MÀSTER EN COMPUTACIÓ

DEPARTAMENT D'INFORMÀTICA I MATEMÀTICA APLICADA

UNIVERSITAT DE GIRONA

Generation and interactive visualization of 3D vegetation

Master Thesis

Ismael GARCÍA FERNÁNDEZ

Advisors:
Prof. Mateu SBERT CASASAYAS
Dr. Gustavo PATOW

July 2007

Abstract

In this master thesis we present a simple method to render complex trees on high frame rates while maintaining parallax effects. Based on the recognition that a planar impostor is accurate if the represented polygon is in its plane, we find an impostor for each of those groups of tree leaves that lie approximately in the same plane. The groups are built automatically by a clustering algorithm. Unlike view-aligned billboards, these impostors are not rotated when the camera moves, thus the expected parallax effects are provided. On the other hand, clustering allows the replacement of a large number of leaves by a single semi-transparent quadrilateral, which improves rendering time considerably. Our impostors well represent the tree from any direction and provide accurate depth values, thus the method is also good for shadow computation. We combined this representation with a new texture-based representation for the foliage. The technique provides several new contributions with respect to previous approaches. The new algorithm allows progressive level of detail both at the geometric and at the shader levels. It also preserves the parallax effects of the original polygonal model keeping leaf positions, orientations, and preserving the overlapping of the leaves as seen from any view point. In addition, the texture-based representation provides high-definition close views without introducing high memory requeriments. We adapted a realistic lighting model with soft-shadows and a global illumination precomputation, allowing to render highly complex scenes with thousands of trees in real time.

Categories and Subject Descriptors (according to ACM CCS): http://www.acm.org/class/1998/ I.3.3 [Computer Graphics]: Image generation, I.3.7 [Computer Graphics]: 3D Graphics and Realism

Funding

My work in this dissertation has been possible thanks to grant BR UdG from the University of Girona and supported by GameTools FP6 (IST-2-004363) project, TIN2004-07451-C03-01 and TIN2004-07672-C03-01 projects from the Spanish Government.

Acknowledgements

En los momentos de escribir estos agradecimientos, aparecen en mi mente muchas imágenes de momentos especiales durante la realización de esta Master Thesis, y en general durante los 5 años que he estado trabajando en la Universitat de Girona; de personas que con las que he compartido buenas experiencias durante este tiempo, ayudándome y ofreciéndome el apoyo en momentos difíciles.

En primer lugar, a Mateu Sbert, por darme la oportunidad de poder trabajar en el tema que me apasiona, por sus consejos y experiencia, ofreciéndome además la posibilidad de vivir de mi pasión. Nunca olvidaré todas las cosas buenas.

También tengo un sincero agradecimiento hacia László Szirmay-Kalos de la Universidad de Budapest, porque empezamos juntos en el tema de esta Master Thesis, por ayudarme con constancia y paciencia, además de escucharme y animarme a realizar el doctorado.

A Fran, Gus y Tere, no tengo muchas palabras, simplemente un nudo en el estómago que me impide expresar todo lo que debo agradecerles, por aguantarme en el trabajo juntos, por escucharme y mostrarse como verdaderos amigos, más que únicamente compañeros de trabajo. Os debo más de lo que os imagináis. En especial a ti, Gus, porque hemos acabado este trabajo juntos.

A Skukie, porque en el corto periodo que hemos trabajado juntos, he podido observar que eres una persona de gran corazón, siempre guardaré un buen recuerdo de ti, deseando que vuelvas a Girona para poder tener esas charlas sinceras.

A mis compañeros de laboratorio, Sergi, Nico y Jordi Catà de Girona y Tamás Umenhoffer, László Szecsi y Mátyás Premecz de Budapest, por hacer del día a día, el recuerdo de un conjunto de buenos momentos. Y a los compañeros de *cafés y comidas*, Albert, Àlex, Narcís, Carles, Gonzalo, Francesc, Marité y ... por mostrar vuestro cariño y aprecio por ese chico nervioso que siempre va con prisas... Y a ti Céline, gracias por el favor que me hiciste.

Mi familia, por darme más de lo que nunca podré agradecer, y por haberme apoyado y animado incondicionalmente en mi trabajo.

Y en último lugar, pero en el primero de mi corazón a Cris, porque tú conoces lo mejor y lo peor de mi, y aún así, deseas seguir pasando el resto de la vida juntos, increíble!! Por escucharme de forma abierta, y por tu gran paciencia que espero poder recompensar algún día.

Ismael García Fernández

Girona, July 21, 2007

Contents

List of Figures

List of Tables

Chapter 1

Introduction

IN this chapter we provide an overview of the work presented in this document, starting from a general introduction in the area of interactive tree rendering to a more comprehensive description.

Since more than 20 years ago, computer graphics research has taken special attention to the visual representation of natural phenomena. We would like to initially point out some of the intrinsic difficulties of visually representing such complex natural effects, specially with trees, which are our main object of study. Natural phenomena visual effects, observed from the human scale, provide clear evidences of complex shape descriptions, inherent dynamic evolution and behaviour, related with physical and chemical processes.

Starting from this general scope, a study of the previous research work about natural phenomena simulation in computer graphics is presented in chapter 2. This study has driven us to understand the great challenges still remaining in this field. For example, as we mention earlier, the shape complexity of natural phenomena, specially in our current object of study, trees, move us to one of the first questions we could pose. How can we build such complex natural structures using a computer? Since many years ago, a lot of research has been focused on this. Therefore, several authors introduced structure models for describing this complex natural appearence. From fractals (e.g. L-Systems), graphs, since geometric and image based models have been presented. Once upon we have a representation model, we should evaluate which visual effects can be handled by this model. In the case of plants and trees, the accurate fidelity with real species, lighting conditions, dynamic motion and growth evolution are probably some of the most important visual properties that the models should take into account. In addition, these models must be a trade of between the preservation of such realistic visual natural complexity and, at the same time, without over passing the available hardware capabilities. Even more, assuming that in our case, the object of study is providing realistic real-time representations of such natural effects.

The interactive rendering of vast natural scenes, involving hundreds of thousands of trees and other vegetal species is an open problem. This is specially true in forest scenes, whose complexity is impossible to model in detail without efficient level-of-detail (LoD) algorithms and data structures. The most commonly used plant and tree models are geometrically generated [Xfr06, EAS06], providing complex geometric models with hundreds of thousands polygons. These models can provide foliage representations achieving the natural parallax effect, when they are observed interactively from different viewpoints. However, these models cannot be used in real-time visualizations of natural scenes with thousands of trees. Another common representation are image-based foliage representations usually failing to keep the parallax effect, providing low resolution detail, or falling back to wrong foliage alignment.

In our current work, we present a simple image-based method to render complex trees on high frame rates while maintaining parallax effects. Since the leaf canopy of trees contributes with a huge number of polygons in tree models, and the traditional geometry simplification methods are typically not applicable, because leaves consist of many individual polygons, new solutions must be found.

In chapter 3, we an extremal geometric simplification method based on the recognition that a planar impostor is accurate if the represented polygon is in its plane. We find an impostor for each of those groups of tree leaves that lie approximately in the same plane (see figure 1.1). The groups are built automatically by a clustering algorithm. The generated impostors are not rotated when the camera moves, thus the expected parallax effects are provided. On the other hand, clustering allows the replacement of a large number of leaves by a single semi-transparent quadrilateral, which improves rendering time considerably. Our impostors correctly represent the tree from any direction and provide accurate depth values, thus the method is also good for realistic lighting computations (e.g shadows).

In addition, we include a new texture-based representation for the foliage, detailed in chapter 7.1. The technique provides several new contributions with respect to previous approaches. The new algorithm allows progressive level of detail both at the geometric and at the shader levels. It also preserves the parallax effects of the original complex polygonal model keeping leaf positions, orientations, and preserving the overlapping of the leaves as seen from any view point.

This texture-based representation provides high-definition close views without introducing high memory requeriments, compared to all the previous image-based approches available for trees.

In chapter 5 we describe the considerations that should be taken into account to achieve real-time vegetation rendering at high frame rates with the representation defined in previous chapters. In addition we describe the adapted lighting model with soft shadows and a global illumination precomputation, allowing to realistacally render highly complex scenes with thousands of trees in real time.

In chapter 6 we present a detailed description of the results obtained, comparing them with other previously studied approaches and discussing implementation details that can be interesting to understand the underlying benefits and drawbacks of the presented solutions.

Finally in chapter 7 we will comment in detail some of the proposed improvements and extensions of the presented foliage model that can enable us to incorporate new or more realistic natural effects such as dynamic motion, growth evolution stages and an even more realistic lighting model.

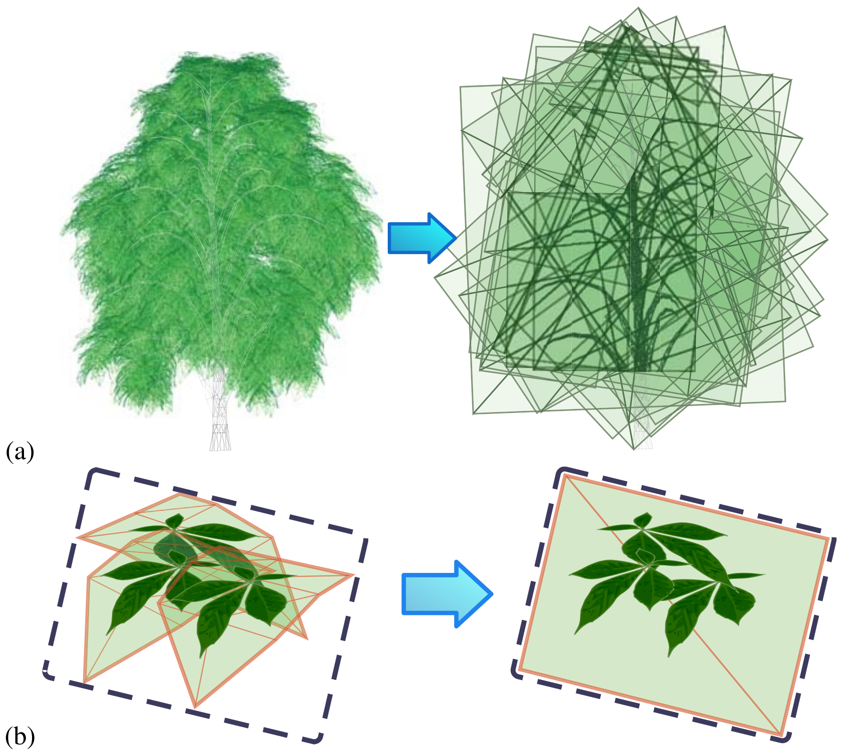


Figure 1.1: Replacing large groups of leaves by semi-transparent quadrilaterals with a texture map. (a) The polygonal leaves are replaced by a billboard cloud. (b) Each group of leaves is replaced by a billboard.

