Real-time Rendering of Dynamic Vegetation

Alexander Kusternig*

Institute of Computer Graphics
Vienna University of Technology
Austria

Abstract

This paper presents a new approach to shading realistic leaves at high frame rates. The proposed physically based translucency model allows the accurate simulation of subsurface scattering in leaves while accounting for the leaf micro-structure. This approach is based in the local frame of the leaf, so it can be instantiated to any number of leaves without needing to generate new data. Besides rendering leaves in a realistic way, a method is proposed to animate trees in wind in a physically based and efficient manner. This method allows the bending of branches as well as the movement of individual leaves according to the wind. A small set of parameters allows an artist to tweak the appearance of the animation to her liking in real-time.

Keywords: Real-time Rendering, Natural Phenomena, Natural Scene Rendering, Physically Based Rendering, Tree Animation, Brown Noise, Natural Motion

Figure 1: Translucent leaves rendered with our method.

1 Introduction

Natural scene rendering is an important research topic in real-time rendering. As research is constantly pushed forward by new generations of graphics hardware, new methods are developed to increase the realism of the generated images. There are three main problems when rendering vegetation: The first problem is the realistic appearance of the leaves of vegetation. The second one is animating the geometry in a realistic way on a frame by frame basis by constantly changing wind influence. The third problem is the one of geometric complexity needed to represent a tree or bush. These problems are discussed in detail in the following paragraphs.

Leaves are not simply flat green surfaces. Instead they consist of multiple layers of different tissues, and their surface properties vary depending on the plant type. In general, the front side of a leaf has high specularity whereas the back side is mostly diffuse in the respective reflective behaviour. In addition to the reflective properties of a leaf, translucency plays a important role in the appearance of a leaf. This translucency is a result of light transmitted through the thin layers of the leaf and subsurface scattering. Most state of the art leaf rendering techniques use some simplified forms of the 8D bidirectional subsurface scattering reflectance distribution function (BSSRDF) that would be needed to fully describe the appearance of a translucent leaf fully. Due to the highly scattering media in a leaf the viewer-angle-dependencies can be neglected when dealing with the translucency properties of a leaf, so simplifications are possible. However, preprocessing lighting information for a whole plant is a very CPU-intensive process, and even during rendering large amounts of illumination data have to be present to evaluate the lighting correctly. These methods are discussed in section 2.

This paper uses precomputed lighting information only for the translucency part of the lighting, while rendering the direct illumination with advanced per pixel lighting techniques to handle the complex specular behavior. Also, the translucency information is precomputed on a per leaf basis, so the data can be re-instantiated for all leaves on a tree, saving massive amounts of data. This could be achieved by employing the “Half Life 2” basis which allows calculating precomputed radiance transfer in local leaf space. Details on our method are found in section 3.

Static vegetation rendering poses a lot of problems on its own, but believable animation is another very complex topic. Most techniques available today use physically based models or procedural motion generation for the branches of a tree. This tends to by very CPU heavy and permits only a small number of branches. Also, these

*kchustl@cg.tuwien.ac.at

branches stay inflexible during animation, which does not compare to reality, where small twigs tend to bend in the wind. New interesting approaches come from computer games which need to render large numbers of trees at high frame rates but neglect the physical background on generating the motion data. These methods are explained in section 4.

This paper presents a new method that combines these different approaches by precomputing hierarchy information for a tree, but allowing flexible and fast animation that is entirely handled by the GPU. It includes a handful of animation parameters that can be tweaked by an artist to her liking to allow any animation behavior intended. Section 5 displays our animation method in detail.

The geometric complexity of vegetation poses another problem on efficient rendering, but this is not in the scope of this paper because we assume a tree to be rendered at maximum detail. Nonetheless, the methods proposed by this paper can be easily combined with existing level of detail methods.

Section 6 shows the performance that was achieved in our DirectX 10 implementation and possible future extansions of our method.

