Third Hungarian Conference on Computer Graphics and Geometry, Budapest, 2005

Tree Rendering with Billboard Clouds

Ismael Garcia, Mateu Sbert, and László Szirmay-Kalos

University of Girona and Budapest University of Technology
Emails: igarcia@ima.udg.es, mateu@ima.udg.es, szirmay@iit.bme.hu

Abstract

This paper presents a method to render complex trees on high frame rates while providing accurate occlusion and parallax effects. Based on the recognition that a planar billboard is accurate if the represented polygon is in its plane, we find a billboard for each of those groups of tree leaves that lie approximately in the same plane. The tree is thus represented by a set of billboards, called billboard cloud. The billboards are built automatically by a clustering algorithm. Unlike classical billboards, the billboards of a billboard cloud are not rotated when the camera moves, thus the expected occlusion and parallax effects are provided. On the other hand, this approach allows the replacement of a large number of leaves by a single semi-transparent quadrilateral, which considerably improves the rendering performance. A billboard cloud well represents the tree from any direction and provides accurate depth values, thus the method is also good for shadow, obscurances, and indirect illumination computation. In order to provide high quality results even if the observer gets close to the tree, we also propose a novel approach to encode textures representing the billboards. This approach is called indirect texturing and generates very high resolution textures on the fly while requiring just moderate amount of texture memory.

1. Introduction

One of the major challenges in rendering of vegetation is that the large number of polygons needed to model a tree or a forest exceeds the limits posed by the current rendering hardware [3]. Rendering methods should therefore apply simplifications. To classify these simplification approaches, we can define specific scales of simulation at which rendering should provide the required level of realism:

- *Insect scale*: A consistent, realistic depiction of individual branches and leaves is expected. The images of individual leaves should exhibit parallax effects when the viewer moves, including the perspective shortening of the leaf shape and its texture, and view dependent occlusions.
- *Human scale*: Scenes must look realistic through distances ranging from an arm's reach to some tens of meters. Consistency is desired but not required.
- *Vehicle scale*: Individual trees are almost never focused upon and consistency is not required. Viewing distance may exceed several hundred meters.

There are two general approaches for realistic tree rendering: geometry-, and image-based methods. As it takes roughly hundred thousand triangles to build a convincing model of a single tree, geometry-based approaches should apply some form of Level of Detail technique [6].

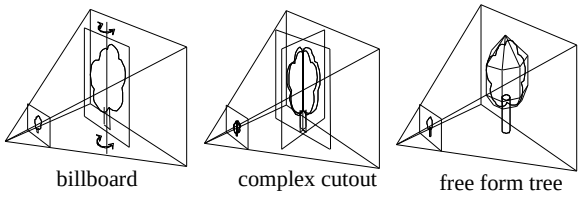

Figure 1: *A tree representation with images*

Image-based methods [12, 1] represent a trade-off of consistency and physical precision in favor of more photorealistic visuals (figure 1). *Billboard rendering* is analogous to using cardboard cutouts. In order to avoid the shortening of the visible image when the user looks at it from grazing angles, the billboard plane is always turned towards the camera. Although this simple trick solves the shortening problem, it is also responsible for the main drawback of the method. The tree always looks the same no matter from where we look at it. This missing parallax effect makes the replacement too

easy to recognize. The user expects that the size and texture of the leaves, as well as the distance and relative occlusions of leaf groups change as he moves. In order to handle this problem, the *view-dependent sprite* method pre-renders a finite set from a few views, and presents the one closest in alignment with the actual viewing direction. A popping artifact is visible when there is an alignment change. The *complex cutout* approach uses texture transparency and blending to render more than one views at the same time onto properly aligned surfaces. Both methods yield surprisingly acceptable results in vehicle scale simulations, but fail to deliver quality in close-up views. In order to handle occlusions correctly, billboards can also be augmented with per-pixel depth information [13, 9, 14]. If the tree is decomposed to parts and each part is represented by such a *nailboard* or *2.5 dimensional impostor*[15], the parts can be properly composited. On the other hand, rendering different parts with different billboards a limited parallax is also provided.

