GPU Based Stochastic Foliage Simplification

Sebastian Fichtner*
University Konstanz

Figure 1: *Forest scene simplified with $h = 0.7$ and total $\lambda = 0.06$. The viewing volume contains 368 trees made of 61 mio vertices*

Abstract

This bachelor project is dedicated to interactive rendering of tree foliage using stochastic simplification on the graphics hardware.

After giving a brief overview on latest level of detail methods for foliage rendering, the concept of stochastic simplification as introduced by [Cook et al. 2007] is discussed followed by a report on our GPU based implementation of this method for tree foliage. In the end we can show that stochastic simplification offers a substantial performance benefit and allows to produce quality images of complex plants at real time on customary hardware.

Keywords: Foliage, GPU, level of detail, stochastic simplification

1 Introduction

Visualizing nature is one of the biggest challenges in the field of computer graphics. Objects that result from natural growth rather than construction are hard to emulate on a computer and demand far more geometry information to describe them faithfully. A first system to generate authentic images of complete nature scenes was introduced by [Deussen et al. 1998].

Computational generation of plant images must deal with both major steps of displaying 3d-graphics: modeling and rendering. Numerous procedural methods and some sophisticated tools have been developed with respect to botanical knowledge, that allow to create realistic and highly detailed plant models. While modeling is concerned with the effectiveness of image generation, rendering must address its efficiency. Since rendering techniques emerging from other areas are often applicable to plants, various possible approaches to improve efficiency unfold. One common strategy to increase efficiency is to cull invisible geometry in eye space. Another one is to simplify geometry in object space which is often referred to as "level of detail". While culling is mostly independent of the type of objects (except for heuristic methods of occlusion culling), simplification can take advantage of the way objects are structured and perceived. Foliage for instance has some unique characteristics and deserves special attention in the context of simplification. Section (2) gives a brief overview on simplification methods for foliage. Section (3) explains stochastic simplification with an eye on foliage rendering. Section (4) shows how we implemented the method for tree models utilizing the GPU. In sections (5) and (6) we conclude this work by summarizing its results and giving an outlook on potential ways to improve the implementation.

*sebastian.fichtner@uni-konstanz.de

2 Related Work

A 3d-scene is usually defined by much more information than its rendered image reflects. Some obvious reasons for this are perspective distortion, rasterization and occlusion. The goal of simplification is to speed up rendering of geometry by processing only a minimal set of simple primitives. A central task is to determine a smaller set that results in the same visual experience. To accomplish that simplification approaches aim to find primitives that effectively describe a scene in object space. Therefore most related work relies on a multiresolution hierarchy for object approximation.

2.1 Hierarchical Simplification

For the cost of increased preprocessing and memory demands hierarchical simplification allows to drastically reduce the number of rendered primitives. It is based on the principle that each node in a hierarchy approximates its children. Looking at how nodes are created and rendered, two broad directions of hierarchical simplification may be identified which will be reviewed in the following sections.

2.1.1 Image Based

Geometry information can be approximated by images. [Szijártó and Koloszár 2003] suggested to create impostors from leaves in a first rendering step and use them to render the tree foliage in a second step. [Behrendt et al. 2005] presented an algorithm to cluster model vertices and find hierarchical billboard approximations wich were used to render large amounts of trees in the background. Similar techniques were applied by [Dietrich et al. 2005] and [Colditz et al. 2005].

2.1.2 Geometry Based

Geometry based approaches approximate models by geometric primitives like points, lines and polygons. An early attempt to dynamically mix polygons and simpler primitives in a hierarchical representation was presented by [Chen and Nguyen 2001]. They replace polygons by points for distant objects. [Deussen et al. 2002] introduced a system for interactive rendering of whole eco systems. For efficiency distant plants were simplified with respect to their

visual importance and rendered with points or lines depending on their structure. The number of rendered primitives was made adjustable.

In recent years the idea of applying multiresolution rendering techniques to foliage geometry received a lot of attention. Most algorithms basicly apply agglomerative clustering and join leaves controlled by an error function. [Remolar et al. 2003] and [Rebollo et al. 2007] develop their Foliage Simplification Algorithm to allow for variable levels of detail and take advantage of the GPU. Xiaopeng Zhang and Qingqiong Deng did intense research on the matter. They implemented efficient foliage simplification in [Deng et al. 2006] and [Zhang et al. 2006]. [Deng et al. 2007a] introduced a simplification method dedicated to coniferous foliage. Consideration of botanical knowledge was added by [Deng et al. 2007b]. [Deng and Zhang 2008] and [Deng et al. 2009] further improved efficiency of memory and CPU usage and exploited rendering capabilities of the GPU.

