections and propagate them down the branch hierarchy (see Figure 1). These are stored with each vertex in addition to its own x-value. The trees in our examples have 4 hierarchy levels, so each vertex has a vector $\vec{w}$ of 4 values associated in addition to the branch index.

In order to allow non-rigid deformations, the deformation model needs to be able to correctly transform the local coordinate axes between hierarchy levels. For this, we introduce a novel two step deformation method in Section 4 based on a structural mechanics model that takes the basic physical properties of a branch into account. The model has a closed

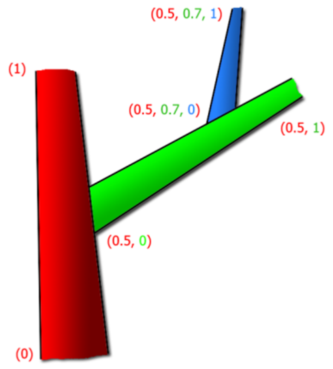

Figure 1: *$\vec{w}$ distribution of branches in a tree.*

form solution, allowing for the required tangent and normal transformation.

To animate this model, Section 5 covers how the motion affecting each individual beam are modeled using a wind model and a dynamic system.

Applying these two main components of our system to actual tree geometry is explained in Section 6, while Section 7 shows the application of both methods to leaves.

4. Beam Model

The model used for describing the geometry and physics of a beam determines how realistic branches swaying in the wind appear to the viewer. A common approximation for realistic animation systems is to model the beam as an elastic cylinder (uniform beam), and describe the deformation due to a uniform traversal force using a polynomial deflection function depending on some basic physical properties of the beam. However, uniform beams are not a good approximation of tree branches, leading to unrealistic bending. As can be seen by the direct comparison in the accompanying video, the tips of branches are more flexible while thicker parts are stiffer which is not accounted for in uniform beams.

In this section, we describe a beam model that is more appropriate for branches in combination with a method to maintain the length of branches even though we make use of transversal deflection without breaking the restrictions of hierarchical vertex displacement.

4.1. Euler-Bernoulli Beam Model

The Euler-Bernoulli Beam Model is a structural mechanics model for long and thin beams with a length to thickness ratio of 15:1 and above, and is therefore suitable for trunks and branches [TYW74]. A branch is treated as a linearly tapered circular beam as seen in Figure 2, defined by its length L and the radii s_1, s_2 at the root and free end, thus incorporating the essential physical properties of a branch or trunk.

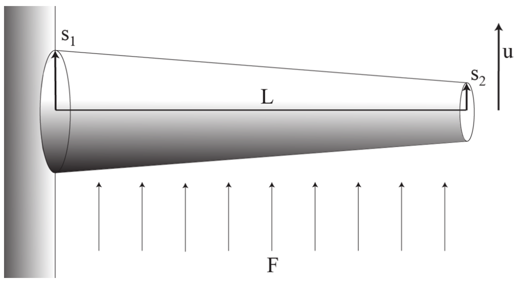

Figure 2: *Beam model used to calculate the deflection of branches*

The deflection of a branch is described by the Euler-Bernoulli differential equation

$$\frac{\partial^2}{\partial x^2}\left(EI(x)\frac{\partial^2 u(x)}{\partial x^2}\right) = F \tag{1}$$

where $u(x)$ is the (unknown) deflection of the beam according to the constant transversal force F. I is the area moment of inertia (not to be confused with the mass moment of inertia), and E is the elastic modulus, which we assume to be constant.

As each branch is fixed at its root, the boundary conditions at the root are

$$u|_{x=0} = 0 \qquad \frac{\partial u}{\partial x}|_{x=0} = 0. \tag{2}$$

The boundary conditions at the free end are

$$\frac{\partial^2 u}{\partial x^2}|_{x=L} = 0 \qquad \frac{\partial^3 u}{\partial x^3}|_{x=L} = 0 \tag{3}$$

To simplify the solution and to fit it to the hierarchical vertex displacement data in Section 3, we normalize the model to unit length by scaling the radii by L, and introduce the taper ratio α of the two radii

$$r_{1,2} = \frac{s_{1,2}}{L} \qquad \alpha = \frac{r_2}{r_1} \tag{4}$$

and a rescaled elasticity modulus $E' = EL$. The area moment of inertia for a circular area corresponding to the beam axis is

$$I = \frac{\pi r^4}{4} \tag{5}$$

with the radius varying linearly over the length of the beam. Here the importance of the taper of a branch becomes evident as the area moment is a quartic function of the radius, thus significantly affecting the resulting deflection as the radius varies. The area moment of inertia along the beam results in

$$I(x) = \frac{\pi r_1^4((\alpha-1)x+1)^4}{4}. \tag{6}$$

Under the given boundary conditions (2) (3) and varying $I(x)$ (6), the Euler-Bernoulli Equation (1) has an analytical

α	c_2	c_4	Δx_{max}
0.05	0.221875	0.754029	0.0469
0.1	0.3326	0.398924	0.0024
0.2	0.374571	0.129428	0.0081
0.3	0.364816	0.024577	0.006