Chapter 2

State-of-the-art

We present a general overview of the most relevant previous works related to realtime rendering of vegetation. The problem of rendering complex natural scenes has already received a lot of attention. There are two general approaches for interactive realistic rendering for trees: geometry-based and image-based techniques. Our categorization of the rendering algorithms is based on the main rendering primitive used. Of course this distinction is not always applicable; hybrid methods will be described in the section of their main contribution (typically, this would be either geometry-based, or image-based methods).

The reader should take into account that some of the initial works commented here, were not realtime at the time of their publication few years ago, but they have been seminal work of further extensions appeared more recently.

We have also included special sections that spot some specific topics like realtime vegetation ilumination techniques in section 2.3 and some general procedural image-based methods related to our new texture based solution in section 2.4. At the end of the chapter in section 2.5, we summarize the presented algorithms classifying them with respect to the general features provided or not by each method and we point out some of the inspiring motivations we gathered from this study.

2.1 Geometry-based techniques

Polygonal and especially triangular models have traditionally been the predominant rendering primitive in computer graphics. Most of the hardware acceleration developments have also focused on triangular data; therefore this rendering primitive has a certain advantage when it comes to realtime rendering.

Many polygonal rendering methods for vegetation apply generic acceleration methods such as triangle strips, to speed up rendering. Furthermore, see [AMH02] for an overview of such methods.

Although these geometric techniques are not directly related with our work, their study have been useful to recognize some their main drawbacks in interactive tree rendering environments, that we can aim to solve using our image-based strategy.

2.1.1 Triangle based algorithms

An early algorithm for real-time rendering of fractal plants and trees has been introduced by Oppenheimer in 1986 [Opp86]. Due to the early time of their publication, their paper discusses many aspects of fractal modeling and self similarity that can now be presumed. Natural trees are not strictly self similar; there is typically some deviation in the symmetry due to environmental differences. The greater the deviation of the tree parameters, the more random and gnarled the tree will appear. Oppenheimer defines the resulting tree *as statistically* self-similar to represent this fact. The fractal model in the system introduces ways to control the variance of these parameters.

In addition, bump mapped polygonal prisms are used to render the resulting models. The branches emanating from a limb simply interpenetrate the limb; several prisms are linked together to approximate curvilinear shapes. The limbs texture is procedurally generated by adding fractal noise to a ramp, and then passing the result through a sawtooth function. The function used in [Opp86] also wraps seamlessly in u and v. The rendering of leaves is not discussed in the paper; however sample screenshots in [Opp86] do feature leaves and blossoms, so the algorithm can be adapted accordingly (see figure 2.1(a)). At the time of publication, the rendering of complex tree images could take several hours to render; obviously trying to design a desired tree at this rate is not very effective.

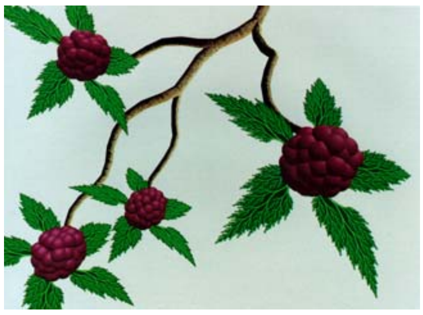
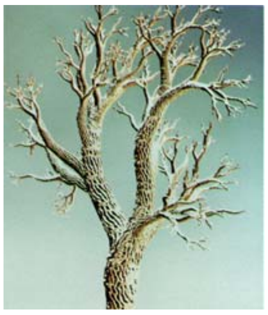
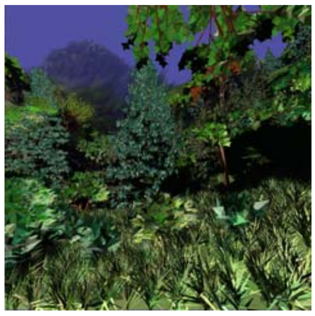
(a)  (b)  (c) 

Figure 2.1: In *Real time design and animation of fractal plants and trees* [Opp86]. (a) Fractal leaves and small bump mapped branches of a cherry tree. (b) Fractal complex branches. In *Multiresolution rendering of complex botanical scenes* [MFI97]. (c) Complex scene.

Marshall et al. [MFI97] in 1997 presented a system for rendering very large collections of randomly parameterized plants. Their multiresolution rendering system compiles plant models into a hierarchical volume approximation based on irregular tetrahedra. This partitioning creates a binary tree similar to BSP trees, which can be traversed quite efficiently. Their plant model allows plant information to be stored at various levels of detail and memory usage. The generation of actual geometry for any subvolume can be delayed until it is needed. This drastically reduces memory consumption and initialization time, as the binary tree does not need to be built fully. This compilation progress begins with a full tetrahedral volume as a first approximation to an object, which is then further refined as needed to accomodate individual polygons. Rendering then proceeds in a typical multiresolution rendering way. For objects that are close to the viewer, explicit polygons are generated, while objects that are hidden or further away are rendered as groups of microsurfaces approximating the contents of the bounding tetrahedra (see figure 2.1).

Since the leaf canopy of trees contributes a huge number of polygons in tree models, Remolar et al. [RCRB03] proposed a simplification method that specifically targets foliage. Traditional geometry simplification methods are typically not applicable, since leaves consist of many individual polygons. Therefore, topology preserving algorithms will not succeed, and non-preserving methods typically introduce a significant change to the overall appearance. The algorithm described in [RCRB03] aims at diminishing the number of polygons in the crown, while maintaining overal appearance. This is achieved by introducing a new method, the leaf collapse: Two leaves are replaced with a new one that preserves an area similar to that of the collapsed leaves. In a preprocessing step, a multiresolution model is created from a sequence of leaves collapses. The resulting data structure is therefore created bottom-up as a binary tree, with a polygonal representation of individual leaves (the highest resolution) as the leaves, and the root nodes being the polygons required for a minimum representation. Therefore, the resulting data structure is a 'forest' of binary trees (see figure 2.2).

Lluch et al. [LCV03] proposed another multiresolution method based on parametric L-systems. Their algorithm is based on a metric that quantifies the visual relevance of the branches of a tree. This paper focuses on the branch structures; leaves have not been considered (see figure 2.3). The level of detail algorithm operates on the underlying L-system, thus avoiding the generation of geometry that will not be rendered. To capture the relevance of individual chains generated from the L-system, an intermediate *weighted tree* data structure is created. From this data structure, the *multiresolution chain* can then be generated. In addition to the output of the L-system itself, the multiresolution chain supports two new instructions: *SAVE(id)* and *RESTORE(id)*. These can be used to store and restore the current state for some unique identifier. This allows the weighted tree to be stored as a reordered chain that is sorted by individual node weights. Higher weights (more important nodes) are stored first; finer LODs can be added to any point of the tree through the *RESTORE(id)* instructions. Later on, they developed an image-based technique aimining to have a more efficient foliage representation (see section 2.2).

Lluch et al. [LVF$^+$01] observed that one of the main issues of rendering polygonal trees is that many growth models produce disconnected meshes for each branch; bifurcations are often simplified as the interpenetration of such meshes. If a single mesh could be obtained instead, this would facilitate the application of multiresolution and simplification methods. The tree representation used in their proposal is based on sequences of elliptical (or circular) contours; a library created by the same research group is then used to obtain meshes from two such contours. However, bifurcations require special treatment, as they cannot be represented as elliptical structures. To handle these sections, they developed an algorithm called *refinement by intervals*. Intermediate contours are generated at regular intervals over the branching section until two separate elliptical sections have been reached.

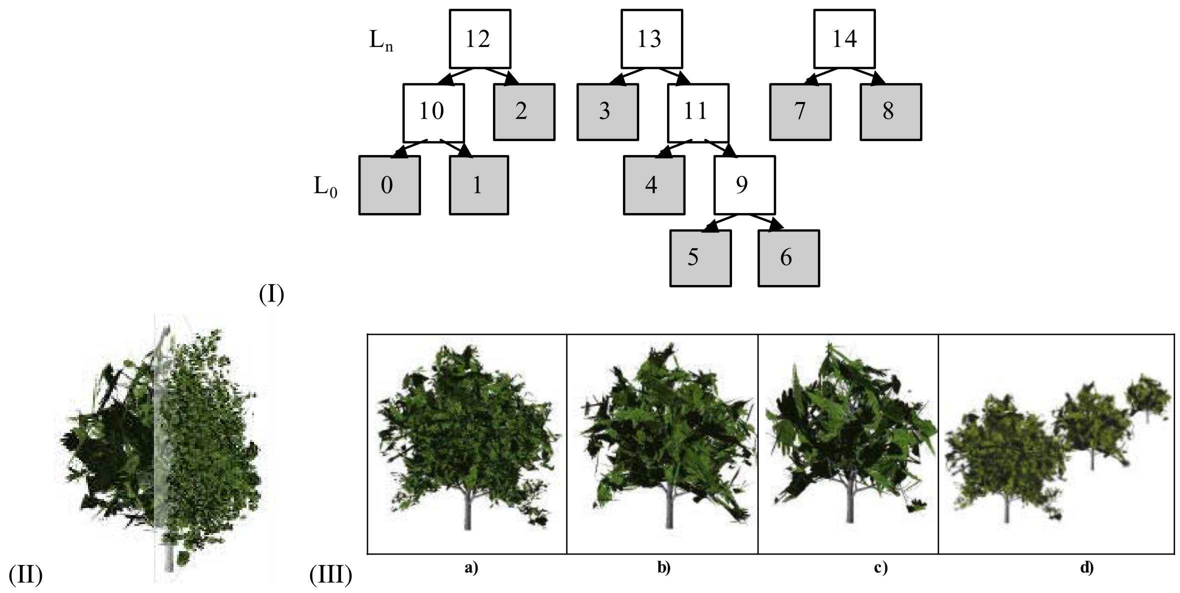


Figure 2.2: In *View-Dependent Multiresolution Model for Foliage* [RCRB03]. (I) Example of data structure used to represent foliage as 'forest' of binary trees. The top level nodes (Numbers 12, 13, 14) represent the lowest detail, while the leaf nodes (grey) contain the highest resolution. (II) View-dependent levels of detail: Interest area is determined by a plane; 18.406 polygons. (III) Different uniform levels of detail of the same tree: (a) 13.420 polygons, (b) 1.558 polygons, (c) 472 polygons. In (d), they are shown depending on the distance to the viewer.