2 Related work on leaf rendering

2.1 Precomputed radiance transfer

Precomputed radiance transfer (PRT) works by calculating coefficients for the exiting radiance of surfaces at different conditions of incoming light in the hemisphere above the surface. These solutions do not operate in euclidian space but in other basii such as Spherical Harmonics (SH) [19] or Wavelets (which were introduced to avoid overshooting at shadow boundaries). The big advantage of PRT is that once these coefficients are computed (which is a very time-consuming CPU-intensive process), the lighting conditions in the scene can change in real-time, and direct and indirect lighting will adapt accordingly. The final evaluation process of the lighting is a very simple task that consists of multiplying the coefficients on the surface with the ones from the light source, and adding them up. However, the geometry has to stay static in the scene, because shadows are "baked" into the coefficients.

The precision of the solution depends on the number of coefficients used. This is especially important for direct lighting which introduces high frequency illumination changes due to shadows. On the other hand, changes in indirect lighting are of low frequency and could be approximated with only a few coefficients.

Normally, the color values resulting from the PRT evaluation on a per vertex or per texel basis are interpolated linearly, which is a problem with surfaces that have non-uniform reflectance attributes or a noticeable microstructure, which is the case with leaves. Sloan et al. [18] propose a method to combine PRT with normal mapping in order to retain the detailed micro-structure on surfaces.

2.2 Subsurface scattering

Full subsurface scattering would require the evaluation of an 8D BSSRDF (bidirectional subsurface scattering reflectance distribution function) [12] to handle self shadowing, changes in reflectance, and thickness. Instead, existing methods rely on the 4D BTDF (bidirectional transmission distribution function). Wang et al. [22] use a 4D BTDF to calculate the light transmission in plants. They produce impressive results, but also state that over 60 MB are needed to store the coefficients of a plant with about 15 leaves. This makes the method not feasible for larger trees.

2.3 Ray casting

A technique proposed by Govaerts et al. [8] represents the internal structure of leaf tissue as 3D geometry data, and performs ray casting to calculate subsurface scattering and translucency information. This produces accurate results for light propagation inside leaves, but obviously it is not possible to evaluate the model in real-time.

3 Rendering realistic leaves

3.1 Overview

We photographed real leaves to get real-world reflectance properties for diffuse lighting as well as translucent lighting coming from the backside. Afterwards we took 3D scans from both sides of the leaves to gain thickness information and generate a normal map from the 3D model.

Our leaf rendering model is split into direct and indirect illumination. The direct illumination is rendered using normal mapping with shadow mapping for shadows from direct illumination and a Cook Torrance specular term for the upper side of the leaves to simulate the very broad specular reflection properties of the leaf. The indirect illumination is handled by precomputed ambient occlusion data that takes into account the translucency properties of the leaves.

The translucency is also evaluated per pixel and handled by a new solution based on precomputed radiance transfer that takes into account subsurface scattering and local thickness variations along the leaf surface. The method is based on the "Half Life 2" basis that is easy to evaluate and also works in local leaf space, allowing instancing of the coefficients on all leaves in the scene. This produces a very detailed and accurate translucency appearance in combination with the normal map.

3.2 Acquiring the data

Our rendering technique requires four textures for each side of the leaf: A surface albedo map to modulate the

Figure 2: From left to right: albedo map, normal map, HL2 coefficients map, translucency color map

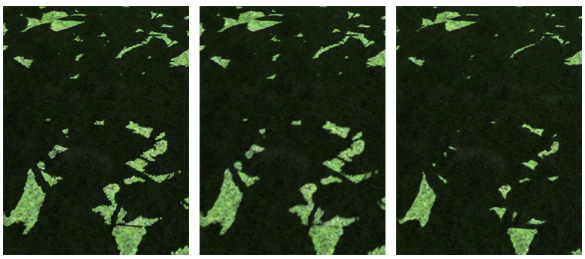

Figure 3: (a) PCF filtering (b) 6 PCF Poisson Taps (c) VSM

diffuse incoming light, a normal map to capture the fine details on the leaf surface, a translucency map to modulate the light transmitted through the leaf, and a "Half Life 2" coefficients map to calculate the amount of light transmitted. All textures are at a resolution of 1024*1024. Figure 2 shows one set of four textures for the front side of a leaf.

3.2.1 Photographing leaves

Photographs of real leaves were taken under controlled lighting conditions. The leaves were placed in a fixing frame, and lit by a 1000W large box diffuser once from the front side to get the surface albedo, and once from the back side to get the translucent color. This process was done for both sides of the leaf. A Canon EOS 20D digital camera was used to take the photographs. Specular highlights were removed from the albedo map with standard image processing tools.