Table 1: *Coefficients for equation (7) of different values of the taper ratio α.*

yet not trivial solution which is given in Appendix A. It is interesting to note that the force F affects the solution linearly, which means that the amplitude of the deflection $u(x)$ is proportional to F, while the root radius influences the deflection with r_1^{-4}. Figure 3 shows the deflection for different taper ratios normed to $E'F/r_1^4 = 1$. As can be seen, the taper

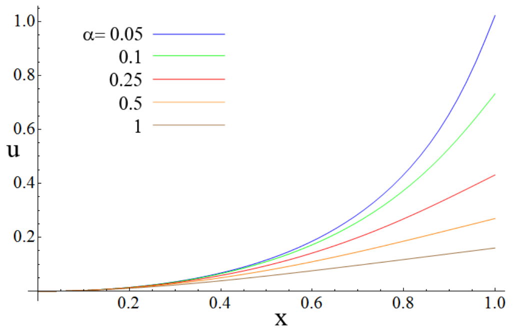

Figure 3: *Deflection for different taper ratios. With decreasing taper ratio, the beam deflection increases due to the thinning of the branch.*

of a branch has a very strong influence on both the deflection amplitude and the form of the deflection and can not be neglected in a realistic branch model. A beam with no taper ($\alpha = 1$) as commonly used (e.g. [SF92]) deflects in a way that the free end is nearly linear, compared to a very curved and stronger deflection along the beam at small taper ratios.

In order to speed up the calculation of $u(x)$, which will be done in a vertex shader, we perform a least squares fit of equation (25) to the polynomial

$$u(x) = c_2x^2 + c_4x^4 \tag{7}$$

depending on the taper ratio of each branch. Table 1 shows coefficients for some taper ratios, along with the maximum absolute error Δx_{max} of this approximation. For taper ratios above 0.1, the fit is virtually exact. Through the fit, α, E' and r_1 are represented by the coefficients c_2 and $c4$.

4.2. Length Correction

The Euler Bernoulli beam model works well for small deflections, but strong deflections (amplitude $> L/4$) show noticeable stretching since the deflection is only applied in the transverse direction. Unfortunately, the exact incorporation of the length requires the solution of an elliptical integral [BD45] which can not be formulated explicitly. In order

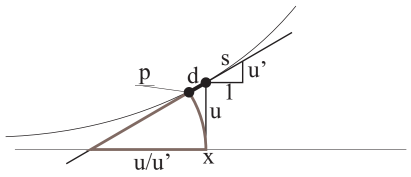

Figure 4: *Geometry of length correction (parameter x omitted for clarity).*

to length correct a deflected branch, we correct the stretch introduced by the shear of 7 by moving the deflected vertex along its tangent, effectively converting the shear into a rotation. Using the stretch factor of the shear (which is also the length of the tangent vector), $s(x) = \sqrt{1+u'^2(x)}$, the local length difference $d(x)$ of the original and deflected beam is

$$d(x) = \frac{u(x)}{u'(x)}(s(x) - 1) \tag{8}$$

Starting from an original point on a branch $\vec{p}_o = (x,y)^T$, the uncorrected point would be $\vec{p}_u = (x, y+u(x))^T$. The final deflected point $\vec{p}$ can be found by moving the originally deflected point $\vec{p}_u$ along the tangent direction to unstretch the beam (see Figure 4):

$$\begin{aligned} \vec{p}_x &= x - \frac{d(x)}{s(x)} \\ \vec{p}_y &= y + u(x) - \frac{u'(x)d(x)}{s(x)} = y + \frac{u(x)}{s(x)} \end{aligned} \tag{9}$$

with $x \in [0..1]$. Thus, the original point is deflected using the offset vector $o(x)$:

$$\vec{p} = \vec{p}_o + \vec{o}(x), \qquad \vec{o}(x) = \frac{1}{s(x)}\begin{pmatrix} -d(x) \\ u(x) \end{pmatrix} \tag{10}$$

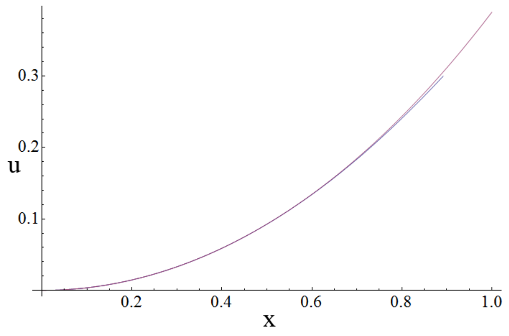

Figure 5: *Strongly deflected and length corrected beam.*

Figure 5 compares a strong deflection to its corrected deformation and a direct comparison of a length corrected to a beam without length correction can be seen in the accompanying video.

4.3. Branch Deformation

The previous subsection derived a 2D deflection operator for a unit-length beam. In order to apply the deflection and length correction to the vertices of an arbitrary branch, we need to express the deflection in the local coordinate system of the branch and un-normalize to the original branch length L. We assume that each vertex is given in the local coordinate frame $(\vec{t},\vec{r},\vec{s})$ of the branch, where $\vec{t}$ goes along the axis of the beam (see Figure 6).