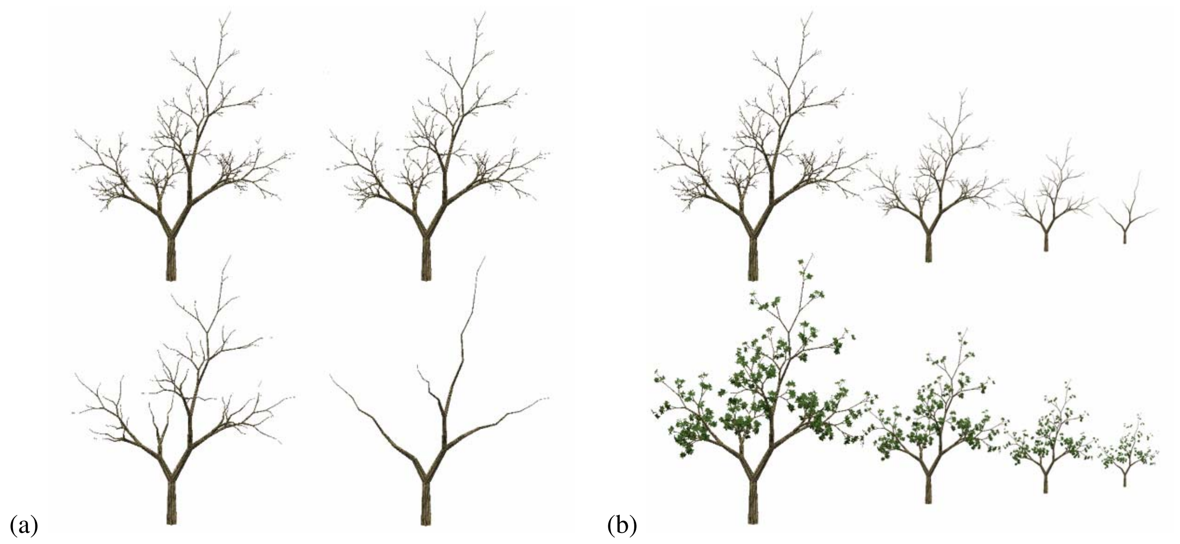


Figure 2.3: In *Procedural Multiresolution for Plant and Tree Rendering* [LCV03], (a) A tree generated at four different levels of detail (3252, 2103, 872 and 172 polygons). (b) The trees from (a), rendered at their respective sizes.

These intermediate contours can then be used to generate appropiate polygonal representations (see figure 2.4). The authors note that even though refinement by intervals causes a significant increase in polygon count, the resulting continuous mesh can easily be reduced by a decimation algorithm.

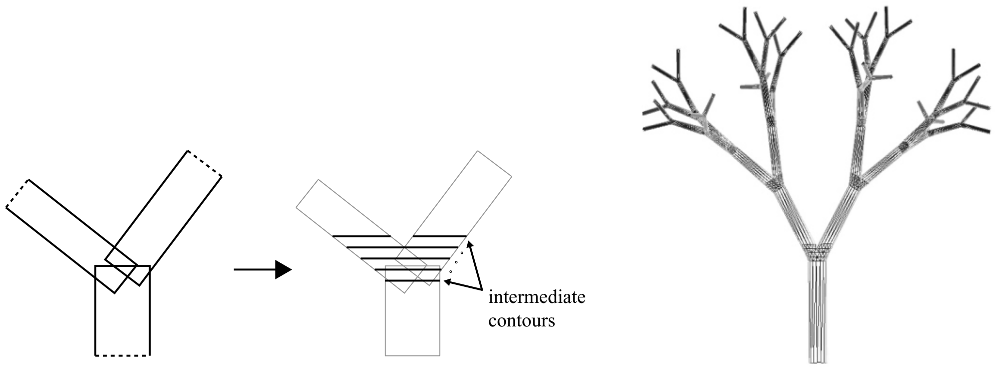


Figure 2.4: In *Modelling of Branched Structures using a Single Polygonal Mesh* [LVF+01]. Intermediate contour generation.

2.1.2 Point based algorithms

In recent years, it has been shown that point-based rendering can be effective for rendering any complex geometry. In the case of interactive natural scenes, these techniques could allow a larger number of rendered trees than triangle-based approximations. According to larger observer distances, the point-based techniques reinterpret the branch meshes as lines, and the leaf polygons are transformed onto points.

William T. Reeves [RB85] in 1985 described a stochastic modelling system that was used to render forest images. Each tree is drawn as a set of particles, line segments and small circles, representing branches and leaves respectively. These particles are generated from a recursive representation in a preprocessing step. To model self-shadowing, a probabilistic model based on the particle's position and orientation has been implemented. External shadows from other trees are also approximated through a probabilistic function (see figure 2.5). At the time of publication, it was clearly not a realtime system (five to ten hours of rendering per image), however the performance of computer systems has increased dramatically in the past two decades, probably leading to interactive frame rates in current graphics hardware.

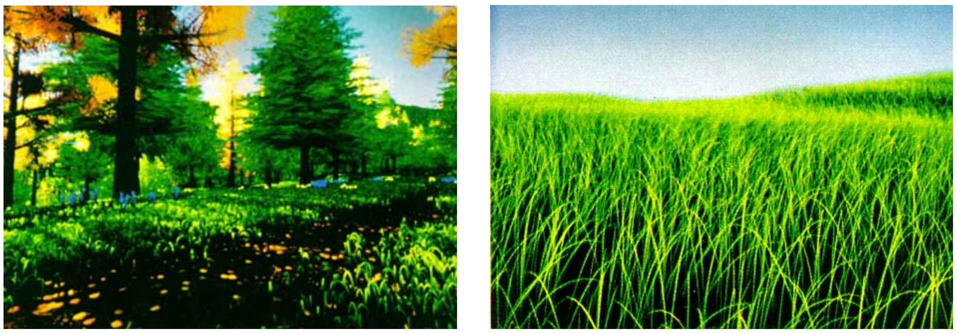


Figure 2.5: In *Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems* [RB85]: (a) Forest Scene from The Adventures of André and Wally B. (b) Still from a Film of Blowing Grass.

Weber and Penn's [WP95] classic paper on modeling and rendering realistic trees also includes point based rendering. Although their publication focuses mainly on the modeling aspect, they make use of point and line primitives for leaves and branches, respectively. The representation created from their model is not explicitily converted to geometry, but interpreted at runtime. At close distances, full-resolution polygonal geometry is created; with progressively increasing ranges stems will be rendered as lines and leaves as points. Heuristic equations are used to transition between these representations (see figure 2.6).

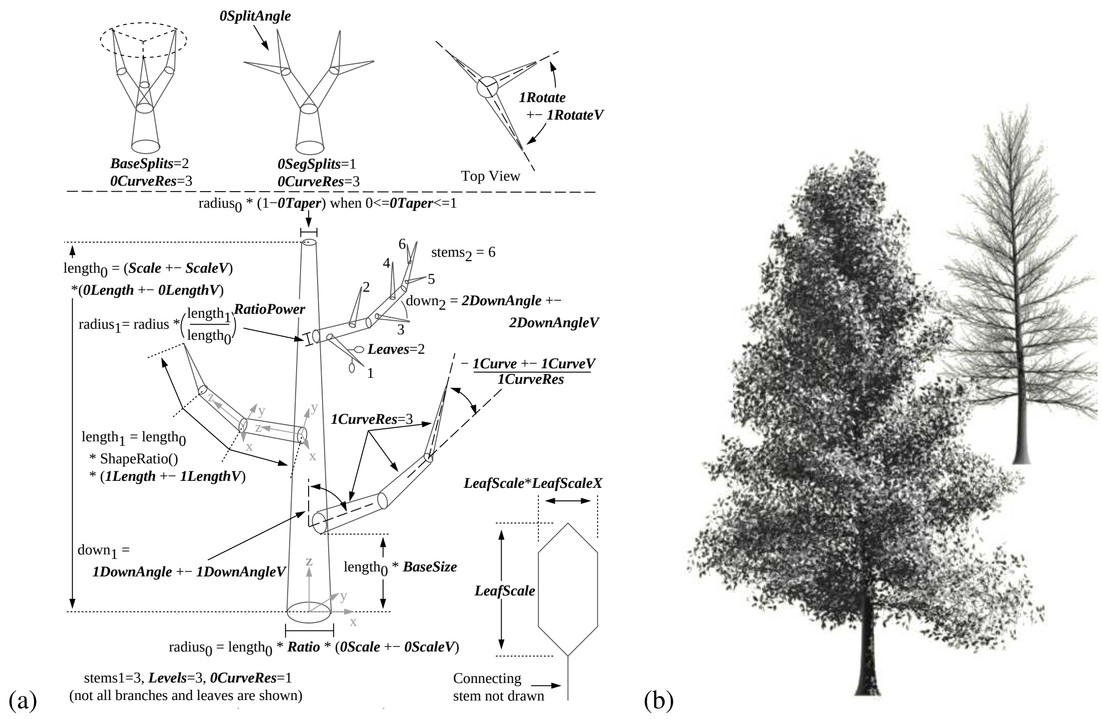

0SplitAngle
BaseSplits=2
0CurveRes=3
0SegSplits=1
0CurveRes=3
1Rotate
+− 1RotateV
Top View
radius0 * (1−0Taper) when 0<=0Taper<=1
length0 = (Scale +− ScaleV)
*(0Length +− 0LengthV)
radius1= radius * (length1/length0)RatioPower
stems2 = 6
down2 = 2DownAngle +− 2DownAngleV
Leaves=2
− 1Curve +− 1CurveV / 1CurveRes
1CurveRes=3
length1 = length0 * ShapeRatio() * (1Length +− 1LengthV)
LeafScale*LeafScaleX
LeafScale
down1 = 1DownAngle +− 1DownAngleV
length0 * BaseSize
radius0 = length0 * Ratio * (0Scale +− 0ScaleV)
stems1=3, Levels=3, 0CurveRes=1
(not all branches and leaves are shown)
Connecting stem not drawn
(a)
(b)

Figure 2.6: In *Creation and Rendering of Realistic Trees* [WP95]. (a) Tree diagram of the defined custom parameters. (b) Black Tupelo example tree generated.