An advanced approach that has been actually implemented in commercial entertainment software is the basic *free-form textured tree model*, where the images are imposed on the approximated geometry. Resulting visuals are satisfactory, although due to the simple geometric model used, close-up views usually look artificial.

Finally, the *billboard cloud*[2] approach decomposes the original object into subsets of patches and replaces each subset by a billboard. These billboards are fixed and the final image is the composition of their images. Unlike nailboards, the billboards of a billboard cloud do not have per-pixel depth, but the depth is calculated from the plane of the billboard.

The method of this paper falls into the class of billboard clouds, but it prepares the billboards automatically and so carefully that the results are satisfactory not only on vehicle scale, but also on human scale, and with some compromises on insect scale as well. Since the billboard cloud model has been proposed to model conventional man-made 3D objects, such as teapots, helicopters, etc., we have to alter their construction to take into account the special geometry of natural phenomena. Our method is also good to generate shadows [11], to compute indirect illumination, and can be extended to a hierarchical, level of detail approach [8].

2. The new method

In order to reduce the geometric complexity of a tree, we use billboards having transparent textures to represent leaf groups. Since our billboards are not rotated when the camera changes, the expected parallax effects are provided, and the geometric accuracy is preserved.

The key of the method is then the generation of these billboards. We should first observe that a planar billboard is a perfectly accurate representation of a polygon (i.e. leaf) if the plane of billboard is identical to the plane of the polygon. On the other hand, even if the leaf is not on the billboard plane but is not far from that and is roughly parallel to the plane, then the billboard representation results in acceptable errors. Based on this recognition, we form clusters of the leaves in a way that all leaves belonging to a cluster lie approximately on the same billboard plane and replace the whole group by a single billboard. In order to control clustering, we shall introduce an error measure for a plane and a leaf, which includes both the distance of the leaf from the plane and the angle of the plane and leaf normals. The applied *K-means clustering algorithm* [7] starts with a predefined number of random planes, and iterates the following steps:

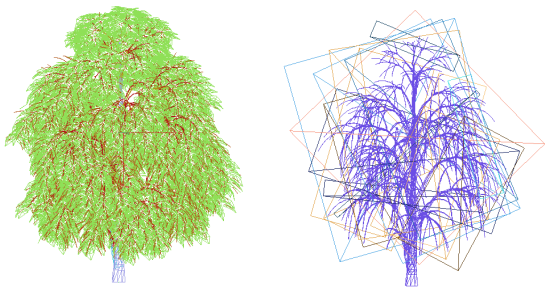

Figure 2: *The basic idea of the proposed method: the original tree defined by a hundred thousand polygons is approximated by a few fixed billboards, where a single billboard itself approximates many leaves.*

1. For each leaf we find that plane that minimizes the error. This step clusters the leaves into groups defined by the planes.
2. In each leaf cluster, the plane is recomputed in order to minimize the total error for the leaves of this cluster with respect to this plane.

Let us examine the definition of the error measure and these steps in details.

2.1. Error measure for a leaf and a plane

Those $\mathbf{p} = [x, y, z]^T$ points are on a plane that satisfy the following plane equation (T denotes matrix transpose):

$$D_{[\mathbf{n},d]}(\mathbf{p}) = \mathbf{p}^T \cdot \mathbf{n} + d = 0,$$

where $\mathbf{n} = [n_x, n_y, n_z]^T$ is the normal vector of the plane, and d is a distance parameter. We assume that $\mathbf{n}$ is a unit vector and is oriented such that d is non-negative. In this case $D_{[\mathbf{n},d]}(\mathbf{p})$ returns the signed distance of point $(\mathbf{p})$ from the plane.