Let $\bar{x}$ be the coordinate along $\vec{t}$, then $x = \bar{x}/L, x \in [0..1]$, and the unnormalized deflection function $\bar{u}(\bar{x}) = u(x)$. By the chain rule, $\partial\bar{u}/\partial\bar{x} = 1/L\partial u/\partial x$. To take care of the amplitude of the deformation, we project the amplitude $\vec{A}$, which is proportional to the net force acting on the branch, onto $\vec{r}$ and $\vec{s}$, and multiply the deflection with the corresponding amplitudes A_r and A_s to arrive at deflection curves in the two directions, using the normalized coordinate x:

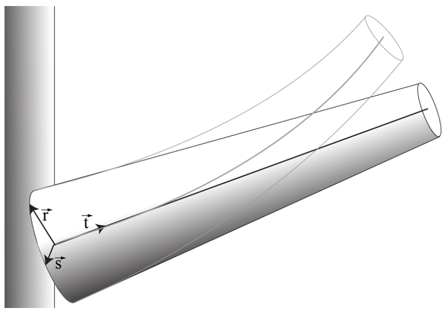

Figure 6: *Local coordinates for a branch*

$$u_{r,s}(x) = A_{r,s}u(x), \qquad u'_{r,s}(x) = A_{r,s}\frac{u'(x)}{L} \tag{11}$$

Please note that $A_{r,s}$ does not directly set the strength of the deflection since taper ratios and radii vary from branch to branch. The resulting 3D position is thus

$$\vec{p} = \vec{p}_o + \vec{o}(x), \quad \vec{o}(x) = \begin{pmatrix} -d_r(x)/s_r(x) - d_s(x)/s_s(x) \\ u_r(x)/s_r(x) \\ u_s(x)/s_s(x) \end{pmatrix} \tag{12}$$

To transform tangents and normals, we would have to calculate the Jacobian J_p of this transformation. However, the length-corrected deformation leads to several complex higher-order terms, which are difficult to evaluate in real time. We therefore use the Jacobian of the non length-corrected deflection and evaluate it at the length corrected position

$$J_{p_u} = \begin{pmatrix} 1 & 0 & 0 \\ u'_r(x - d_r(x)/s_r(x)) & 1 & 0 \\ u'_s(x - d_s(x)/s_r(x)) & 0 & 1 \end{pmatrix} \tag{13}$$

and find the deflected (unnormalized) tangents as $\vec{t} = J_{p_u}\vec{t}_o$, and the deflected (unnormalized) normals as $\vec{n} = J_{p_u}^{-T}\vec{n}_o$, where $\vec{t}_o,\vec{n}_o$ are the tangent and normal vectors of a vertex projected into the local frame of a branch. The error introduced is minimal (a maximum 2.4 degrees for the tip of the beam) since the derivatives of the deflected curve and the length corrected curve are virtually the same for the same x (see Figure 5). Section 6 will show how this deformation can be implemented hierarchically.

5. Synthesizing Branch Motion

A tree interacting with wind is a highly complex dynamic system that is difficult to solve through numerical simulation and we also want to uphold the restrictions of hierarchical vertex displacement, not allowing any access to previous states. Since the characteristics of branches swaying in wind is mainly determined by the vibration frequencies of the branches, we use *spectral methods* similar to [CGZ*05] to synthesize branch motion to drive the branch deformation model. The principal idea is to generate noise functions that obey similar frequency distributions as empirically observed data or results of a full simulation. In this section, we show an optimized method of synthesizing noise functions that allow fully aperiodic motion for a whole tree using only a small number of noise textures, making the method well suited for real-time applications.

5.1. Turbulent Wind

Trees cause the wind acting upon them to become much more turbulent than free flowing wind. The turbulence in the wind field is dominant over the directional contribution, making it hard to tell the wind direction from tree motion (at least in low to medium wind). This makes it possible to model wind velocity using its power spectrum. A common model has been created from empirical data [SS86]:

$$P_w(f) \propto \frac{v_m}{(1+f/v_m)^{\frac{5}{3}}} \tag{14}$$

where v_m is the mean velocity of the wind.

5.2. Motion Spectrum

The overall motion of a branch can be approximated using the physical model of a damped harmonic oscillator, incorporating the dynamic properties of a branch such as its resonance frequency f_h, mass m, and damping γ_h caused for example by the leaves' resistance to wind and internal damping. While coupling due to the branch hierarchy can be incorporated at higher processing and memory cost [Sta97], a reasonable assumption is that each branch oscillates about its root independently [CGZ*05]. This is mainly because due to the typically different branch lengths at different hierarchy levels, the resonance frequencies are far apart.