3.2.2 3D scanning of the leaf geometry

The leaf geometry was scanned using a Minolta V1-910 scanner. Again, both sides of the leaf were scanned. The resulting 3D model was processed with GeoMagic and Maya to remove 3D scanning errors and get a watertight model of the leaf. A thickness map was computed from the model, and a normal map was generated directly from the geometry.

3.3 Direct lighting

The direct lighting consists of an ambient, a diffuse, and a specular term. The ambient shading is done by using the per vertex ambient occlusion color. The diffuse shading part is done by a per pixel normal mapping shader. The specular term is handled by a modified Cook Torrance reflectance model [4]. Roughness and refraction indices are taken from data measured from real leaves by Bousquet [3] in 2005. This model creates a broad specular reflectance that matches the rough surface of real-world leaves more accurately. Also, the Fresnel term in this reflectance model leads to higher specular intensity at grazing angles.

3.3.1 Shadows

Shadows are handled by directional shadow mapping. The shadow map is focused on the tree with a resolution of 4096*4096 pixels, and a bit depth of 32bit. The 32bit floating point precision is used to perform percentage closer filtering (PCF) [16] on the graphics hardware directly. Multiple shadow mapping techniques have been evaluated for their performance and appearance. The best results were achieved with using a sum of 6 Poisson-filtered PCF lookups. These 6 lookups do not noticeably reduce the shader performance when compared to one single lookup. This is due to the lookups being performed in parallel to the complex arithmetic instructions in the pixel shader. However, using six lookups instead of only one produces much softer shadows and reduces the "flickering" of texels that would appear when the leaves move in the wind.

Variance shadow mapping (VSM) [5] was implemented, but resulted in shadows that appeared too hard. Besides, blurring of the depth values of the shadow map is very costly, and also produces shadows that are larger than the objects casting them. Usually this is no problem, but with lots of very small objects in the shadow map that only fill a few texels each, the resulting shadows "bleed" into one another, and produce a much denser shadow area on the ground. This can be seen in the comparison figure 3. The Poisson-filtered PCF lookups produced softer and fuzzier shadows, which look more natural on the rough surfaces of leaves, tree bark, and the ground.

3.4 Indirect lighting

Indirect lighting is precomputed on a per vertex basis. However, this poses no problem in animation, as the changes of indirect illumination are of very low frequency, and the overall form of a tree top stays the same even under strong wind influence. Ambient occlusion (AO) is used to calculate the indirect illumination. The AO solver takes into account the translucency attributes of the leaves, so the AO colors become more greenish if the light hits the bark after traveling through multiple layers of leaves.

As can be seen in Figure 4, the indirect illumination greatly increases the realistic look of the generated images and also gives the viewer a cue about size and depth of the treetop.

Figure 4: The indirect lighting term only. Note how the leaves and the bark get gradually darker inside the tree top.

3.5 Translucency and the "Half Life 2" basis

Normal mapping is used to render the ambient, diffuse, and specular properties of the leaves, but a PRT solution is used to handle the calculation of the light transmitting the leaf slab to model the translucency on the back sides of leaves. We use the recently developed "Half Life 2" (HL2) basis [10] introduced by the Source Engine in 2004. Traditional PRT methods work on Spherical Harmonics bases or Wavelets which need costly transformation of the light into coefficients for these bases. The HL2 basis on the other hand allows us to work in local leaf space, because it is defined by only three orthogonal basis vectors. That allows us to instantiate these HL2 coefficients for all leaves in the scene, as opposed to SH or Wavelet simulations.

One drawback of the HL2 basis is that it uses only three basis vectors, so only very low frequent lighting is possible. Fortunately this is no problem because we only store information for the light transmission in the HL2 coefficients and no information about direct illumination. The low frequency light changes even support the subsurface scattering effect.

The three HL2 basis vectors are orthogonal and distributed uniformly across the hemisphere above the surface. During rendering the weighting of the coefficients is evaluated by a simple dot product between the incident light vector and these basis vectors. This means that also negative weights are possible (as with SH). Simply adding up the results from these three dot products gives the overall intensity of transmitted light for a given light vector. By multiplying this light intensity by the translucency color map, we get the final translucency color. Additionally, this intensity can be modulated by the shadow term of direct illumination.