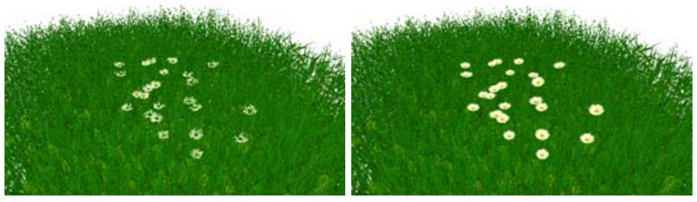
Deussen et al. [DCSD02] have presented a system for interactively rendering large plant populations by using point and line primitives. A hierarchical scene data structure is used to support a coarser representation of distant regions. Additionally, a visual importance factor can be manually assigned to objects, which allows certain objects to be rendered at a higher quality than others. Point and line representations of the polygonal input data is generated semiautomatically. The user needs to choose the primitive to be sed for each part of the plant and possibly assign the importance factor if required. Point and line data is then generated automatically and stored in display lists. Point and line representations and the respective polygonal data are reordered randomly (but in the same order for point primitives and polygons) to avoid popping artefacts. Through the random reordering, switching part of an object from polygonal to point or line representation is note localized to some area of the object, but is distributed over the entire model (see figure 2.7). Rendering point data is performed by estimating the number of points required for a faithful representation (ie. no holes and correct coverage). Blending between polygonal and point data is supported by rendering only part of the polygonal display list, and displaying the remainder as point data. Since both lists are in the same order, the entire model will be covered. For line data, the area covered by the entire line set is calculated and compared to the triangle set it represents. Rendering then proceeds similar to the point data.

The main goal of the previously cited point-based techniques is to decrease the number of rendered primitives without losing visual important characteristics. Gilet et al. [GMN05] uses a triangle set plus a hierarchical point-based LOD representation. They define vegetation blocks that can be composed of a single tree or group of compact trees. These blocks are instantiated on the landscape. The key point is to adapt the precision of each sub-part of a block according to the view dependent position. Each cell block contains a binary tree where each node defines an average point of the sub-tree. These points are size-sorted to be efficiently managed by the GPU. An advantage of point sets is to allow quasi-continuous simplification without high extra costs (see figure 2.8).

2.2 Image-based techniques

Image-based methods represent a trade-off between consistency and physical precision in favor of more photorealistic visuals. The performance of the image-based algorithms tend to be independent of the object complexity and controlled by the output resolution alone. This makes them quite suitable for complex objects such as vegetation.

(I)



(II)

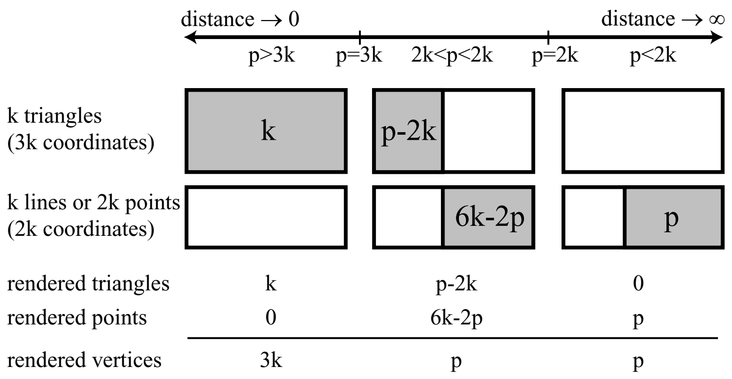
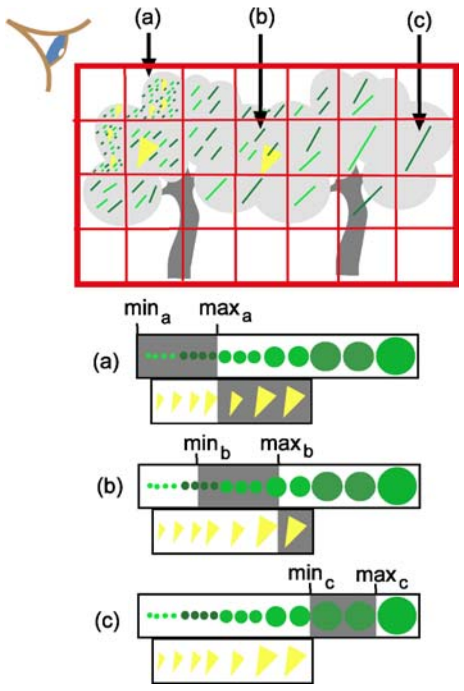


Figure 2.7: In *Interactive Visualization of Complex Plant Ecosystems* [DCSD02]. (I) Assigning importance factors. left: default values. right: setting a higher importance factor to the daisies, causing them to be rendered as polygons. (II) Blending polygonal and point based rendering based on the available rendering budget.

(I)



(II)

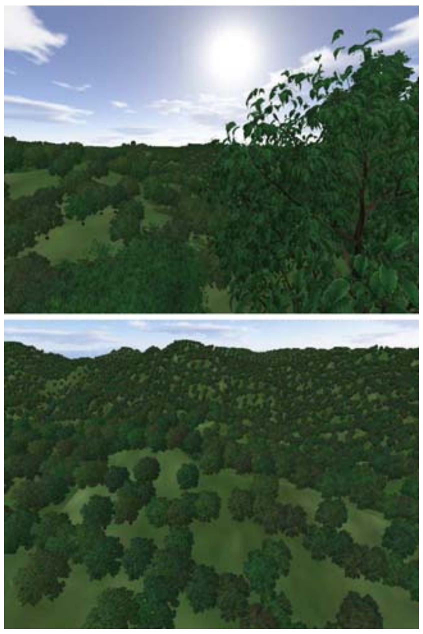


Figure 2.8: In *Point-based rendering of trees* [GMN05]. (I) Two thresholds min and max define in pixel unit the range of acceptance primitives. For a given cell (b), we deduce the range (resp. the lower boundary) of points (resp. triangles) which have to be rendered. Rendered points and triangles have grey background on the arrays. According to the position of the cell in the grid, we shift the min and max thresholds to lower values (a) for detailed cells and to upper values for coarse cells (c). (II) Landscape scene containing 200.000 trees.

An algorithm proposed by Max [MO95] in 1995 uses precomputed z-buffer views to approximate arbitrary viewpoints. Precomputed views are acquired through a simple longitude/latitude sphere partitioning scheme. Since there is little coherence between leaves in a tree, the reconstruction for an arbitrary view point is performed on a per-pixel basis (see figures 2.9(a) and (b)). This typically leaves some pixels undefined where no information can be extracted from the available views. The authors have chosen to implement multiple z-buffer layers to reduce these artifacts. Dynamic shading and shadowing is supported by storing compressed normal vector and material information for each pixel of the precomputed views. During the shading post-process, these values can be used to compute diffuse and phong shading. Shadows can be found by reconstructing a z-buffer view for the light source and testing output pixels against this buffer. Since normal vector and material information is available, shading can be applied in a post-processing step once for each output pixel instead of each time pixel data is written to the output frame buffer. On the other hand, this method requires high amounts of memory to store the z-buffers, normals and material information for each pixel to compute lighting changes, and can only provide acceptable results for far views. In addition cannot support any kind of animation at all and doesn't provide any LOD technique to smooth the transition for closer views.

Max et al. [MDK99] combine a hierarchical tree model with an image based rendering method that supports hardware acceleration. Although their rendering times were not real-time at the time of publication, it may be feasible with current graphics hardware (see figure 2.9 (c)), though as commented earlier, this method have severe restrictions to be effective for real-time large natural scenes at different view scales, specially from medium to close views.

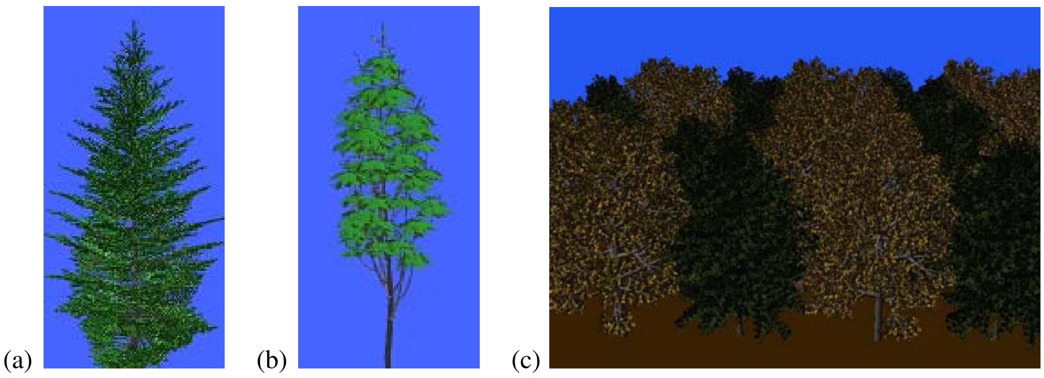


Figure 2.9: In *Rendering Trees from Precomputed Z-Buffer Views*. [MO95] (a) Reconstructed Abies. (b) Reconstructed Magnolia. Hierarchical Image-Based Rendering Using Texture Mapping Hardware [MDK99]. (c) A long view of a whole forest.

Their approach precomputes multi-layered depth images containing color and normal information using standard z-buffer hardware. Six orthogonal views are calculated for each level in the hierarchy. Multiple depth layers are computed by using hardware z clipping to partition the object into several slabs. To avoid excessive number of textures, the number of not equivalent subobjects in the hierarchy must be limited. Based on the object distance from the viewpoint, the hierarchical description is traversed until either the current level is a sufficient approximation, or the actual polygons need to be generated. During rendering, the hierarchy is first traversed and a list of reprojection matrices accumulated for each texture. All visible instances of a texture are then rendered in order, thus significantly reducting texture swapping.