The amplitude spectrum of the *stationary solution* of a harmonic oscillator with its resonance frequency f_h and damping γ_h, driven by an external force (14) is given by:

$$V_h(f) = \frac{V_w(f)}{2\pi m(2\pi(f_h^2 - f^2)^2 + (2\pi f \gamma_h)^2)^{\frac{1}{2}}} \tag{15}$$

where $V_w(f)$ is the force spectrum of wind. We introduce a light form of coupling between branches by calculating the mass m of each oscillator from the branch itself and all its subbranches, with ρ_w being the density of wood:

$$m = \rho_w \sum_i \frac{\pi L_i}{3}(s_{i,1}^2 + s_{i,1}s_{i,2} + s_{i,2}^2) \tag{16}$$

5.3. Stochastic Motion Synthesis

To synthesize motion in the time domain, we generate noise in the frequency domain, modulate it by wind and oscillator response functions, and transform back to time domain – hence the name "spectral method". More specifically, we first generate a "noisy" wind force spectrum by modulating a random Gaussian field $G(f)$ by the square root of the power spectrum of wind $P'_w(f) = G(f)\sqrt{P_w(f)}$. Similar to [CGZ*05], we assume wind force to be proportional to wind velocity: $V_w(f) \propto P'_w(f)$. Plugging this into (15) and taking the inverse Fourier transform of $V_h(f)$ gives the motion texture in the time domain. Please note that we do not consider phase shift since the phases of the Gaussian field as well as the resulting velocity spectrum are both uniformly distributed. Other noise functions such as Perlin noise can be used for a non physical approach, but they can not account for the resonance frequency of branches and a correct wind spectrum.

5.4. 2D Motion Textures

The process described so far gives 1D noise functions in the time domain. Realistic trees have thousands of branches, requiring a massive number of different noise functions, all of which should be aperiodic.

The main idea of our motion synthesis method is to generate *2D motion textures* instead of 1D functions. Observe that the velocity spectrum can be written as $V_h(f) = G(f)H(f)$, with $H(f)$ representing the combined spectral response function of the harmonic oscillator and wind. We start from a 2D random Gaussian field $G(x,y)$ and calculate a 2D velocity spectrum $V_h(x,y) = G(x,y)V_h(\sqrt{x^2+y^2})$. The corresponding 2D motion texture is the inverse Fourier transform of $V_h(x,y)$.

Trajectories. Let us define a linear trajectory of a texture sample point with a random 2D vector $\vec{m}$, sampling the 2D-periodic (due to the fourier transform) motion texture with texture repeat for values outside the unit square. Since the spectrum is radially symmetric, each such trajectory creates a 1D signal with a spectrum of $V_h(f)$. Furthermore, such trajectories are *aperiodic* as long as the trajectory does not close on itself. To avoid this case, we test for the irrationality of the vector component ratio m_x/m_y and reject the vector if the ratio is very close to a rational value with a small nominator and denominator. Figure 7 shows an example trajectory through a motion texture and its result.

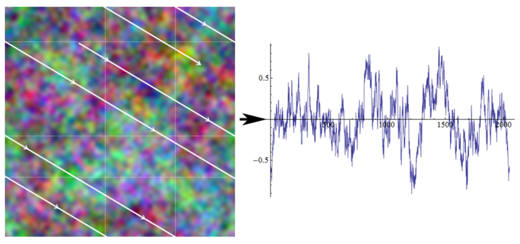

Figure 7: *Example wrapped trajectory through the motion texture (left) which results in a aperiodic signal with a defined spectrum (right).*

Usage for branches. We exploit 2D motion textures in two different ways: First, all trajectories through the texture are aperiodic, thus the tree animation will not show annoying periodicity artifacts. Second, we reduce the memory requirements for branches. Instead of generating a separate motion function for each branch, we use just one 2D motion texture for each hierarchy level and generate a trajectory for each branch in this level from a unique random vector. The physical properties that enter the computation of a motion texture are averaged from all branches at the corresponding hierarchy level. While this approach ignores differences in some physical attributes of branches at the same hierarchy level, this is a valid approximation since these branches have roughly the same properties and attributes, such as number of sub-branches and leaves attached.

Branch frequencies. However, there is one way to reintroduce variation. Instead of following the trajectory with the exact speed determined by the f_h which was used to create the texture, we vary f_h individually by varying the length of the motion vector $\vec{m}$ for each branch. This effectively scales the spectrum of $V_h(f)$ in the time domain. In order to determine an appropriate f_h for each branch, we use empirical data from [Cod00], which incorporates the dragging forces and internal damping of a branch, giving:

$$f_h = 2.55L^{-0.59} \tag{17}$$

Thus each branch has its own resonant frequency according to its length. Since branches in one hierarchy level have roughly similar lengths, the rescaling is small and the difference to a motion texture calculated using f_h in the first place is imperceptible. Note that this empirical result is based on broad leaf trees and needs to be modified for leafless branches or fir tree branches. Leafless branches have approximately 2.5 times the frequency calculated by equation (17).