2.2 GPU Based Simplification

To benefit from GPU power, rendering must be divisible in numerous independent equal tasks for parallel processing. Those tasks correspond to the processing of single elements like points, leaves or cluster nodes. [Szijártó and Koloszár 2003] first process leaves to create images and then render these impostors in a second rendering stage. Their method successfully takes advantage of the local randomness of leaves without maintaining a hierarchy. Nodes from a conventional multi resolution hierarchy can not be stored in GPU memory without preparation, because the GPU can only select a continuous array segment for processing. Usually the CPU has to traverse the hierarchy to determine which nodes have to be processed. Then the relevant nodes are passed to the GPU. Transfering elements to the GPU each frame heavily decreases performance. A method to solve this is called "sequential point trees" and was introduced by [Dachsbacher et al. 2003]. They showed how to sequentialize a point hierarchy and store it on the GPU. The only data passed to the GPU is the fraction of elements to be processed resulting in great performance. [Rebollo et al. 2007] also store vertex data on the GPU and adapt their Foliage Simplification Algorithm to the graphics hardware. Another solution to the problem was developed by [Kalaiah and Varshney 2005] who relieved data transmission by stochastic compression of clusters and geometry extraction on the GPU. Recently [Deng et al. 2009] used an advanced version of sequential point trees to render foliage. All those sequentialization- and extraction strategies become dispensable with stochastic simplification as the following sections will show.

2.3 Stochastic Simplification

Hierarchical methods were applied to all kinds of objects. Tree foliage however has some distinct properties:

- It consists of large amounts of elements (leaves) which are similar in size, geometry and color.
- Elements are independent units and do not represent continuous surfaces.
- The global distribution of elements reflects the object (tree) structure whereas their local position can be considered as equally distributed or random.

Stochastic simplification as introduced by [Cook et al. 2007] is applicable to objects with these very properties. It was successfully used at Pixar to speed up rendering of high quality images. Its basic idea is that such objects can be well approximated by a random subset of their elements. So instead of generating additional approximating primitives as a preprocessing step, they just render a fraction of the given elements. This straight forward method is well suited for parallel processing on GPUs. The next section will look at it in greater detail and estimate how it can be applied to foliage.

3 Concept

Stochastic simplification is done by selecting a subset from the given elements for rendering. The selected elements must then be modified in color, size and possibly opacity to preserve the objects overall appearance. This modification only depends on the fraction of rendered geometry which of course is the same for each element of an object, so it can be done on the GPU.

3.1 Level of Detail

First the application must decide on how many elements to exclude. The object area B in image space is measured in pixels of the bounding box. B_0 is the predefined minimal area of an object at which it is rendered in full detail. Then the fraction $b = B/B_0$ indicates the area reduction of an object in image space and acts as the basis of decision. If $b < 1$ the object is simplified with respect to b. It is possible to simplify different types of objects at individual levels by giving each one its own minimal size B_0. Since the computation of a bounding box area in pixels is more complex than getting an objects camera distance d, we calculate b differently: If simplification should start at distance d_0 then

$$b = {d_0}^2/d^2 \tag{1}$$

where d is the objects actual camera distance. While b just represents the potential visible fraction of an objects full size in image space, the level of detail λ is the actual fraction of rendered geometry. Apart from the object size [Cook et al. 2007] also consider motion blur to determine λ. Since we're not rendering for a limited frame rate and plants are not moving in world space, we don't use motion blur effects. So λ only depends on b and an adjustable parameter h:

$$\lambda = b^{log_h(1/2)} \tag{2}$$

h specifies the value of b at which half of the elements are rendered and is used to calibrate the aggressiveness of detail reduction. We get the linear reduction of $\lambda = b$ by setting $h = 1/2$. [Cook et al. 2007] state that

> "h should never be greater than $1/2$ because λ_{size} would decrease faster than b; this would mean using fewer elements per pixel as the object got smaller, and in order to preserve the overall object area those elements would have to get larger on the screen as the object got smaller.".

However we don't inherit that restriction, because that very effect might be intended. Objects with greater camera distance are less important and might as well be displayed in less image space detail. After all, simplification is mostly a trade off between performance and image quality, and a more aggressive simplification combined with a great enough value of d_0 might just be fine for interactive foliage rendering as Figure 1 demonstrates. Furthermore, distant plants are more likely to be occluded by others.