3.5.1 Calculating the HL2 coefficients

The calculation of the HL2 coefficients relies on a technique to simulate subsurface scattering for thin slabs by Donner et al. [6]. The thickness map gained from the 3D scanning is used to weigh the amount of light transmitted through the multiple layers of the leaf slab. The transmission properties on the other hand are assumed to be the same over the whole surface of the leaf. Details on the calculation of the coefficients and error measurement can be found in Habel 07 [9]. We found that the error is only 3% between our solution and Donner's method. Applying a full PRT solution would require the calculation of three HL2 coefficient maps for each color channel. We calculate the coefficients for the dominant wavelength of green light (510nm) only. Also, the view dependency usually found in BRDFs is dropped in our solution because the translucency properties of leaves are practically diffuse [2]. This results in only one 3-component texture to store the three HL2 coefficients.

Figure 5: The translucency term at different incident light angles: f.l.t.r.: Light coming directly from behind the leaf, and moving to the right.

3.5.2 Efficiency

We can instantiate the same HL2 coefficients map for all leaves in the scene because our approach works in local leaf space. This is a great advantage over SH or Wavelet-based methods that would need unique data for every leaf. Reusing the same data set for all leaves allows us to use a 1024*1024 HL2 coefficients map, which captures all the small details on a leaf surface like self shadowing and bulges. Figure 5 shows how the translucency changes at grazing angles, revealing the detailed structure of the leaf. Looking up the amount of light transmitted through the leaf slab only takes three dot products between the light vector and the basis vectors, and adding up the results of these dot products, which makes for very fast and efficient shader code.

4 Related work on tree animation

4.1 Scientific models

Most of the existing methods for animating trees work by separating the model into individual branches and creating a skeleton with bones assigned to them. These bones are rotated depending on the wind influence [7]. Rotation information normally comes from physically based or

stochastic models. This is mostly done on the CPU and demands the propagation of the hierarchical transformations from the parent bones to their children. The performance goes down with an increasing number of branches. Wong et al. [23] report an average of about 30 frames per second for small plants with about 20 large leaves (whereby each leaf is attached to multiple bones). A big disadvantage of these models is that the branches cannot bend and stay inflexible during the animation. This does not match to reality, where small twigs are bent by wind.

Jos Stam [21] uses modal analysis to generate motion data. This produces very accurate results, but also requires a large processing time. This method cannot run at interactive frame rates for trees with a number of branches similar to the tree model we use in our implementation.

Ota et al. [13] try a hybrid method that combines stochastic motion from $1/f^{\beta}$ noise to move the individual leaves with a mass spring model to move the branches. The endpoints of the leaves are moved along their local frame, and the whole leaf is rotated around the petiole axis. They confirm that the motion of individual leaves is a very important factor in the believability of plant animation, a fact that is missing in the methods that rely on branch movement only.

4.2 Artist-controlled animation

In recent years computer games have presented new methods for vegetation animation. Their models are based on completely different priorities: The main issue with games is performance and believability, not physically based simulations. This results in methods that operate on the vertices directly instead of calculating branch hierarchies. One particularly impressive example is the game "Crysis" from the company CryTek [11]. Their wind animation system is entirely artist-controlled, with artists painting movement information for every single vertex on a plant by hand. This is tedious work and takes weeks per plant until perfected, but the results are impressive.

This type of animation is limited by the strength of the wind and the overall look of the plant: If the wind gets too strong, then the vertices start to move around wildly in a seemingly uncorrelated fashion. Also, this method only works for plants with one long stem, like palm trees. The model would not work correctly for weeping willows, for example. Details on the implementation of the Crysis plant animation can be found in Sousa's GPU Gems 3 article [20].

5 Efficient physically based animation

Our method combines the physically based hierarchical swaying motion of branches with a fast and efficient rendering method that is based on per vertex displacement only. We also ensured that the animation parameters are easily controllable by a user to meet the needs for all different types of trees. Our method calculates a hierarchy of branches moving in the wind in a preprocessing step, but then all the data is distributed locally to the vertices. The leaves are transformed together with this hierarchy and are additionally moved independently but in a correlated motion, depending on the wind.