From the point of view of clustering, leaf i is defined by its unit length average normal $\mathbf{n}_i$ and its center $\mathbf{p}_i$, that is by pair $(\mathbf{n}_i, \mathbf{p}_i)$.

polygonal tree | 107 billboards, 163 FPS | 54 billboards, 256 FPS | 29 billboards, 292 FPS

Figure 3: *Visual and speed comparisons of the original polygonal tree and the tree rendered with different number of billboards.*

Leaf i approximately lies in plane $[\mathbf{n},d]$ if $D^2_{[\mathbf{n},d]}(\mathbf{p}_i)$ is small, and its normal is approximately parallel with the normal of the plane, which is the case when $1-(\mathbf{n}_i^T\cdot\mathbf{n})^2$ is also small. Thus an appropriate error measure for plane $[\mathbf{n},d]$ and leaf $(\mathbf{n}_i,\mathbf{p}_i)$ is:

$$E([\mathbf{n},d]\leftrightarrow(\mathbf{n}_i,\mathbf{p}_i))=D^2_{[\mathbf{n},d]}(\mathbf{p}_i)+\alpha\cdot(1-(\mathbf{n}_i^T\cdot\mathbf{n})^2), \quad (1)$$

where α expresses the relative importance of the orientation similarity with respect to the distance between the leaf center and the plane.

2.2. Finding a good billboard plane for a leaf cluster

To find an optimal plane for a leaf group, we have to compute plane parameters $[\mathbf{n},d]$ in a way that they minimize total error $\sum_{i=1}^N E([\mathbf{n},d]\leftrightarrow(\mathbf{n}_i,\mathbf{p}_i))$ with respect to the constraint that the plane normal is a unit vector, i.e. $\mathbf{n}^T\cdot\mathbf{n}=1$. Using the Lagrange-multiplier method to satisfy the constraint, we have to compute the partial derivatives of

$$\sum_{i=1}^N E([\mathbf{n},d]\leftrightarrow(\mathbf{n}_i,\mathbf{p}_i))-\lambda\cdot(\mathbf{n}^T\cdot\mathbf{n}-1)=$$

$$\sum_{i=1}^N(\mathbf{p}_i^T\cdot\mathbf{n}+d)^2+\alpha\cdot\sum_{i=1}^N(1-(\mathbf{n}_i^T\cdot\mathbf{n})^2)-\lambda\cdot(\mathbf{n}^T\cdot\mathbf{n}-1) \quad (2)$$

according to n_x,n_y,n_z,d,λ, and have to make these derivatives equal to zero. Computing first the derivative according to d, we obtain:

$$d=-\mathbf{c}^T\cdot\mathbf{n}. \quad (3)$$

where

$$\mathbf{c}=\frac{1}{N}\cdot\sum_{i=1}^N\mathbf{p}_i$$

is the *centroid* of leaf points. Computing the derivatives according to n_x,n_y,n_z, and substituting formula 3 into the resulting equation, we get:

$$\mathbf{A}\cdot\mathbf{n}=\lambda\cdot\mathbf{n}, \quad (4)$$

where matrix $\mathbf{A}$ is

$$\mathbf{A}=\sum_{i=1}^N(\mathbf{p}_i-\mathbf{c})\cdot(\mathbf{p}_i-\mathbf{c})^T-\alpha\cdot\sum_{i=1}^N\mathbf{n}_i\cdot\mathbf{n}_i^T. \quad (5)$$

Note that the extremal normal vector is the eigenvector of symmetric matrix $\mathbf{A}$. In order to guarantee that this extremum is a minimum, we have to select that eigenvector which corresponds to the smallest eigenvalue. We could calculate the eigenvalues and eigenvectors, which requires the solution of the third order characteristic equation. On the other hand, we can take advantage that if $\mathbf{B}$ is a symmetric 3×3 matrix, then iteration $\mathbf{B}^n\cdot\mathbf{x}_0$ converges to a vector corresponding to the largest eigenvalue from an arbitrary, non-zero vector $\mathbf{x}_0$ [16]. Thus if we select $\mathbf{B}=\mathbf{A}^{-1}$, then the iteration will result in the eigenvector of the smallest eigenvalue.