Jakulin [Jak00] combines traditional polygonal geometry for the trunk and limbs of a tree with an image-based rendering system for the crown. The crown is rendered using multiple parallel layers (*slices*). The group of slices for a specific view direction is called a *slicing*. During preprocessing, several sets of these slices are created from various view points. For each slicing, the primitives (ie. individual leaves) are assigned to the closest slice (see figure 2.10). Each slice is then rendered to an individual texture. During rendering, the two slicings that are closest to the actual view direction are rendered simultaneously with correct transparency and blending. The goal of this algorithm was to accomodate architectural walkthroughs and driving simulations, so viewing trees directly from above or below the tree is not supported. However, this is not an inherent limitation of the method, and the author speculate that blending three slicings would be appropiate for arbitrary viewpoints. On the other hand, the alpha blending between the two closest slices in certain view points leads to a ghost-like appearence of the foliage when the observer moves from medium to close views.

Therefore, all slices are perpendicular to the ground plane, and blending between two sets provides sufficient coverage.

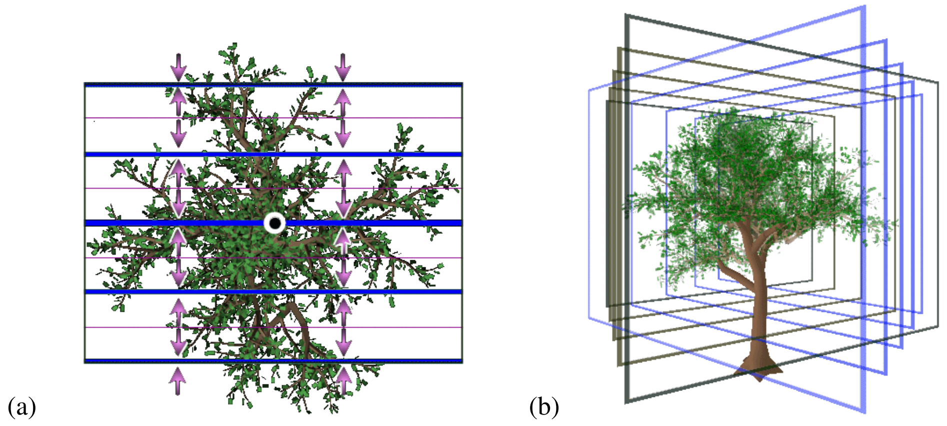


Figure 2.10: In *Interactive Vegetation Rendering with Slicing and Blending* [Jak00]. (a) Each primitive of the crown is assigned to the closest slice. (b) Multiple slicings are blended to create a solid-looking rendering. Rectangular frames have been added to slice textures to aid visualization. Both slicings have a discrepancy of about 30 degrees.

Lluch et al. [LCV04] continued their work by introducing an image-based rendering approach for rendering foliage. Based on an L-system tree model, they create a hierarchical data structure that includes bounding boxes at each level. All leaves within the bounding box are then projected to each of the bounding box planes, and stored as impostor textures. To increase visual realism, impostors are not only created for the usual $x = \pm 1$, $y = \pm 1$ and $z = \pm 1$ planes, but also for the main diagonals ($x \pm y = 0$ etc.). The bounding box, and therefore also its impostors, are oriented in the local coordinate system for each level. Not all possible levels of recursion are visited during impostor creation. To reduce the spatial cost of the model representation, a threshold based on the relative size of the branch (compared to the entire tree) can be set. At run time, the hierarchy is traversed until a suitable (distance based) level of detail is reached. Appropiate viewing distances for each level of the hierarchy are precomputed for better performance. If no suitable impostor available, the original (polygonal) geometry is used (see figure 2.11). This method presents a LOD technique, however the change from the fully polygonal representation to the bounding boxes must be done too further away from the observer, falling to only interactive frame rates, in order to diminish the parallax problems appearing when switching the representation.

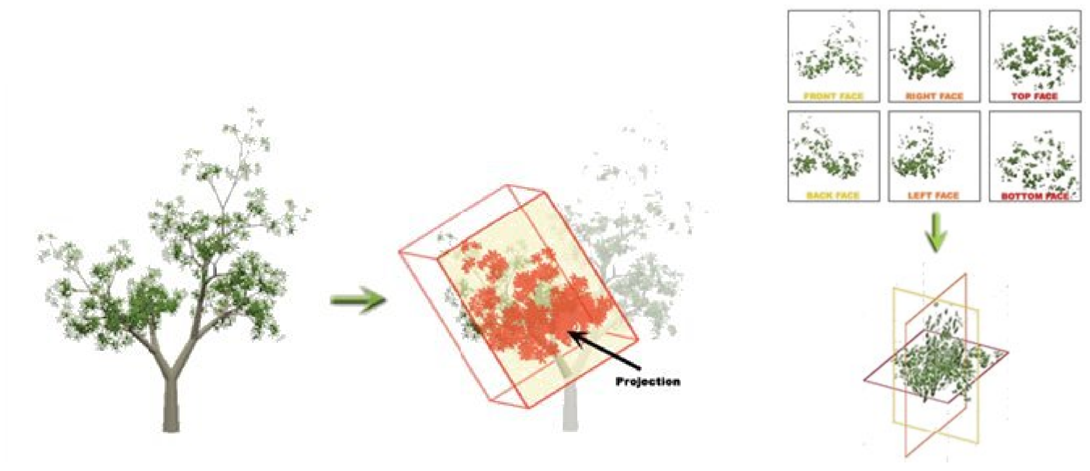
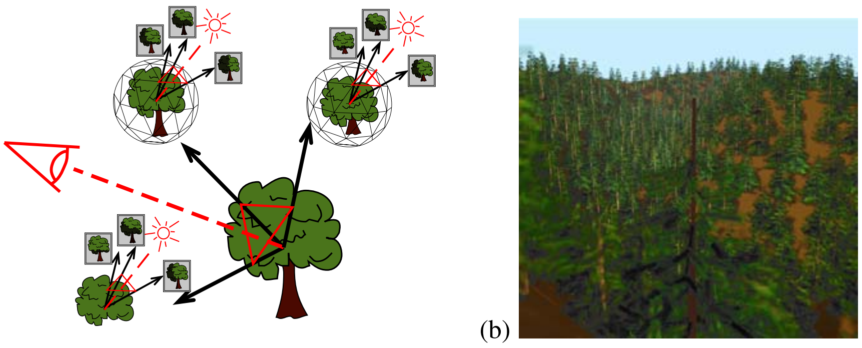


Figure 2.11: In *An Image-Based Multiresolution Model for Interactive Foliage Rendering* [LCV04]. Pregenerated textures replace tree leaves, being associated to a hierarchy of bounding boxes.

The image-based rendering system proposed by Meyer et al. [MN98] provides a framework for rendering trees with complex effects such as shading, self shadowing, and dynamic illumination. They combine a hierarchy of bidirectional textures (HBT) to provide billboards for each given observer and light directions with a hierarchical visibility structure for self-shadows and cast shadows. This representation is efficient for trees, as it is hierarchical display lists calls are heavily used (see figure 2.12).



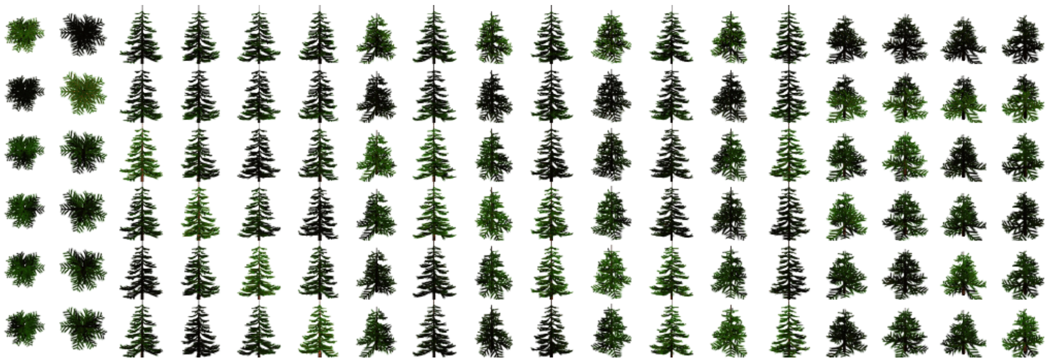


Figure 2.12: In *Interactive Rendering of Trees with Shading and Shadows* [MNP01]. (a) The complete BTF allows the reconstruction of a billboard for given view and light directions by combining up to 9 stored images (in their implementation). (b) views in a forest (1000 trees on a landscape, with shading, shadows, and fog). (c) The BTF associated to the highest level (horizontal axis: 18 view directions, vertical axis: 6 light directions, 64×64 resolution per billboard, with colors and transparencies).

Bidirectional texture functions (BTFs) are computed by associating a billboard representation with each pair of view and light directions, similar to a BRDF (see figure 2.12). Between 6 and 258 different view directions can be approximated by interpolating 9 BTFs. These BTFs are associated to each level in the hierarchy either by creating a new, unique BTF of through instancing. During rendering, either BTF or the actual geometry is rendered depending on the distance. To support dynamic lighting effects, approximate visibility cube-maps are computed for each level of the hierarchy. Since occlusion depends on the position within the hierarchy, separate cube-maps need to be generated for all instances. Shadowing can then be computed during rendering by traversing the hierarchy of visibility cubemaps. Casting shadows is supported through 'traditional' shadow maps by rendering from the light source. The use of different billboards for each observer and light direction makes animation not feasible at all, and the method is only addressed to far views of large real-time forest.

On the other hand, Neyret converts complex natural objects into volumetric textures, which are then raytraced [Ney98]. Mip-mapping is used to reduce texture data, and to generate samples from trees in the distance. The method provides high quality images of very complex scenes at moderate rendering times of several minutes, but it is not applicable in interactive applications. More recently, this technique was adapted to graphics hardware by Decaudin and Neyret [DN04] (see figure 2.13). The results are very effective for far views of large natural scenes but lacks from a LOD technique to deal with a smooth transition in closer views.

Reche et al. [RMMD04] capture real trees from a small number of calibrated photographs of a tree and use volumetric texture representation. The method is based on estimated opacity in a volume, generating and displaying view-dependent textures attached to cells of that volume. They achieved interactive frame rates but at the expense of high memory cost (see figure 2.14). Later, the same authors presented a texture compression and a multi-resolution approach to reduce this memory cost [LRMDM06]. However, their model is not well suit for dynamic lighting or close views.