3.2 Excluding Elements

The easiest way to select a fraction λ from N given elements is to choose the first λN elements from the buffer they are stored in. This guarantees for changes of λ that elements are always excluded (and

(a) *original model* **(b)** $\lambda = 0.05$ *without contrast preservation* **(c)** $\lambda = 0.05$ *with contrast preservation*

Figure 2: *The effects of area- and contrast preservation on an aesculus hippocastanum model*

included) in the same order. Randomizing the buffer order during preprocessing ensures that every determined subset is equivalent to a random selection. Sorting elements by their visual importance can further improve approximation precision.

3.3 Area Preservation

Excluding $(1-\lambda)N$ elements from rendering scales down the total area of rendered elements by λ. To compensate for that it may seem logical to scale up each selected element by $1/\lambda$. However to compute a correct scaling factor s we have to consider the difference between total- and visible area. The more elements an object contains, the smaller is one elements contribution to the objects visible area because more of the element is occluded by others. Aiming to preserve visible area [Cook et al. 2007] showed that s can be calculated using the ratio r of one elements average visible area to the objects total visible area:

$$s = \frac{1-(1-r)^{1/\lambda}}{r} \tag{3}$$

Due to the stochastic nature of this simplification method, object geometry is rather overlaid by some noise than substantially distorted. Deformations may only be visible at the silhouettes of far distant objects.

3.4 Contrast Preservation

The smaller a sample of a population is, the greater is the error under which it predicts the populations average of some attribute. In our case the error under which selected elements approximate the local or global average color grows with decreasing level of detail. A single leaf for example may drastically misrepresent the overall foliage color. [Cook et al. 2007] explain the importance of this effect by looking at how colors of individual fragments are determined during rasterization. Because simplification leads to a smaller number of elements per fragment, the approximation error of each fragment grows and causes higher contrasts between pixel colors. In addition to the involvement of supersampling this explanation implies the assumption that image quality is preferred over efficiency and that simplification only takes place when elements become smaller than pixels. However when elements are larger than pixels, contrast preservation becomes even more necessary because approximation errors become directly visible and the effect of color blending during the process of perception also gets lost with increasing element size. Optimal contrast preservation would set a selected elements color to the average color of the excluded elements it locally approximates. As mentioned earlier elements were randomly selected and ought to be processed independently and equally, so we can not determine their corresponding local average. Anyway, local averages adjust to the global average as the level of detail decreases, which enables us to approximately calculate a corrected color c' for an element color c only considering a factor α_λ and the overall average color $\bar{c}$:

$$c' = \bar{c} + \alpha_\lambda (c - \bar{c}) \tag{4}$$

$\alpha_\lambda \in [0 \dots 1]$ is a weight that declares how strongly the final color c' represents the original color c for a given level of detail λ. [Cook et al. 2007] found that $\alpha_\lambda = \sqrt{\lambda}$ conserves overall color variance.

3.5 Smooth Animation

Now we are able to render only as much elements as needed while preserving the objects appearance. But continuous changes of λ for a certain object would still result in artifacts due to elements popping in and out. This happens when moving the camera or shifting parameter h or d_0. To reduce that effect elements are slowly faded in and out either by size or by opacity. [Cook et al. 2007] linearly fade elements over the interval $[(\lambda-0.1)N \dots (\lambda+0.1)N]$ in terms of buffer position.

4 Implementation

4.1 Basics

Implementation was done with C++, OpenGL and GLSL in the XCode environment. After loading a tree model from file, we modify the normals of its leafs for better shading results. The orientation of the leafs can be considered as random, so shading them based on their implicit normals would not reflect the volumetric shape of the tree crown. We average each leaf normal with the normal pointing from the crowns center to the leaf. Shading is done in a fragment shader and involves computation of diffuse and specular

(a) $h = 0.2$, $\lambda = 0.38$ **(b)** $h = 0.5$, $\lambda = 0.11$ **(c)** $h = 0.8$, $\lambda = 0.02$

Figure 3: *A forest scene in 800 × 600 pixels. The viewing volume contains 332 trees made of 52.2 mio vertices*

intensities, while light color is a continuous function of the angle between light- and polygon normal simulating the artistic method of contrasting blue toned ambient- against red toned sunlight. To interactively examine the scene a camera system was implemented that allows game like movement. Finally a sky box was added to evaluate results against a realistic nice background.