Signal smoothing. The inverse Fourier Transform generates textures where the highest frequency is 2 texels per cycle. A good reconstruction of such a signal would require an appropriate reconstruction filter (e.g., sinc). However, graphics hardware provides only linear filtering, which would lead to unpleasant motion artifacts. Therefore, we *prefilter* the motion texture. Since we start with a frequency space representation, this can easily be done by applying a box filter to the spectrum. In practice, we start with the desired frequency range and extend the frequency domain by a factor of 4, padding the additional frequencies with 0. The resulting motion texture has a highest frequency of 8 texels per cycle, giving a much smoother result after linear interpolation. Since it is the frequency range around the resonance frequency which needs to be represented best, we set $f_{max} = 2f_h$ as the maximum frequency represented in the box region. The lowest representable frequency is thus $f_{min} = f_{max}/(8res)$, where *res* is the texture resolution.

Damping. Damping is one of the critical parameters in determining branch motion. Figure 8 shows the spectra and parts of the resulting texture in the normally damped (low γ_f) and overdamped (high γ_f) case. In the former case, the resonance frequency is dominant and will show up in the time domain accordingly, whereas the latter case hides the resonance frequency. Note that the overdamped case has a spectrum very close to a $1/f^{\beta}$ with $\beta \approx 2$, which explains why $1/f^{\beta}$ noise can be used to approximate the motion in some cases (as in [SFT*03]). However, this example demonstrates that it is important to take branch physics into account when animating trees. In practice, it is hard to estimate the damping coefficient γ_h of branches, as it depends on parameters such as the leaf mass, leaf distribution as well as distribution and viscoelastic damping of wood. Furthermore, trees targeted at real-time graphics are modeled after their appearance and not according to the correct dynamics of a tree, which leads to unsatisfactory results if γ_h is derived from the geometry. A comparison of empirical measurements [MM04] suggests values of γ_h in the slightly underdamped region, whereas large branches with a large number of subbranches and leaves are close to the critically damped case.

5.5. Wind Direction

As already mentioned, the wind direction is not essential for slight to medium winds, which are governed by turbulence. Stronger winds, however, will cause large branches that are orthogonal to the wind direction $\vec{W}$ to receive a directional force that causes strong bending. To model that behavior, for each branch the turbulent wind amplitude is offset by the force of the strong wind component according to the orientation of the branch, given by its local frame:

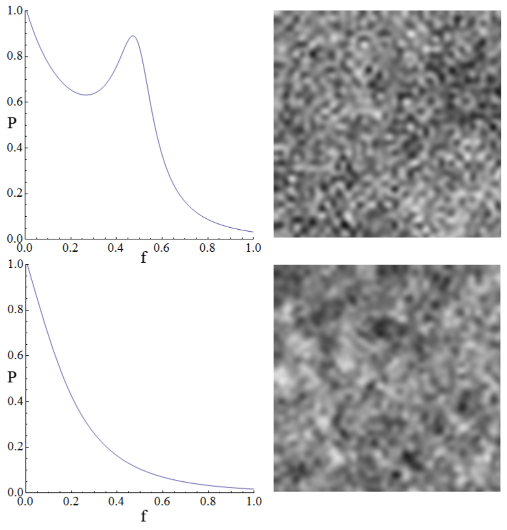

Figure 8: *Spectra and parts of the resulting textures for the damped (top) and the overdamped case (bottom).*

$$A'_r = A_r + (\vec{r}\cdot\vec{W}) \quad (18)$$
$$A'_s = A_s + (\vec{s}\cdot\vec{W}) \quad (19)$$

We also very slowly modulate the wind strength (length of $\vec{W}$) to increase the realism of the animation.

6. Applying Beam Deformation and Branch Motion

Since we need to carry out several deformations for each vertex, we store vertices in object coordinates and express the deformation, i.e., the offset, tangent and normal transformations shown in Section 4.3 in terms of the branch coordinate frame vectors, thus avoiding a full transformation into the local space. The positional deformation of one level simplifies to

$$\vec{p_D} = \vec{p_o} - \frac{\vec{t}d_r(x) - \vec{r}u_r(x)}{s_r(x)} - \frac{\vec{t}d_s(x) - \vec{s}u_s(x)}{s_s(x)} \quad (20)$$

with the length corrected x values

$$x_{D,r,s} = x - \frac{d_{r,s}(x)}{s_{r,s}(x)} \quad (21)$$

the tangent and normal transformation results to

$$\vec{t_D} = \vec{t_o} + (u'_r(x_{D,r})\vec{r} + u'_s(x_{D,s})\vec{s})(\vec{t}\cdot\vec{t_o}) \quad (22)$$
$$\vec{n_D} = \vec{n_o} - (u'_r(x_{D,r})(\vec{r}\cdot\vec{n_o}) + u'_s(x_{D,s})(\vec{s}\cdot\vec{n_o}))\vec{t} \quad (23)$$

The deformation is executed down the tree hierarchy, starting from the trunk until the level of the vertex is reached (i.e., as long as $x = \vec{w_i} \neq 0$). For the first iteration, $\vec{p_o}, \vec{t_o}, \vec{n_o}$ are the original position, tangent and normal in object space, for all further iterations they are set to $\vec{p_D}, \vec{t_D}, \vec{n_D}$ from the previous iteration. Analogue to the tangents, all involved child