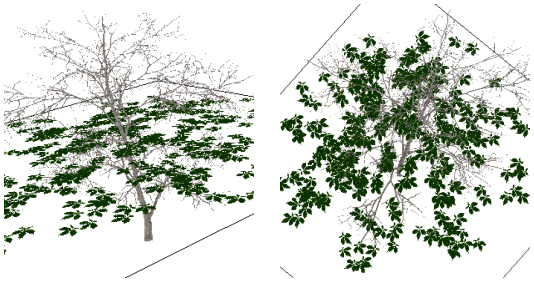

Figure 4: *Two from the 32 leaf clusters representing the tree*

2.3. Smooth level of detail

The accuracy of the impostor representation can be controlled by the number clusters. If clusters are organized hierarchically, then the accuracy can be dynamically set by specifying the used level of the hierarchy. The level is selected according to the viewing distance, which results in a level of detail approach. To make the changes smooth, we can interpolate the plane parameters of the two enclosing levels.

3. Indirect texturing

The proposed method uses a single texture for the impostor that includes many leaves. It means that the impostor texture should have high resolution in order to accurately represent the textures of individual leaves. Note that lower resolution textures lead to poor results even if sophisticated texture filtering is applied (figure 6).

The resolution of the impostor texture can be reduced without sampling artifacts if the impostor stores only the position and orientation of the leaves, while the leaves rotated at different angles are put in a separate texture (figure 5). Let us identify the rectangles of the projections of the leaves on the impostor. Each rectangle is defined by its lower left coordinate, the orientation angle of the leaf inside this rectangle, and a global scale parameter (due to clustering the impostor planes are roughly parallel to the leaves thus leaf sizes are approximately kept constant by the projections). If we put the lower left coordinate of the enclosing rectangle and the orientation angle into the impostor texture, and fill up the texels that are not covered by leaves by a constant value outside [0,1], then this texture has constant color rectangles, which can be down-sampled. In fact, this down-sampling could replace a rectangle of the leaf projection by a single texel. We call this low resolution texture as the *leaf distribution impostor*. During rendering, we address this leaf distribution impostor with texture coordinates u, v. If the first color coordinate of the looked up texel value is outside the [0,1] interval, then this texture coordinate does not address a leaf, and thus should be regarded as transparent. However, if the first coordinate is in [0,1], then first two color coordinates r, g are considered as the texture coordinates of the lower left corner of the leaf rectangle, and color coordinate b is regarded as the rotation angle of the leaf. The rotation angle selects the texture that represents this rotation, and u, v texture coordinates are translated by r, g and scaled by size s of the leaf rectangle, i.e. $u' = (u-r)/s, v' = (v-g)/s$ will be the texture coordinates of the single leaf texture selected by component b.

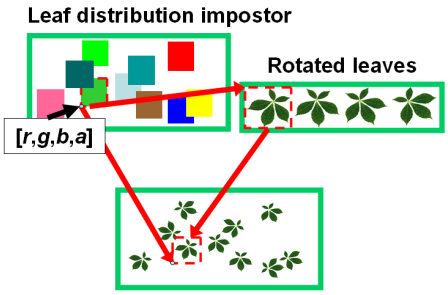

Figure 5: *Indirect texturing. The billboard is replaced by a leaf distribution impostor and the rotated images of the leaves.*