5.1 Generating the animation data

Our weights generator takes an existing 3D tree model as input data. The model has to be split into one object for the trunk and all branches, and one object for the leaves. This model can be created by a standard tree generation package like XFrog or NatFX, no branch or bone information is demanded. The whole weights generation process consists of two passes: The weights for the trunk and branches are generated in the first pass. In the second pass, the leaf weights data is generated.

5.1.1 Building a branching hierarchy

The trunk model is separated into branches by identifying continuous segments of triangles. This is how generated tree models come from generators like XFrog and NatFX. Each segment of triangles is treated as a branch. They are ordered by their diameter, the branch with the largest diameter is assumed to be the stem. A hierarchy is generated by starting with the stem, and finding all branches intersecting with it. This is repeated until the whole hierarchy information for the tree is built.

5.1.2 Weights, phases, and motion vectors

Animation information for up to four levels of hierarchy are supported in the current implementation, since this is the maximum of components that one vertex attribute set can hold. Also, the tree used in our implementation only has four hierarchy levels which already sums up to about 1500 branches. Any branches that would lie beyond the fourth hierarchy level would generate no unique weights data, but simply take the one from their parent branches.

Our animation method relies on four different animation attributes. All of these attributes are stored for up to four levels of hierarchy: The first attribute is a set of weights that define the amplitude of the swaying motion and are unique for every vertex. The second attributes are the phases that define an offset value in the noise texture lookup and are unique per branch. They result in each branch swaying at its own time, but with the same frequency as the other branches of the same hierarchy level. The last two attributes are the up an right vector of the branch frame in model coordinates. They control the directions of the actual displacement of the vertices, and are per branch attributes like the phases, too. Applying values from the noise texture to control the swaying results in

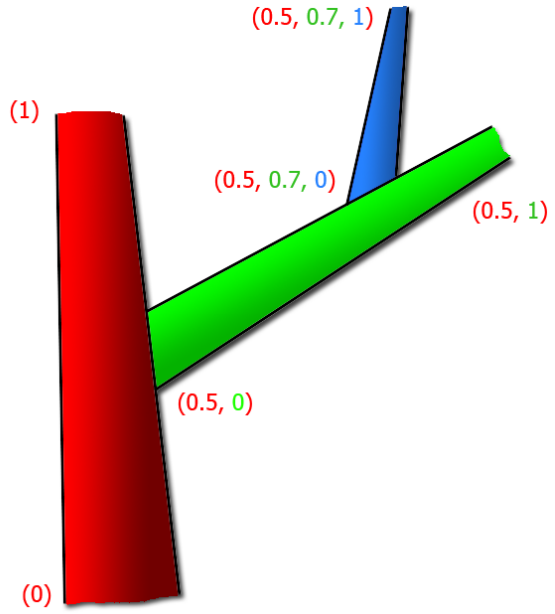

Figure 6: The weights are propagated from the top level to the children

two independent swaying motions along the X and Y axis of the branch

The weights are stored directly per vertex into the vertex buffer, while the other attributes are stored in a texture per branch and looked up in the vertex shader.

5.1.3 Assigning weights

Weights are generated for all vertices of a branch. They are distributed linearly along the branch, starting at 0 for the branch starting point. The weights are normalized for each hierarchy level, so the longest branch of each hierarchy level receives a weight of 1 at its endpoint. This allows for weights data in the range of 0 to 1 for all hierarchies, which allows efficient tweaking for the final look of the animation, but also ensures that the individual branch lengths have an influence on the bending behavior.

Since the weights are generated for the branches in order of their hierarchy level there is always weight data available from the parent branch. The weights data from the parent branch is simply propagated to the current branch. This ensures that the vertices on twigs of a sinle branch adopt the same animation behavior as the parent branch, since their motion computation relies on the same per vertex information as the vertices of the parent branch, plus their own animation information. Figure 6 displays how the weights are propagated from the parent branch to the children.