Per hierarchy level	motion texture
Per branch	$c_2, c_4, L, \vec{t}, \vec{r}, \vec{s}, \vec{m}$ motion tex. index
Per vertex	$\vec{w}$, branch index

Table 2: *Data required to deform a vertex and associated normal and tangent. The per branch data is accessed through the branch index and the motion textures are accessed through the motion texture index.*

branch local frames under the current hierarchy level need to be transformed according to equation 22. Normals and tangents need to be normalized only after all deformations have been carried out. As shown in 3, the vertex needs access to the branch parameters of all its parent branches to execute these transformations. In particular, the terms dependent on x (e.g., $u_{r,s}(x)$) are simple polynomials in x with parameters c_2, c_4 (equation 7). Denormalization also requires the length L, and motion is accounted for by looking up $A_{r,s}$ from a motion texture (equation (11)). We store the branch parameters of the whole tree in a branch data texture, which the vertex can access based on its branch index, treating the texture as a data array accessible in the vertex shader. Each column of this floating point texture contains all the data of a branch, followed by the data of all parent branches. Analogue to the branch index, every branch has a motion texture index and motion vector $\vec{m}$ for the texture lookups to obtain $A_{r,s}$ (used in the evaluation of $u_{r,s}$). Using a texture-based data representation has the advantage that low-level per-branch parameters such as α or L can be edited in real time by simply modifying the corresponding texels. Table 2 summarizes which variables are represented on which level.

7. Leaves

Leaves need to be treated slightly different from branches because they do not need complex deformation and their motion behavior is different. But we can apply modified versions of the shown set of methods to ensure full consistency of the branch and leaf animation.

7.1. Leaf Deformation

We represent leaves as flat quads (although any geometric representation could be used). Leaves are treated as part of the last branch hierarchy and inherit all deformations of their parent branches, so a leaf vertex has the same data as the branch it is attached to. Apart from the inherited branch deformations, an additional animation is executed to perform the fluttering of leaves in the wind, modeled with a simplified version of the branch deformation, but with additional torsional motion.

For the local coordinate system, we make use of the tangent space $(\vec{t}_t, \vec{b}_t, \vec{n}_t)$. The UV coordinates (denoted by u_t, v_t to avoid confusion) of the vertices serve as local coordinates in this space, using a corrected coordinate $u_t' = (1-u_t)/2$ along the $\vec{t}_t$ axis, assuming a centered stipe of the leaf (see Figure 9). The branch deformer from Section 4.3 is applied twice: for translational and for torsional flutter. For the translational flutter, the local coordinate axes $(\vec{t}, \vec{r}, \vec{s})$ are set to $(\vec{b}_t, \vec{n}_t, \vec{t}_t)$, so that v_t is used for x. For the torsional flutter, $(\vec{t}, \vec{r}, \vec{s})$ are set to $(\vec{t}_t, \vec{b}_t, \vec{n}_t)$, with the corrected u_t' used for x. Note that the opposing signs of u_t' ensure the counter movement of opposing vertices.

In both cases, we execute equation (20) and (23) to carry out the deformation. However, as opposed to branches, we do not need a non-linear deformation for leaves, so we use a simple linear function $u^l(x) = A^l x$ for all evaluations of $u(x)$, where A^l determines the strength of the deformation from motion texture lookups (see next subsection).

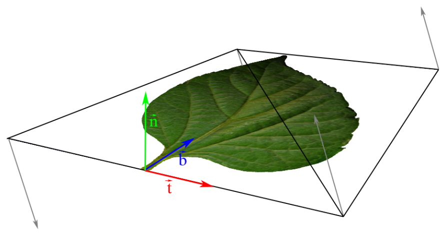

Figure 9: *Additional torsional deformation preceding the transversal deformation for the leaves.*

7.2. Leaf Animation

Since leaves are lightweight and small, we treat them as samplers of the turbulent wind field, and generate motion textures that use only the wind spectrum $P_w^l(f)$ as input (see Section 5.3). The spatial relation of leaves in gusts of wind causes leaves to behave in a similar way if they are close together, i.e., nearby leaves should have similar amplitudes and frequencies (but not necessarily directions). This could be accounted for by creating a 3D turbulence field for each flutter direction and moving this field along the wind direction $\vec{W}$. However, 3D textures of sufficient resolution to transport high frequencies and to avoid periodicity would be too memory intensive.