To represent tree models using billboards, the "billboard clouds" approach, introduced by Decoret et al. [DDSD03], can be used. It represents a geometry by a set of arbitrarily oriented billboards. However, for tree

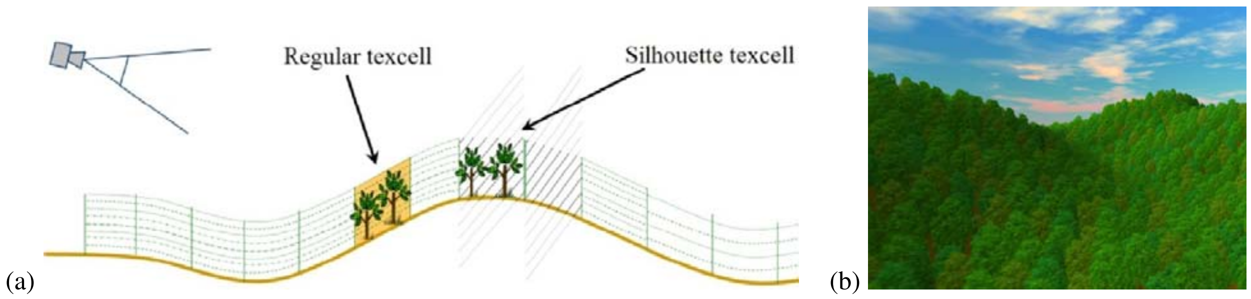


Figure 2.13: In *Rendering Forest Scenes in Real-Time* [DN04]. (a) Slicing forest scheme, where each tree is represented with a set of textures (texcell) facing the camera. (b) Scene with 9212 tiles, 37000 trees in real-time.

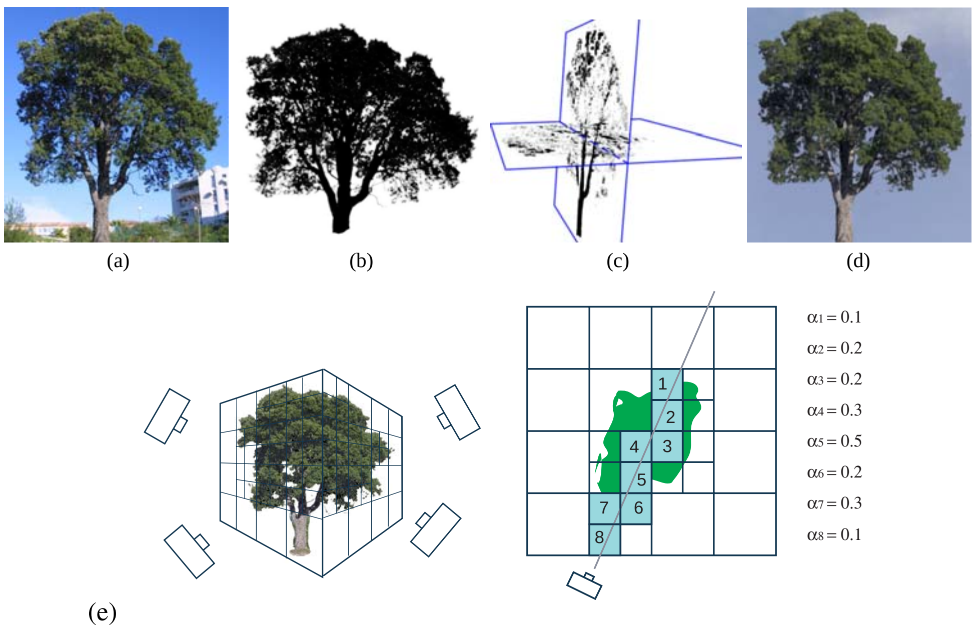


Figure 2.14: In *Volumetric Reconstruction and Interactive Rendering of Trees from Photographs* [RMMD04]. (a) One of the original photographs of an oak. (b) The α mask used for the opacity estimation. Two cross slices of the resulting opacity are shown in (c). A synthetic image of the original view, using our view-dependent rendering, is shown in (d). Textures are attached to billboards in cells of the volume and are generated based on estimated opacity. (e.left) A recursive grid is placed around the tree which is surrounded by the cameras. (e.right) For a large number of pixels in these photographs, each pixel is semitransparent, and we estimate an alpha value corresponding to the cumulative opacity. A ray emanating at pixel p traverses the grid; a set of cells are collected to estimate the opacity of the volume.

models the proposed algorithm does not work optimally, since in this case the used plane-space transform does not provide significant results due to the non-compact geometry of the tree models. Fuhrmann et al. [FUM05] introduce a number of adaptations to improve the quality of the simplification, especially for tree models (see figure 2.15). They use an extended algorithm to generate image-based representations of arbitrarily complex tree models, providing satisfactory billboard cloud representations including leaves and branches at expense of being quite memory and CPU demanding with possible numerical stability problems. In addition, they use super-sampling in an off-screen buffer, and then downsampling with respect to a user-defined texture size, trying to preserve the small details of the leaves, but they are not effective enough for close views.

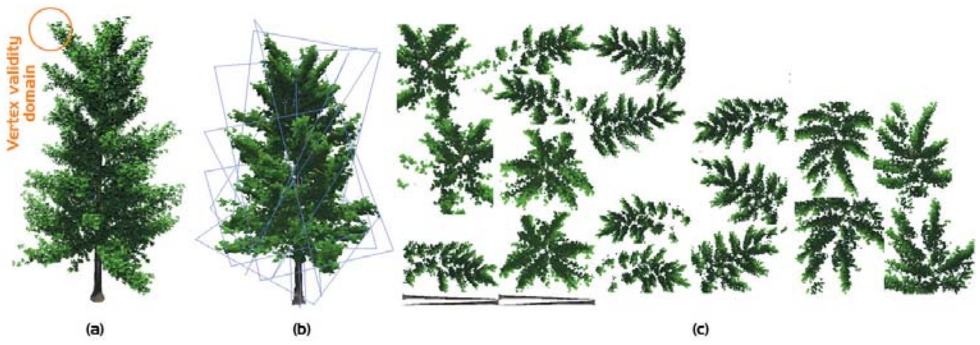


Figure 2.15: In *Extreme Model Simplification for Forest Rendering* [FUM05]. A 110.000 polygons tree model (a) simplified to 11 billboards. (b) The error threshold is shown by the vertex validity domain. Billboard textures are depicted in (c).

Behrendt et al. [BCF$^+$05] use a modified idea of Decoret et al. [DDSD03] and applied a clustering algorithm directly to the vertices of the triangles that constitute the tree model. Each of the clusters is then represented by one or more billboards in dependency of the form of the respective point cloud. These billboard cloud representations are able cope with leaves and branches, requiring the triangles of the geometric description in a hierarchical form. All triangles that belong to one branch including its sub-branches are stored in the file. On the other hand, they use shell textures to represent larger parts of the scene in the background, because, although the billboard cloud approximation of individual plant models is simple, rendering of massive scenes remains difficult. The main idea is to approximate the geometry of the vegetation with a 3D texture rather than directly displaying it, as in [DN04]. As a result the performance is independent of the geometric complexity of the vegetation. In addition they introduce a lighting model using a special representation for the reflection function with spherical harmonics. In a pre-processing step a set of coefficients for spherical harmonic lighting is computed from the original object. During runtime the evaluation of the coefficients gives the lighting environment. One of the additional problems they addressed is caused by blending the billboard representation and the polygonal model. The shell textures for massive plant representation can be further than e 20m - 100m from the virutal camera. However, the full geometric complexity must be used in the direct vicinity of the obsever up to a distance of about 5m - 20m relative to the size of the plant. In this range of distances there can be hundreds of thousands of triangles in a very dense outdoor environment (see figure 2.16). In order to avoid having low interactive frame rates, they switch to the billboard representations at very close distances, which cause parallax problems.

SpeedTree's models [SPE05], are an interesting blend: The stem is composed of polygon meshes with texture and bump maps applied, while the foliage is represented by several view-aligned billboards instead of static billboards like it is the case with the bulk of low-polygon models. This trick allows for good looking models with low polygon counts. However, the billboards become strikingly obvious when looking at trees from above or below and rotating the view, in which case the foliage rotates and partly obscures each other, as shown in figure 2.17. Since the described motion is rather common in real-time applications, such as videogames, this behavior is a big problem. On the other hand, levels of detail are employed for the trees that blend quite well and on the lowest level of detail, a tree is replaced with a single billboard.

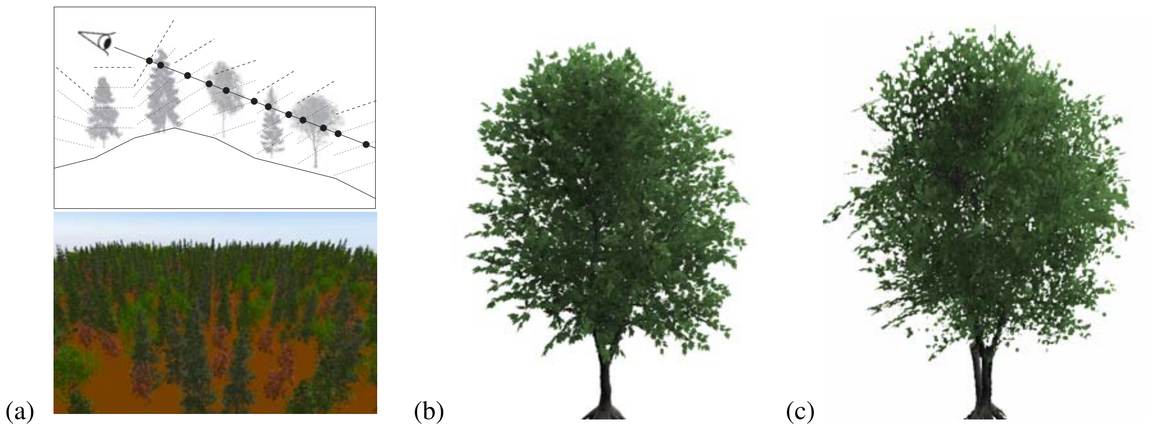


Figure 2.16: In *Realistic real-time rendering of landscapes using billboard clouds* [BCF+05]. (a) Slanting slices solve the problem in case the viewing direction is parallel to the slice geometry and there is no sample at all on the viewing ray. (b) Polygonal tree with 126.000 triangles. (c) Approximation with 60 billboards.