4.2 Performance

To be able to compare the performance of our implementation with conventional methods on scenes with hundreds of tree instances, we first applied some common principles of efficient rendering. Strong acceleration was derived from comparing immediate mode against vertex arrays and vertex buffer objects. A single tree instance was rendered through the standard pipeline provided with normals, colors, texture coordinates and vertices. Vertex arrays were 4.5 times faster than immediate mode while VBOs were able increase frame rate by factor 5.1. In the following we assume VBOs being used.

Another substantial performance gain was achieved by culling. We implemented both- clip space- and eye space culling. To really challenge those implementations 1000 trees were put into a scene. Without simplification frame rate was approximately zero when no culling was applied, so culling was evaluated using simplification with parameter $h = 0.5$. Now having 10% of all trees in the viewing volume both methods were around 7 times faster than not to cull at all. But the different complexities only reveal them selves when not masked by a relative high rendering load. When all trees were culled, clip space culling only reached 27% of the performance of eye space culling. As expected, clip space culling can not compete, although we handle projection- and view matrix by ourselves in the camera system. Due to the perspective distortion caused by projection, all corners of a bounding box must be transformed to clip space and tested against the clipping cube. Culling in eye space only demands to test a bounding sphere against the planes of the viewing frustum and avoids multiplying view- and projection matrix.

4.3 Simplification

For each tree instance its level of detail and scaling factor are determined by the CPU according to equations (2) and (3). The average ratio $r \approx 0.0001$ of our tree models was determined by experimenting. Before rendering the first λN vertex buffer elements, λ and s are passed to the GPU.

4.3.1 Area Preservation

Our leafs are plain polygons. They are scaled by translating their vertices in a vertex shader. Therefore the coordinate relative to the center of the corresponding leaf must be provided with each vertex. We will call this the polygon coordinate. Since tree models are fully textured the color buffer object is available for storing those coordinates. How must polygon coordinates be scaled, to increase the area of a leaf by factor s? Decomposing the leaf polygon reduces this problem to scaling a triangle area as shown in Figure 4 where C is the center the polygon and c is any of its sides.

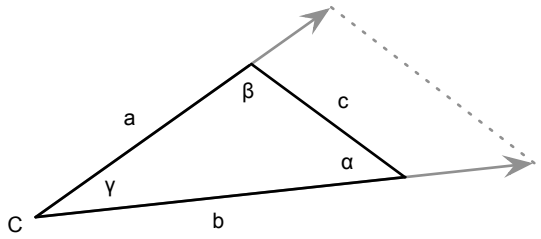

Figure 4: *Scaling the area of a plain polygon*

Scaling sides a and b by the same factor causes a parallel shift of c but doesn't change any of the angles. Now the triangle area T is given by the formula

$$T = \frac{|a|^2 \sin(\beta)\sin(\gamma)}{2\sin(\alpha)}$$

where $|a|$ denotes the length of side a. The scaled area sT resolves to

$$sT = \frac{(\sqrt{s}|a|)^2 \sin(\beta)\sin(\gamma)}{2\sin(\alpha)}$$

So polygon coordinates must be scaled by $\sqrt{s}$.

4.3.2 Contrast Preservation

The leafs of one tree are very similar in color. Since our tree models texture all their leafs with one and the same bitmap, there appeared to be no need for adjusting colors. Nevertheless contrast preservation might still be necessary. Shading leads to high contrasts in brightness due to the random character of leaf normal orientations. So to preserve contrast we adjust shading instead of coloring. As our shading implementation is based on Phong Shading, the varying part of visible intensity is a weighted sum of diffuse- and specular

reflection. To modify the variance of that part we must know its average value. Since we set the exponent of the specular cosine to 1, both terms are the product of a weighting factor and the cosine of the angle between two normalized vectors. So their average can be approximated using the expectation value $E(\cos)$ which is the average cosine over all possible angles:

$$E(\cos) = \frac{2}{\pi} \int_0^{\pi/2} \cos(x)\ \mathrm{d}x = \frac{2}{\pi} \approx 0.63662 \tag{5}$$

Note that the specular term is weighted by zero if the angle between eye- and reflection vector is greater than $\pi/2$. Figure 2 illustrates area- and contrast preservation showing orthographic projections of a tree model with different parameters.