5.2 Animating the branches

The branches are animated in a fast and efficient way. The weights for a hierarchy of up to four levels is known per vertex, as are the phases that modulate the lookup into the noise texture and the branch movement vectors that finally displace the vertex. The noise texture is a tiling 2D $1/f^{\beta}$ noise, which is suitable to represent natural motion [14]. A β value of 2 is used to gain smooth swaying motion. Bending of branches and twigs is possible by modulating the weights to be applied in a non-linear way.

5.3 Animating the leaves

Leaves adopt the weights and branch index data from the trunk vertex nearest to the petolia. This results in the leaves moving correctly together with the branches that they are attached to.

Additionally, the endpoints of each leaf move around independently a little bit, controlled by "rolling" the 2D noise texture over the tree in wind direction, and taking the noise values for each leaf position as displacements. This leads to independent, but also correlated, motion for all leaves, as one can see gusts of wind moving through the tree. This technique is similar to the one proposed by Bakay et al in "Real-Time Animated Grass" [1]. Also, each leaf is rotated into the direction of the wind, resulting in all leaves bending in approximately the same direction, giving an additional visual cue to where the wind comes from.

6 Results

6.1 Implementation performance

The DirectX 10 implementation renders to a framebuffer of the size of 1024*768 pixels. The tree model used has about 70000 triangles for the trunk, and about 12000 leaves which are represented as alpha tested quads. The leaves textures need about 50 MB on the GPU. The whole scene is rendered into a floating point target for HDR rendering, and tone mapped by a modified Reinhard tone mapper [17]. The light shafts are computed in a separate pass and applied to the image before tone mapping. A simple 7*7 separable bloom is applied to the scene afterwards.

The main bottleneck of the rendering pipeline is the leaves pixel shader, which has 119 instructions according to the DirectX 10 FX compiler output. Interestingly, the computation of the translucency only takes about 15 instructions. The evaluation of the Cook Torrance specular model currently takes the biggest part of the shader together with the final combination of all the different lighting terms and weighting them to get a visually pleasing output. An early Z pass is used to reduce the pixel shader load and ensure that only the fragments of the leaves nearest to the camera have to be rendered with the full shader.

The sky is currently rendered using the Preetham Skylight model [15] and is evaluated per vertex for a sky dome. The sun position and color is updated every frame

on the CPU, and the sunlight color values also come from Preetham's algorithm.

The shadow map used in the application has a size of 4096*4096 pixels, and uses 32bit floating point precision. The 32bit precision allows using hardware PCF [16] in DX10. A 6 taps Poisson filter is used for lookups in the shadow map. Also, Variance Shadow Mapping [5] with a 5*5 blur kernel was applied to the scene, but was dropped later on because the shadows resulting from it appeared too hard and degraded the believability of the generated image. Besides, VSM have a noticeable impact on performance, because the shaders to evaluate the VSM lookup use almost only arithmetic instructions.

The wind animation poses a high workload on the vertex shader with 12 texture lookups needed per vertex to obtain the weights information necessary to transform every vertex. This has to be performed three times per frame in the current implementation: Once while rendering into the shadow map, once for the early Z pass and once for the bright pass. Using render-to-vertex-buffer capabilities should dramatically reduce this overhead to only once per frame, which is planned for future work.

The overall frame rate varies from 60 to 110 frames per second on a Intel Core 2 Duo at 2.67 GHz and a Geforce 8800 GTX. The actual frame rate depends on the size of the tree in the frame buffer due to the number of fragments rendered with the complex leaves pixel shader.

6.2 Future work

Future work on the rendering side could include applying a sophisticated level of detail blending algorithm to blend between the current high-polygon model where each leaf is shaded per pixel to a simpler per vertex lighting model, and finally down to simple billboard rendering. Using an advanced level of detail technique will allow us to render a large number of trees, which is important for believable natural scenes.

Future work regarding the wind animation includes applying a more sophisticated transformation algorithm to the leaves. At the moment, the endpoints of the leaves are displaced only in world coordinates. A possible extension is for the leaves to rotate around the petolia axis, which can be performed effectively in the shader since the leaf's local frame is already well defined by normal, tangent, and binormal. 3D noise could be used to model a full wind motion field.