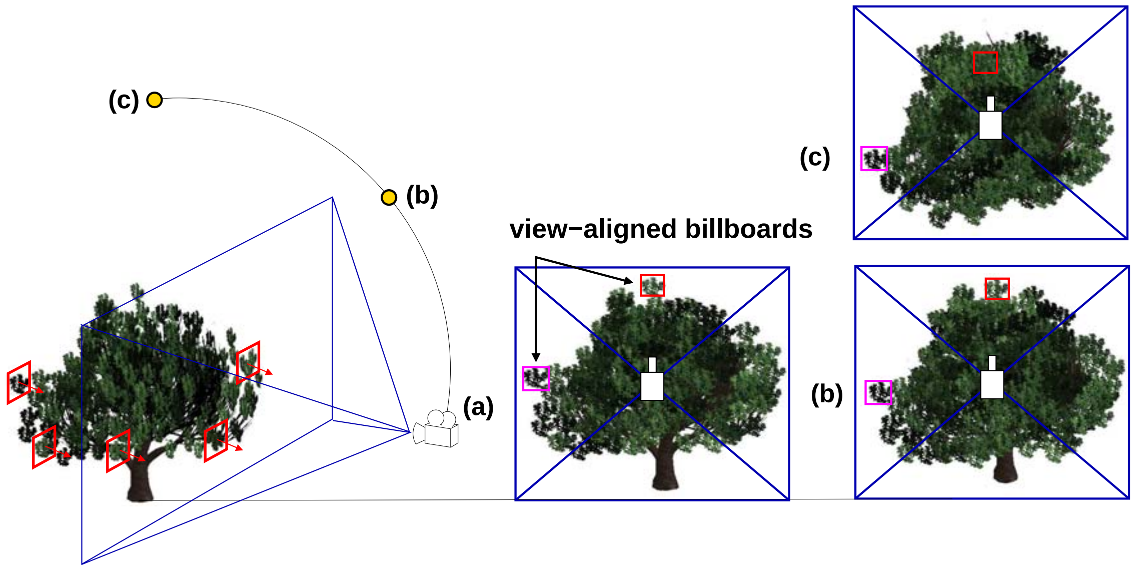


Figure 2.17: In *SpeedTree* [SPE05] represents the foliage is represented by several view-aligned billboards instead of static billboards. However, the billboards become obvious when rotating the view, in which case the foliage rotates following the camera.

2.3 Vegetation lighting models

The lighting is an important in component tree rendering techniques. In addition to the geometric complexity of plants, the complex light interactions between the foliage and the branches, make the realistic lighting in plants very challenging.

The tree rendering benefit from the new generic lighting models appearing so far. But, in addition, a number of techniques have been proposed to calculate the light interactions specifically for the leaves.

Most of the works commented here take into account indirect lighting effects, such as the ambient light term, or including subsurface scattering. Some of the lighting proposals of several authors exceed in computational costs for the real-time forest scenes requirements we established for our current work. Nevertheless, we found interesting to know some details of these models, in order to be able to evaluate how can we integrate or even improve them.

The ambient occlusion term introced by Landis [Lan02] is a shading method which helps to add realism in local reflection models by taking into account attenuation of light due to occlusion. Unlike local methods like the phong shading, ambient occlusion is a global method, meaning that the illumination at each point is a function of other geometry in the scene. However it is a very crude approximation to full global illumination at the expense of a smaller computational cost. More recently, appeared specific ambient occlusion methods for plants and trees. We refer the user to chapter 5 to know more about how we defined the ambient occlusion term in our model.

Hegeman [HPAD06] presented an approximate ambient occlusion term interactive computation for trees. Trees are approximated to spheres or ellipsoids and the ambient occlusion formula is computed considering this shape as if it were full of blocking elements (see figure 2.18). The result is that the more inner an element (leave) is, the darker it appears. It considers skydome ambient occlusion (normal always up). Inter-object occlusion (between trees, or between trees and floor) is done using the solid angle approximation.

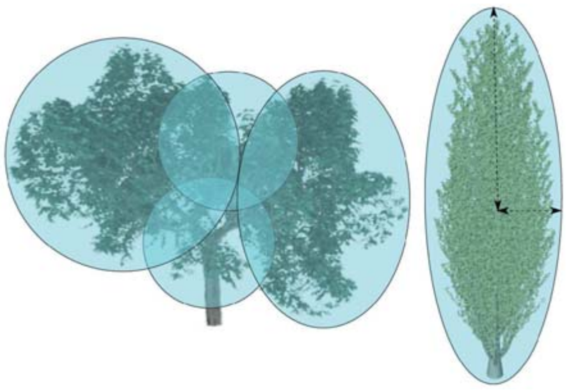


Figure 2.18: In *Approximate Ambient Occlusion For Trees* [HPAD06]. Trees are approximated with ellipsoids and an adapted formula to compute ambient occlusion for ellipsoids filled with blocking elements is used.

Wang et al. [WWD^{+}05] introduced a number of illumination effects into a real-time rendering algorithm for leaves. Their algorithm handles reflectance and translucency, and precomputes the radiance transfer in a whole tree both for indirect illumination and direct sunlight illumination. However, their translucency model is basically Lambertian, and ignores all multiple scattering effects that occur due to changing properties along the leaf surface (see figure 2.19(a)).

Habel et al. [HKW07] proposed a more accurate real-time translucent subsurface scattering model for plant leaves. Their method works for directly lit leaves. They formulate the model through an image convolution process and express the result in an efficient directional basis that is fast to evaluate (see figure 2.19(b))

(a) (b)

Figure 2.19: (a) In *Real-Time Rendering of Plant Leaves* [WWD+05]. Leaves of balata plant. (b) In *Physically Based Real-Time Translucency for Leaves* [HKW07]. A tree featuring physically based translucency.

2.4 Procedural image-based methods

Textures are useful for adding visual detail to geometry, but they don't work as well when extended to cover large areas such as a field of flowers, or many other similar objects. Such uses require either a very large amount of texture data or repetition of the same pattern, resulting in an undesirable, regular look. The new capabilites of programmable graphics hardware can manage the creation of procedural textures in run-time with low memory requirements.

Glanville [Gla05] developed a procedural technique that places small images at irregular intervals by dividing the UV space into a regular grid of cells, and then placing an image within each cell at a random location, using a noise or pseudo-random number function. The final result is the composite of these images over a background. However, they can only build procedural textures with limited overlapping in the range of the cell size (see figure 2.20).

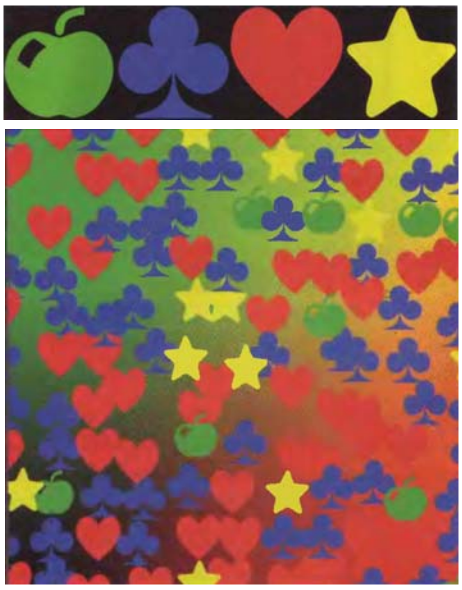
(a) (b) 

Figure 2.20: In *Texture Bombing* [Gla05]. (a) Four sample images stored in a single texture map. (b) Procedural image generated with the sample images.

Lefebvre et al. [LHN05] developed a method to splat small texture sprites onto a surface to define a composite texture. The sprites can be arbitrarily blended to create complex surface appearances. Their attributes (position, size, texture id) can be dynamically updated, thus providing a convenient framework for interactive editing and animated textures. Each sprite is described by a small set of attributes which is stored in a hierarchical structure surrounding the object's surface. Their method is quite flexible but requires high memory cost because of the use of volumetric textures to encode the information (see figure 2.21).

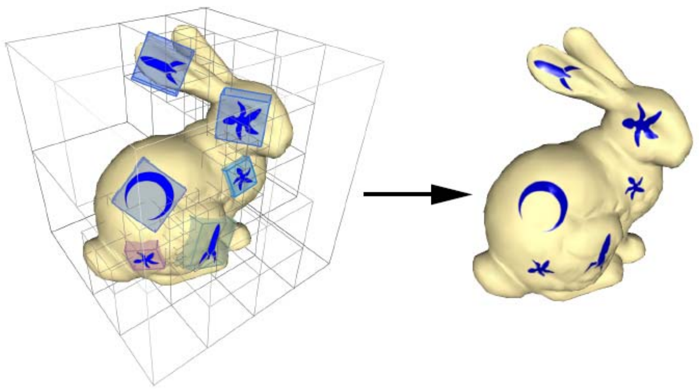

(a)

Figure 2.21: In *Texture Sprites: Texture Elements Splatted on Surfaces* [LHN05]. (Left) The attributes of the sprites are stored in an octree structure surrounding the mesh surface. (Right) image shows a model with lapped textures.

2.5 Conclusions

Table 2.22 summarizes the presented algorithms. It is certainly not possible to capture the intrincate details of each method in one simple table, and the individual algorithms should be studied in detail before drawing any conclusions. Additionally, factors such as quality and performance are difficult to judge. Methods that did not run in real-time a few years ago may be feasible with current hardware, and algorithms that were formerly limited eg. by texture memory size can now run with significantly higher quality.

The majority of algorithms are based on polygonal rendering, presumably because triangles are the best supported primitive in current graphics hardware. Image based methods have emerged recently, because they benefit from the increased texture performance and the standard use of texture mapped polygons for rendering as well. Point based systems have had some exposure since the early beginnings of realtime rendering, but were never as widely accepted and used as polygonal approaches. However, especially Deussen's work shows that they should not be completely ignored [DCSD02].

In the study of the previous work, we have done a classification of all the presented methods. In table table 2.22 we describe the following list of the evaluated characteristics:

- **Branches:** Are branches supported by this algorithm, or do they need to be rendered separately?
- **Leaves:** Are leaves supported by this algorithm, or do they need to be rendered separately?
- **View dependent LOD:** Is there support for view dependent (eg. distance based) levels of detail?
- **Animation:** Does the algorithm support dynamic scenes, eg. movement of branches due to wind?
- **Dynamic lighitng:** Does the algorithm support dynamic lighting? (if not, it is precalculated)
- **Quality:** How good is the image quality?
- **Performance:** How fast is the algorithm?