A texel of the leaf distribution impostor stores the position and the orientation of at most a single leaf whose bounding rectangle overlaps with this texel. As can be seen in figure 5, it may happen that multiple bounding rectangles overlap. Since only one of the overlapping leaf can be stored in a single pixel, this simplification may result in noticeable artifacts in form of clipped leaves if a fully transparent part of the leaf texture occludes another leaf. Such artifacts can be mostly eliminated by using not just one but two leaf distribution impostors. The first impostor is created by rendering leafs in ascending, while the second with descending order. Thus if two leaves overlap, the first leaf is visible in the first impostor, while the second is visible in the second impostor. During the rendering of the indirect texture the fragment shader reads both impostors. If a texel contains no leaf, then the fragment is ignored. If both impostors refer to the same leaf, then its leaf texture is scaled and is applied. However, if the two impostors refer to two different leaves, then the first leaf texture is looked up and it is checked whether its corresponding texel is transparent. In case of a non-transparent texel, the first leaf texture is used. In case of a transparent texel, we look up the second impostor and use the leaf selected by this.

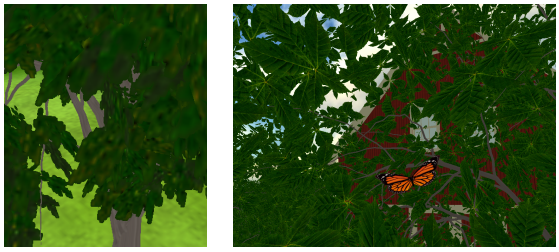

Figure 6: *Comparison of conventional filtered texturing and indirect texturing. Note that the high apparent texture resolution in the indirect texturing result.*

Figure 7: *Indirect texturing.*

Note that indirect texturing exploits the fact that the position of the leaves are not as important as their color pattern, thus stores these two kinds of information separately. The final texture is obtained run time without any storage overhead.

4. Shading

Billboard clouds approximately preserve the original geometry, thus they can also replace the original tree when shadows or global illumination effects are computed. However, when direct illumination is calculated, the application of the same normal for all leaves belonging to a cluster would make the planar substitution too obvious for the observer. Thus for direct illumination computation, the original normals of the leaves are used, which are stored in a normal map associated with the impostor plane. If no indirect texturing is used, then the normal map may store normals in object space. However, in the case of indirect texturing the normal map should have a similar structure as the rotated leaves texture. The values in this texture store the modulation with respect to the main projection direction, and should be transformed to tangent space during rendering.

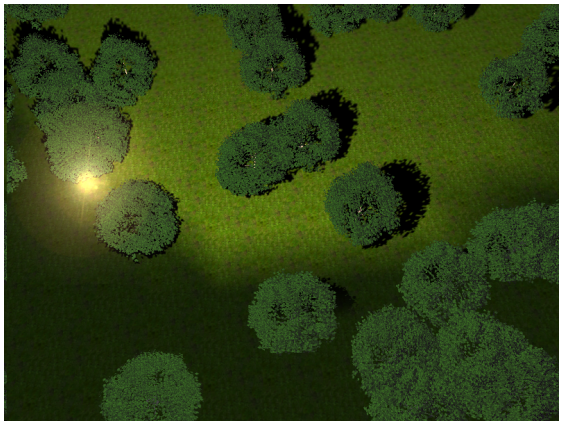

Figure 8: *Trees with illumination and shadow computation.*

To compute shadows, i.e. the visibility from point and directional lights (the direction of the sun in particular), the classical z-buffer shadow algorithm has been implemented with the modification that our implementation skips those fragments which corresponds to transparent texels of the impostor planes.

Assuming a single directional light source representing the sun, and sky light illumination, the reflected radiance of a leaf is computed by the following approximation of the rendering equation:

$$L^r = L^{sun} \cdot (k_d \cdot (\vec{N} \cdot \vec{S}) + k_s \cdot (\vec{N} \cdot \vec{H})^n)) \cdot v + L^{env} \cdot a,$$

where L^{sun} and $\vec{S}$ are the radiance and direction of the sun, respectively, k_d is the wavelength dependent diffuse reflectance of the leaf, k_s is the specular reflectance, n is the shininess, $\vec{N}$ is the unit normal vector, $\vec{H}$ is the unit halfway vector between the illumination and viewing directions, v is the visibility indicator obtained with shadow mapping, L^{env} is the ambient radiance of the leaf's local environment, and a is the albedo of the leaf. The albedo and the diffuse/specular reflection parameters are not independent, but the albedo can be expressed from them. We use the following approximation [5]:

$$a \approx k_d \cdot \pi + k_s \cdot \frac{2\pi}{n+1}.$$

In order to obtain the ambient radiance of a leaf, that is to simulate indirect illumination and sky lighting, an obscurances approach has been used [10]. During preprocessing, random global directions are sampled for a single billboard cloud tree. The global direction is used as a projection direction. The tree is rendered with the graphics hardware applying a depth peeling algorithm to find not only the closest visible point but all pairs of points that are visible from each other in the given direction. This information is used then to obtain a global occlusion factor o approximating the portion of the illuminating hemisphere in which the sky is not visible from the leaf. Simultaneously the average color of the occluding leaves o_c is also calculated. Since both faces of the leaves can be lit, we compute two obscurance maps, the first represents leaf sides having positive z coordinate in their normal vector, the second represents leaf sides having negative values. Using these values, the ambient radiance around a leaf can be approximated as:

$$L^{env} \approx L^{sky}(\vec{N}) \cdot (1-o) + o_c \cdot o,$$

where $L^{sky}(\vec{N})$ is the average radiance of the sky around the direction of the normal vector of the leaf, and o and o_c are the obscurance values depending on the actual leaf side.

The simplified pixel shader code evaluating the color of a leaf point is:

```
float3 N = tex2D(normalmap, uv);
float4 obs = tex2d(obsurancemap, uv);
float  o_c = obs.a;  // ratio occlusion
float  o   = obs.rgb;// occluder color
float3 Lenv = Lsky * (1-o) + o_c * o;
float3 diff = kd * max(dot(N,L),0);
float  spec = ks * pow(max(dot(H,N),0), n);
return Lsun * (diff + spec) * v + a * Lenv;
```

5. Results

The proposed algorithm has been implemented in OpenGL/Cg environment, integrated into the Ogre3D game engine, and run on an NV6800GT graphics card. The tree used in the experiments is a European chestnut (Castanae Sativa) having 11291 leaves defined by 112910 faces and 338731 vertices. The trunk of the tree has 46174 faces. The leaves have been converted to 32 billboards

using the proposed algorithm. When we did not apply indirect texturing, the billboard resolution was 512×512. Setting the screen resolution to 1024×768, leaf rendering is speeded up from 40 FPS to 278 FPS when the leaf polygons are replaced by the billboard cloud. The comparison of the images of the original polygonal tree and the billboard cloud tree is shown in figure 3. Note that this level of similarity is maintained for all directions, and the impostor tree also provides realistic parallax effects.

Figure 9 shows a terrain of 4960 polygons with 2000 trees rendered on 30 FPS without instancing (the rendering is CPU limited). We used the proposed level of detail techniques to dynamically reduce the number of leaf cluster impostors per tree for distant trees from 32 to 16 and even to 8.