7 Conclusion

We present a method to render highly detailed trees in a very realistic and efficient manner. The methods introduced in this paper can be applied to any existing rendering pipeline without much work because preprocessing is done offline and online rendering only involves updating the vertex and pixel shaders.

Our leaf rendering model is based on a combination of physically based methods to render specular reflectance and translucency for highly detailed leaves close to the camera. The detailed micro-structure on a leaf surface and inside the multiple layers of leaf tissue are simulated accurately by using normal maps in combination with the HL2 maps. The HL2 basis allows a very fast and efficient lookup at runtime, and using the same HL2 map for every leaf gives us the opportunity to instantiate the leaves and therefore render large numbers of them.

The wind animation is based on hierarchical motion of branches, the actual motion is based on $1/f^{\beta}$ noise, which is found in many natural phenomena. The fast per vertex evaluation permits us to transform a very large number of branches, and also bend these branches according to the wind influence. Every parameter in the animation system can be artist controlled and tweaked in real-time until it is visually pleasing, without the need to preprocess the hierarchy information multiple times.

References

[1] Brook Bakay and Wolfgang Heidrich. Real-time animated grass. In *Proceedings of Eurographics (short paper)*, 2002.

[2] G. Baranoski and J. Rokne. Efficiently simulating scattering of light by leaves. *The Visual Computer*, 17(8):491–505, 2001.

[3] L. Bousquet, S. Lacherade, S. Jacquemoud, and I. Moya. Leaf BRDF measurements and model for specular and diffuse components differentiation. *Remote Sensing of Environment*, 98:201–211, 2005.

[4] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Trans. Graph.*, 1(1):7–24, 1982.

[5] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 161–165, New York, NY, USA, 2006. ACM.

[6] Craig Donner and Henrik Wann Jensen. Light diffusion in multi-layered translucent materials. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1032–1039, New York, NY, USA, 2005. ACM Press.

[7] Thomas Di Giacomo, Stéphane Capo, and François Faure. An interactive forest. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, pages 65–74, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[8] Y. Govaerts, S. J. M. Verstraete, and S. Ustin. Three-dimensional radiation transfer modeling in a dycotyledon leaf. *Applied Optics*, 35(33):6585–6598, 1996.

Figure 7: A tree from the outside and from the inside. Notice the indirect illumination and translucency visible from the inside.

[9] Ralf Habel, Alexander Kusternig, and Michael Wimmer. Physically based real-time translucency for leaves. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 253–263. Eurographics, Eurographics Association, June 2007.

[10] G. McTaggert. Half-Life 2/Valve Source Shading. Technical report, Valve Corporation, 2004.

[11] Martin Mittring. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 97–121, New York, NY, USA, 2007. ACM.

[12] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. *Geometrical considerations and nomenclature for reflectance*. Jones and Bartlett Publishers, Inc., USA, 1977.

[13] Shin Ota, Machiko Tamura, Tadahiro Fujimoto, Kazunobu Muraoka, and Norishige Chiba. A hybrid method for real-time animation of trees swaying in wind fields. *The Visual Computer*, 20:613–623(11)], dec 2004".

[14] Heinz-Otto Peitgen and Dietmar Saupe, editors. *The Science of Fractal Images*. Springer-Verlag New York, Inc., New York, NY, USA, 1988.

[15] A. J. Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 91–100, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[16] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH '87 Proceedings*, pages 283–291, New York, NY, USA, 1987. ACM Press.

[17] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Trans. Graph.*, 21(3):267–276, 2002.

[18] Peter-Pike Sloan. Normal mapping for precomputed radiance transfer. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 23–26, New York, NY, USA, 2006. ACM Press.

[19] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02 Proceedings*, pages 527–536, New York, NY, USA, 2002. ACM Press.

[20] Tiago Sousa. Vegetation procedural animation and shading in crysis. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 16, pages 373–386. Addison Wesley, July 2007.

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

[22] Lifeng Wang, Wenle Wang, Julie Dorsey, Xu Yang, Baining Guo, and Heung-Yeung Shum. Real-time rendering of plant leaves. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 712–719, New York, NY, USA, 2005. ACM Press.

[23] Jason C. Wong and Amitava Datta. Animating real-time realistic movements in small plants. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 182–189, New York, NY, USA, 2004. ACM.