Instead, we show a solution using three independent motion textures: Each leaf vertex is projected onto the three different planes $x, y, z = 0$ in object space and noise values $A^p_{xy,xz,yz}$ are fetched from the motion textures after offsetting the vertex position by $-\vec{W}t$ where t represents time. These 3 values are spatially correlated as desired. However their frequency spectra are scaled by the projection of the wind vector on the coordinate planes. We therefore blend the 3 values in a way that the textures whose frequency is best preserved receives the largest weight:

$$A^l = A^p_{xy}(1 - \frac{|W_z|}{|\vec{W}|}) + A^p_{yz}(1 - \frac{|W_x|}{|\vec{W}|}) + A^p_{xz}(1 - \frac{|W_y|}{|\vec{W}|}) \quad (24)$$

While linear trajectories in the resulting 3D turbulence field do not have the exact desired spectra, they are still consistent and spatially correlated. The same measures against periodicity should be taken for the wind vector as in Section 5.3. Figure 10 shows a visualization of the lookups for one leaf vertex. The three noise textures for the three axes of leaf deformation (Section 7.1) are stored in the 3 channels of an RGB texture. The texture resolution is chosen so

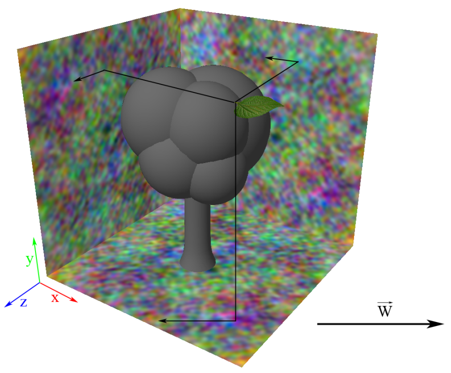

Figure 10: *3D turbulence field through averaged lookups of 3 motion textures.*

that the minimum wavelength represented in the leaf motion textures is 4 times the maximum leaf size. This avoids too high frequencies which could cause vertices of a single leaf to behave inconsistently. Nonetheless, each vertex of a leaf performs a slightly different movement, so the leaf itself does not stay flat. This adds complexity to the overall appearance by mimicking the complex behavior of leaves in wind compared to the rigid rotation of flat quads [SFT*03], which especially becomes apparent if a specular shading model is used for the leaves.

8. Implementation, Results and Discussion

For validation of the method, we have used trees of about 75k vertices, resulting in around 1,500 branches and about 10,000 leaves, divided into 4 hierarchy levels. As input for our method, we require only the geometric description of a tree. In addition, we need the full branch hierarchy including local coordinate frames and beam radii. We have implemented a simple conversion routine to create full hierarchy data just from the geometric description. In this way, we can for example animate Xfrog [DL04] trees without manual intervention. For trees with continuous geometry, one simply needs to determine $\vec{w}$ through the relative position to the local branch coordinate system.

The implementation and performance measurements were done using DirectX 10 and a NVIDIA 8800 GTS graphics card with 512 MB RAM on a Pentium 4 (3.2 GHz). Since it is difficult to measure times separately due to the unified

	static(fps)	animated(fps)	time(ms)
unshaded	299	290	0.104
shaded	56	48	1.49
simplified	56	52	0.68
4 trees	49	32	5.4

Table 3: *Framerate comparison and animation-only time in the unshaded, shaded and full simplification and multiple trees case.*

shader architecture and interleaving of texture lookups and ALU calculations of current graphics hardware, we compare to rendering the non-animated tree, both shaded and unshaded. The shaded scene is rendered with a full HDR pipeline and dynamic filtered shadow maps and advanced shading algorithms, thus using the same resources as a modern computer game.

For the shaded and simplified case, the animation is executed two times due to the shadow mapping pass which is corrected for in the frame rate comparison in Table 3. In the shaded case, the animation time is longer since fewer shading units are available due to load balancing.

It can be seen that the cost of animation (last column) of a full geometry tree is negligible compared to shading the tree, allowing the animation of several highly detailed trees with a cost that scales linearly with the number of trees. We also compare the quality of the animation directly to real-life footage of trees in wind in the accompanying video in order to show that the characteristics of tree motion can be captured well. The video also contains a comparison of a tree under different wind conditions. The proposed technique can be adapted to the animation and shading requirements of a scene. To further marginalize the performance impact, the length correction described in Section (4.2) can be omitted for small deflections in light wind. Additionally, by calculating only the positional animation and shading the tree in its undeformed state, the performance impact can be minimized (see Table 3) without loosing the overall appearance of the animation. A direct comparison of the full and simplified method for light wind is also included in the video. Though we derive values for all parameters from physical properties, these parameters can be tuned to match different needs or artists' visions, which we assume will be the common use case. A screenshot of our tuning setup can be seen in Figure 11. Every single parameter can be set or overridden in real time on a per-level, per-branch and per-vertex basis since previous states of the animation are never accessed.

9. Conclusion and Future Work

We proposed an efficient technique to achieve high-quality animation and deformation of trees using a novel physically based deformation method and physically guided stochastic approach to animate a massive amount of branches and

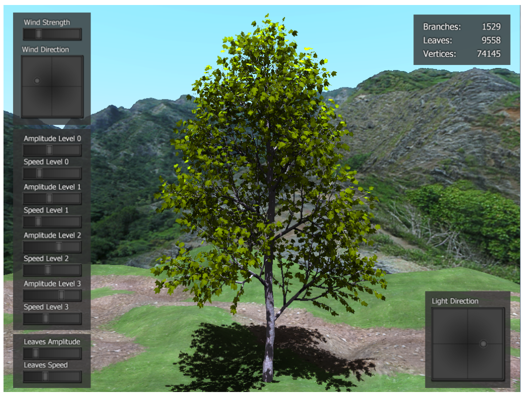

Figure 11: *Prototypical user interface to tune animation parameters.*

leaves at no considerable cost. The presented methods are confined to a vertex shader using hierarchical vertex displacement, leveraging the performance of GPUs and also making it easy to integrate into existing frameworks. The main novel contributions of this work are: a deformation model for tree branches that does not assume uniform beams but correctly accounts for taper; through the analytic expression of the deformation model, normals and tangents can be transformed efficiently to allow advanced shading on the deformed tree; a technique to synthesize an arbitrary number of aperiodic noise functions with a defined power spectrum from a simple 2D texture, thus allowing memory reduction and performance increase for real-time applications.