4.3.3 Smooth Animation

To smoothly animate changes in level of detail we fade out elements by reducing their opacity or size. Because fading depends on the elements relative position in the buffer, we make that position available to the GPU by storing it as a value between 0.0 an 1.0 in the alpha channel of the vertex color. The RGB channels already provide polygon coordinates as previously explained. When using the w component of the vertex, shaders can not be switched off because the standard pipeline would not set w to 1.0 for matrix transformations. Our vertex shader extracts the buffer position i and calculates a fading factor f which declares how visible the element shall be. f is a function of λ and i. A self-evident way to define f is to linearly fade within a certain radius R around i which makes it appropriate to render $\min(N,\ (\lambda + R)N)$ elements.

$$f = \begin{cases} 1 - \frac{i-\lambda+R}{2R} & \text{if} \quad i > \lambda - R \\ 1 & \text{else} \end{cases} \tag{6}$$

We experimented with some modifications of this function:

- increasing performance by rendering only λN elements and fade out earlier so that $f = 0$ for $\lambda = i$
- smoothing f over the fading interval using the function *smoothstep* provided in GLSL
- linear and quadratic shrinking of R with increasing λ so that all elements are fully visible for $\lambda = 1$
- increasing performance by linear shrinking R with decreasing λ so that only a certain fraction of rendered elements is faded rather than a certain fraction of all elements
- choosing $R = \min(1 - \lambda,\ c)$ for a certain constant c so that all elements are fully visible for $\lambda = 1$

Rendering only λN elements didn't preserve visible leaf density even when the second modification was added. When shrinking R with growing λ the fading interval got either too large for small values of λ or too small for large ones. But choosing R as a fraction of λ won back some performance without disturbing fading smoothness. We also kept the last modification to make sure trees within camera distance d_0 don't get altered.

To fade elements by opacity the vertex shader passes f to the fragment shader which multiplies the alpha channel of the texture by f. This leads to better image quality than fading by size especially when no anti aliasing technique is applied. But utilizing alpha blending also has two drawbacks: First, the alpha test and -blend function can hardly be modified for other purposes. For example the alpha test must succeed for values greater than zero in our implementation. A second drawback is that f is applied to each fragment although it is constant over the whole element. The fragment shader may be completely relieved of the fading issue by letting the vertex shader scale elements by fs instead of s.

5 Results

Evaluation of results was done on a MacBook Pro 4,1 with 2.4 GHz and a GeForce8600M GT. It involved rendering scenes containing up to 1000 instances of 10 different tree models. Without simplification those models are made of 157368 vertices at average. Figure 5 illustrates how the total detail level of a scene and frame rate relate to parameter h. Both plots reflect the same data set of a scene where 90% of all trees were culled and viewport resolution was set to 800×600 pixels. Note that models are not simplified at all for $h = 0$. Figure 3 demonstrates the capabilities of our implementation in terms of performance and quality preservation. When moving through a scene the smoothness of fading could be verified for large values of h. With lower h leafs become smaller in image space and fading artifacts get masked by aliasing artifacts. Turning contrast preservation off clearly decreases image quality.

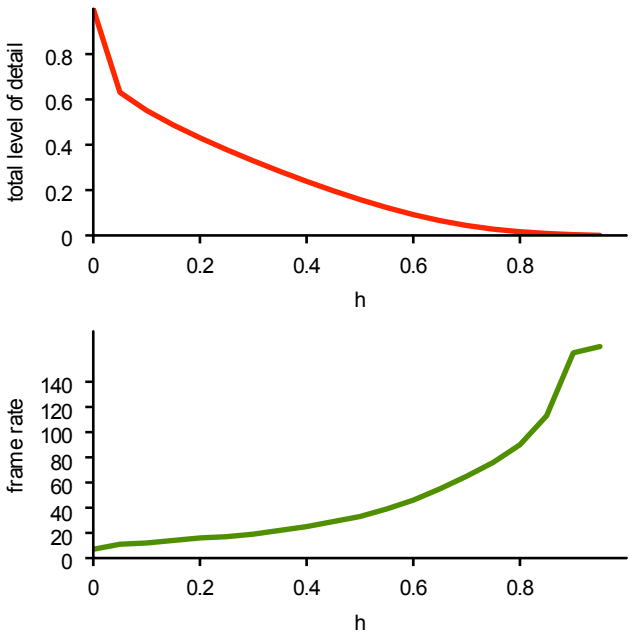

Figure 5: *Reduction of rendered geometry and performance gain for increasing parameter h in a forest scene of 15.8 mio vertices*