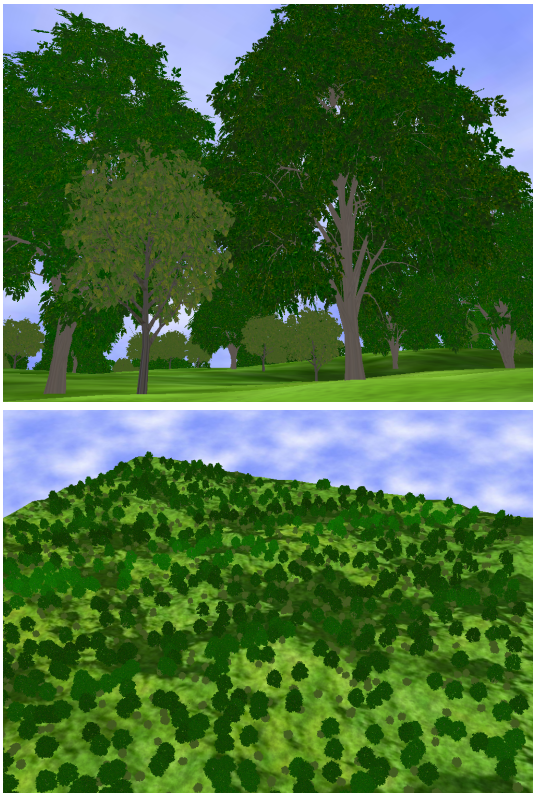

Figure 9: *Two snapshots from a 30 FPS animation showing 2000 trees without shadow and illumination computation*

6. Conclusions

This paper presented a tree rendering algorithm where clusters of leaves are represented by semi-transparent billboards. The automatic clustering algorithm finds an impostor in a way that the represented leaves lie approximately in its plane. This approach is equivalent to moving and rotating the leaves a little toward their common planes, thus this representation keeps much of the original geometry information. Since the impostors are not rotated with the camera, the proposed representation can provide parallax effects and view dependent occlusions for any direction. We also discussed how leaf textures can be visualized without posing high resolution requirements for the impostors, and the possibility of including smooth level of detail techniques.

7. Acknowledgement

This work has been supported by OTKA (ref. No.: T042735), GameTools FP6 (IST-2-004363) project, TIN2004-07451-C03-01 project from the Spanish Government, and by the Spanish-Hungarian Fund (E-26/04).

References

1. C. Andujar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and J. Vinacua. Computing maximal tiles and application to impostor-based simplification. *Computer Graphics Forum*, 23(3):401–410, 2004.

2. X. Décoret, F. Durand, F. Sillion, and J. Dorsey. Billboard clouds for extreme model simplification. In *SIGGRAPH '2003 Proceedings*, pages 689–696, 2003.

3. O. Deussen, P. Hanrahan, R. Lintermann, R. Mech, M. Pharr, and P. Prusinkiewicz. Interactive modeling and rendering of plant ecosystems. In *SIGGRAPH '98 Proceedings*, pages 275–286, 1998.

4. I. Garcia, M. Sbert, and L. Szirmay-Kalos. Leaf cluster impostors for tree rendering with parallax. In *Eurographics Conference. Short papers.*, 2005.

5. E. Lafortune and Y. D. Willems. Using the modified Phong reflectance model for physically based rendering. Technical Report RP-CW-197, Department of Computing Science, K.U. Leuven, 1994.

6. D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann, 2002.

7. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

8. N. Max, O. Deussen, and B. Keating. Hierarchical image-based rendering using texture mapping. In *Eurographics Workshop on Rendering*, pages 57–62, 1999.

9. N. Max and K. Ohsaki. Rendering trees from precomputed z-buffer views. In *Eurographics Workshop on Rendering*, pages 74–81, 1995.

10. Alex Mendez, Mateu Sbert, Jordi Cata, Nico Sunyer, and Sergi Funtane. Real-time obscurances with color

bleeding. In Wolfgang Engel, editor, *ShaderX4: Advanced Rendering Techniques*. Charles River Media, 2005.

11. A. Meyer, F. Neyeret, and Poulin. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering*, 2001.

12. A. Reche, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics*, 23(3), 2004.

13. G. Schaufler. Nailboards: A rendering primitive for image caching in dynamic scenes. In *Eurographics Workshop on Rendering*, pages 151–162, 1997.

14. G. Szijártó. 2.5 dimensional impostors for realistic trees and forests. In Kim Pallister, editor, *Game Programming Gems 5*, pages 527–538. Charles River Media, 2005.

15. G. Szijártó and J.: Koloszár. Real-time hardware accelerated rendering of forests at human scale. *Journal of WSCG*, 2004.

16. E. Weisstein. World of mathematics. 2003. http://mathworld.wolfram.com/Eigenvector.html.