In summary, the appeal of the method lies in its simplicity and efficiency while still achieving high-quality animations: there is no precomputation required, so all animation parameters can be changed interactively and the method is self contained. A large class of tree models can be used directly, making the method immediately accessible. At the same time, the method also improves on the physical plausibility of previous methods by using a more accurate beam model. In the future, we would like to apply the shown methods to other tree representations such as billboard clouds to create consistent animation levels of detail for highly detailed forest scenes.

Acknowledgements

This research was funded by the Austrian Science Fund (FWF) under contract no. P17261-N04.

Appendix A

$$\begin{aligned} u(x) &= \frac{E'F}{r_1^4}\big(x(\alpha-1)(6+x(\alpha-1)(2x(\alpha-1)(3+(\alpha-3)\alpha) \\ &+ 3(4+(\alpha-2)\alpha))) - 6(1+x(\alpha-1))^2\log(1+x(\alpha-1))\big) \\ &\cdot \left(3\pi(1+x(\alpha-1))^2(\alpha-1)^4\right)^{-1} \end{aligned} \tag{25}$$

References

[AK06] AKAGI Y., KITAJIMA K.: Computer animation of swaying trees based on physical simulation. *Computers and Graphics 30*, 4 (2006), 529–539.

[BD45] BISSHOPP K. E., DRUCKER D. C.: Large deflection of cantilever beams. *Quarterly of applied Math 3*, 3 (1945), 272–275.

[CGZ*05] CHUANG Y.-Y., GOLDMAN D. B., ZHENG K. C., CURLESS B., SALESIN D. H., SZELISKI R.: Animating pictures with stochastic motion textures. *ACM Trans. Graph. 24*, 3 (2005), 853–860.

[Cod00] CODER K. D.: Sway frequency in tree stems. *University Outreach Publication FOR00-24* (2000).

[DL04] DEUSSEN O., LINTERMANN B.: *Digital Design of Nature: Computer Generated Plants and Organics*. SpringerVerlag, 2004.

[GCF01] GIACOMO T. D., CAPO S., FAURE F.: An interactive forest. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)* (sept. 2001), Springer, pp. 65–74. Manchester.

[MM04] MOORE J. R., MAGUIRE D. A.: Natural sway frequencies and damping ratios of trees: concepts, review and synthesis of previous studies. *Trees 3*, 18 (2004), 195Ű203.

[SF92] SHINYA M., FOURNIER A.: Stochastic motion-motion under the influence of wind. *Comput. Graph. Forum 11*, 3 (1992), 119–128.

[SFT*03] SHIN O., FUJIMOTO T., TAMURA M., MURAOKA K., FUJITA K., CHIBA N.: 1/fβ noise-based real-time animation of trees swaying in wind fields. In *Computer Graphics International* (2003), pp. 52–59.

[SO99] SAKAGUCHI T., OHYA J.: Modeling and animation of botanical trees for interactive virtual environments. In *VRST '99* (New York, NY, USA, 1999), ACM, pp. 139–146.

[Sou07] SOUSA T.: Vegetation procedural animation and shading in crysis. In *GPU Gems 3*, Nguyen H., (Ed.). Addison Wesley, 2007, ch. 16.

[SS86] SIMIU E., SCANLAN R.: *Wind Effects on Structures*. JohnWiley and Sons, 1986.

[Sta97] STAM J.: Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Computer Graphics Forum 16*, 3 (1997), C159–C164.

[TYW74] TIMOSHENKO S., YOUNG D., WEAWER, WILLIAMS, JR.: *Vibration problems in engineering*. New-York : 1974, 1974.

[WS05] WESSLÉN D., SEIPEL S.: Real-time visualization of animated trees. *The Visual Computer 21*, 6 (2005), 397–405.

[WVHR06] WILLIAM VAN HAEVRE F. D. F., REETH F. V.: Physically-based driven tree animations. *Eurographics Workshop on Natural Phenomena* (2006), 75–82.

[Zio07] ZIOMA R.: Gpu-generated procedural wind animations for trees. In *GPU Gems 3*, Nguyen H., (Ed.). Addison Wesley, 2007, ch. 6.

[ZST*06] ZHANG L., SONG C., TAN Q., CHEN W., PENG Q.: Quasi-physical simulation of large-scale dynamic forest scenes. In *Computer Graphics International* (2006), pp. 735–742.