6 Future Work

A lot of possible improvements are left for future implementation. One basic concern is the effectiveness of element usage which brings up some continuative questions: Can billboards somehow be integrated in order to cover more visible area with less primitives? Are there any other primitives thinkable that might replace far distant elements? How can the priority order of elements be optimized for leafs of a tree crown? Would the use of an octree speed up culling? It is also very desirable to use different shader programs for foliage and edges, so a separate mesh simplification methods could be applied to trunks and edges. Furthermore dedicated foliage shaders would be able to realize visual effects specific

for foliage like half transparent shading or procedural movement. The contrast preservation in our shading implementation should be evaluated on foliage with a maximum variance in color, because in such a case it might not be appropriate to regard only light intensity. Apart from that it appeared that parameter r is quiet sensitive to different types of trees. It should be determined for each model individually. A models visible area in image space might be approximated by rendering it to an off screen buffer and summing up the alpha values. Then the average visible element area would be the element area multiplied by $2/\pi$ analogue to (5).

References

BEHRENDT, S., COLDITZ, C., FRANZKE, O., KOPF, J., AND DEUSSEN, O. 2005. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum 24*, 507–516.

CHEN, B., AND NGUYEN, M. X. 2001. Pop: a hybrid point and polygon rendering system for large data. In *VIS '01: Proceedings of the conference on Visualization '01*, IEEE Computer Society, Washington, DC, USA, 45–52.

COLDITZ, C., COCONU, L., DEUSSEN, O., AND HEGE, C. 2005. Realtime rendering of complex photorealistic landscapes using hybrid level-of-detail approaches. In *Trends in Real-Time Landscape Visualization and Participation*, 97–106.

COOK, R. L., HALSTEAD, J., PLANCK, M., AND RYU, D. 2007. Stochastic simplification of aggregate detail. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 79.

DACHSBACHER, C., VOGELGSANG, C., AND STAMMINGER, M. 2003. Sequential point trees. *ACM Trans. Graph. 22*, 3, 657–662.

DENG, Q., AND ZHANG, X. 2008. Grid-based view-dependent foliage simplification. *Journal of Computational Information Systems*.

DENG, Q., ZHANG, X., AND JAEGER, M. 2006. Efficient multiresolution of foliage for real-time rendering. In *PMA '06: Proceedings of the 2006 International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications*, IEEE Computer Society, Washington, DC, USA, 307–314.

DENG, Q., ZHANG, X., GAY, S., AND LEI, X. 2007. Continuous lod model of coniferous foliage. *IJVR 6*, 4, 77–84.

DENG, Q., ZHANG, X., AND JAEGER, M. 2007. View-dependent hierarchical foliage simplification. *Technologies for E-Learning and Digital Entertainment*, 44–55.

DENG, Q., ZHANG, X., YANG, G., AND JAEGER, M. 2009. Multiresolution foliage for forest rendering. *Computer Animation and Virtual Worlds 21*, 1, 1–23.

DEUSSEN, O., HANRAHAN, P., LINTERMANN, B., MĚCH, R., PHARR, M., AND PRUSINKIEWICZ, P. 1998. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 275–286.

DEUSSEN, O., COLDITZ, C., STAMMINGER, M., AND DRETTAKIS, G. 2002. Interactive visualization of complex plant ecosystems. In *VIS '02: Proceedings of the conference on Visualization '02*, IEEE Computer Society, Washington, DC, USA, 219–226.

DIETRICH, A., COLDITZ, C., DEUSSEN, O., AND SLUSALLEK, P. 2005. Realistic and Interactive Visualization of High-Density Plant Ecosystems. In *Natural Phenomena 2005, Proceedings of the Eurographics Workshop on Natural Phenomena*, 73–81.

KALAIAH, A., AND VARSHNEY, A. 2005. Statistical geometry representation for efficient transmission and rendering. *ACM Trans. Graph. 24*, 2, 348–373.

REBOLLO, C., REMOLAR, I., CHOVER, M., GUMBAU, J., AND RIPOLLES, O. 2007. A clustering framework for real-time rendering of tree foliage. *JOURNAL OF COMPUTERS 2*, 4, 57–67.

REMOLAR, I., CHOVER, M., RIBELLES, J., AND BELMONTE, O. 2003. View-dependent multiresolution model for foliage. In *WSCG*.

SZIJÁRTÓ, G., AND KOLOSZÁR, J. 2003. Hardware accelerated rendering of foliage for real-time applications. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, ACM, New York, NY, USA, 141–148.

ZHANG, X., BLAISE, F., AND JAEGER, M. 2006. Multiresolution plant models with complex organs. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, ACM, New York, NY, USA, 331–334.