In addition, table 2.22 checkmarks indicate that a certain feature is explicitly supported by the algorithm. No checkmark could be either 'omitted in the current state of development' (but generally feasible) or 'conflicts with the intentions of this method' (because, for example, a single polygonal branch mesh naturally won't support leaves). For the performance and quality ratings, an empty circle represents the lowest rating, and full circles the highest. Of course these are entirely subjective, but we feel that they still provide a good overview of how these algorithms perform.

Ideally, we would want an algorithm that supports all the features in table 2.22, at a very high quality and in realtime. Most current algorithms include some features at the cost of others.

For example, approaches that only support branches or foliage will be difficult to integrate if realtime animation is desired. The same holds true for image based approaches, since they typically involve extensive preprocessing and offline generation of textures which may need to be adjusted for dynamic lighting or animation. A notable exception is [MN98], which does support dynamic lighting at the expense of increased memory demands and rendering overhead.

In contrast to rendering methods for other types of scenes, where graphics hardware fill-rate or bandwith limitations are usually the performance bottleneck, many of the more realistic methods described above had CPU overheads. Obviously this did not apply to all algorithms, but the ratio of CPU limited approaches was considerably higher than in other areas. However, this fact is changing since the programmable graphics hardware (GPUs) came out few years ago. The current general trend in computer graphics, and in the interactive rendering of natural scenes as well, is the development of techniques that passes, as much as possible, all the rendering computations directly to the GPU.

Once again looking at table 2.22, it is apparent that realtime animation is not easily implemented in most state of the art algorithms. However, "real" trees are hardly ever perfectly still since even a very light breeze will cause individual leaves or small twigs to move.

In addition, rendering methods that support very high detail are problematic: Image based systems would need very large amounts of texture memory for the required resolution. The only feasible solution could be the integration of a procedural model which avoids the explicit generation of geometry (or other data) unless necessary. Marshall's multiresolution rendering approach does go in this direction [MFI97], but with the restriction of supporting static scenes only. A point based approach could support animation and still allow arbitrarily large scenes. However, point based systems suffer quickly from bandwidth limitations when large areas need to be covered.

This method would therefore benefit from impostors in the same way that polygonal geometry does. Such a render cache would ideally be applicable at any level of the scene hierarchy, from branches to groups of trees.

Therefore, our work focus was the development of an image-based method based on billboards aiming to have highly detailed representations but without the huge memory requirements of the previously presented image-based methods. In order to be able to deal with these requirements we studied the procedural texturing methods presented in section 2.3 to drastically reduce the memory cost. In addition, we wanted to bring a texture-based model capable of handling dynamic realistic lighting effects, such as the presented in section 2.4. We have also referenced our current work [GSSK05b, GSSK05a, GPSSK07] in table 2.22, just to provide an overview of the features we have taken into account. We refer to the appendix A to review our recent published work we are focusing in this document.

Another idea could be to defer the generation of impostors until needed; this would reduce the memory footprint and also allow dynamic updates. Obviously this is best used with a procedural tree model that supports the on-demand generation of geometry for those parts that need to be re-rendered. But this and other proposals will be briefly studied as a future work in chapter 7.

	Algorithm	branches	leaves	view dependent LOD	animation	dynamic lighting	instancing	quality	performance
Polygonal	Fractal Plants and Trees [Opp86]	✓			✓	✓	✓	○	●
	Multiresolution Botanical Scenes [MFATC97]	✓	✓	✓		✓		◐	●
	Single Polygonal Mesh [LVF+01]	✓				✓		●	○
	Multiresolution Foliage [RCRB03]	✓		✓		✓		○	●
	Procedural Multiresolution [LCV03]		✓	✓		✓	✓	◐	●
Point Based	Structured Particle Systems [RB85]	✓	✓			✓		○	○
	Rendering of Realistic Trees [WP95]	✓	✓	✓			✓	◐	◐
	Interactive Plant Ecosystem [DCSD02]	✓	✓	✓			✓	●	○
	Point-based trees [GMN05]							◐	●
Image Based	Precomputed Z-Buffer Views [MO95]	✓	✓			✓	✓	◐	◐
	Hierarchical IBR [MDK99]	✓	✓	✓			✓	◐	○
	Shading and Shadowing [MN98]	✓	✓	✓		✓		●	○
	Slicing and Blending [Jak00]		✓				✓	○	●
	Image-Based Foliage Rendering [LCV04]		✓	✓			✓	◐	◐
	Volumetric forest scenes [DN04]	✓	✓	✓		✓		◐	●
	Volumetric reconstruction of trees [RMMD04]	✓	✓	✓				◐	○
	Extreme simplification for forests [FUM05]	✓	✓	✓		✓	✓	◐	●
	Landscapes billboard clouds [BCOJD05]	✓		✓		✓		◐	●
	Leaf Cluster Impostors for Trees [GSSK05]		✓	✓		✓	✓	◐	●
	Multi-layered Indirect Tex. Trees [GPSSK07]		✓	✓		✓	✓	●	●

Figure 2.22: Summary of the presented algorithms. Checkmarks indicate availabe features; Circles represent low (empty), medium (half filled) and high (filled) quality/performance.

Chapter 3

Extremal geometric vegetation simplification

In this chapter we present a compact foliage representation, which is specially suited for the interactive visualization of large natural scenes. As we introduced in chapter 1, the geometric representations of plants and trees can cope with realistic descriptions of real species. However, they exceed the hardware capabilites to compose scenes with thousands of trees. Trees are commonly represented divided with two distinct parts: trunk with branches, and leaves. The trunk is usually handled by using a general geometric simplification method. For instance, a multiresolution method can be a good solution for interactive environments, as several authors have noticed in the literature [Hop96, Rem05]. On the other hand, the leaf canopy of trees contributes a huge number of polygons, and traditional geometry simplification methods, such as [GH97], are typically not applicable because leaves consist of many individual polygons. Therefore, as suggested by Remolar [Rem05], topology preserving algorithms will not succeed, and non-preserving methods will typically introduce a significant change to the overall appearence (see figure 3.1).

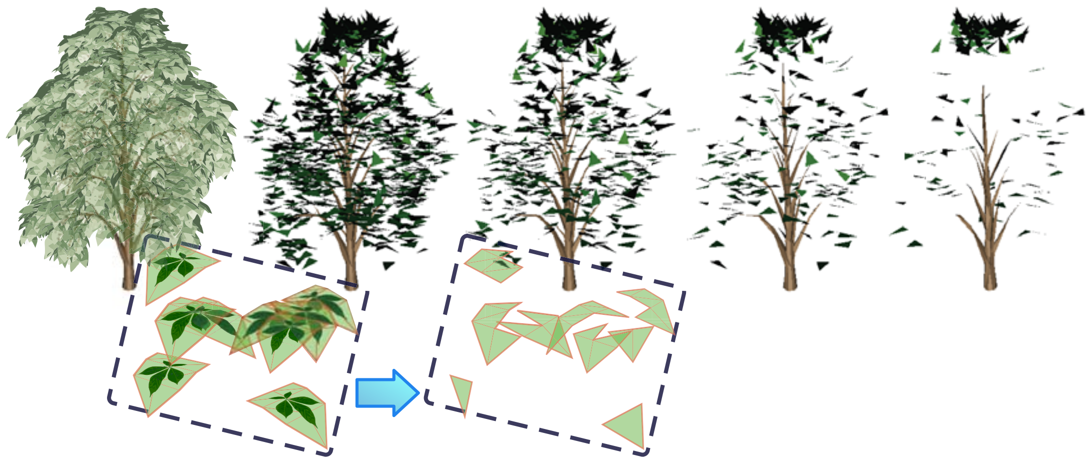


Figure 3.1: Example of geometry simplification using Garland's et al. [GH97] Quadric Error Metric method. This chestnut tree have been simplified in different multiresolution meshes. As can be seen, the trunk and branches are well preserved, compared to the original model. But the leaves are prunned drastically in each level losing their original dense appearence.

Although the triangle primitives are the fastest with current graphics hardware, the geometric descriptions are not the best solution to describe certain kinds of objects such as the foliage of plants and trees in interactive applications, because they can have hundred of thousands of triangles in a few squared meters of a very dense outdoor environment. This situation can only be circumvented switching to simpler representations even in close to medium observer distances.

We have observed that image-based methods can provide a good performance independently of the object complexity and mainly controlled by the desired output resolution. Consequently, these conditions make image-based techniques quite suitable for complex objects such as vegetation. For instance, some previous works use volumetric textures for real-time rendering of vegetation, such as [DN04, RMMD04, LRMDM06], and an increasing number of image-based techinques appeared in the last years (see in chapter 2). However, the main disadvantage of such methods are their high texture memory requirements and that they are only effective to represent objects far from the observer.

Therefore, the main purpose in our research is finding a simple foliage representation that can provide convincing results even from close views, at low computational cost, and avoiding to switch to the complex geometry in such conditions. Under this requeriments, we have choosen to develop a billboard-based method with 2D textures to represent the leaves details.

3.1 Billboard cloud algorithm

Billboards is one of the simplest IBR (Image-Based Rendering) approaches. In the literature, several kinds of billboards have been designed, from sets of quadrilaterals rotated to be always facing with the camera [RH94, SPE05], to fixed aligned layers of quadrilaterals [Jak00]. However, the main drawback of these methods is the lack of parallax effect.

On the other hand, the billboard approach we are introducing can find nearly optimal sets of billboards for an effective simplification, replacing sets of polygonal leaves by simple quadrilaterals while maintaining parallax effects [GSSK05b]. To see a comparison of our proposed method with respect to the classical geometry, and other image-based methods, such as [FUM05], we refer the reader to chapter 6.

3.1.1 Leaves geometry preprocessing

Given that the leaf canopy of trees is stored by the vast majority of geometric vegetation generation tools as a triangle soup, without specific information about which subsets of triangles represent each single leaf, we had to introduce a connectivity-check preprocess to find those disjoint triangle subsets of each leaf, as seen in figure 3.2.

In our current implementation, the polygon connectivity constrain is performed with a global strategy searching through all triangle soup. However, this preprocess can be further optimized using partitioning the three dimensional space with an octree and recursively searching through the different levels.

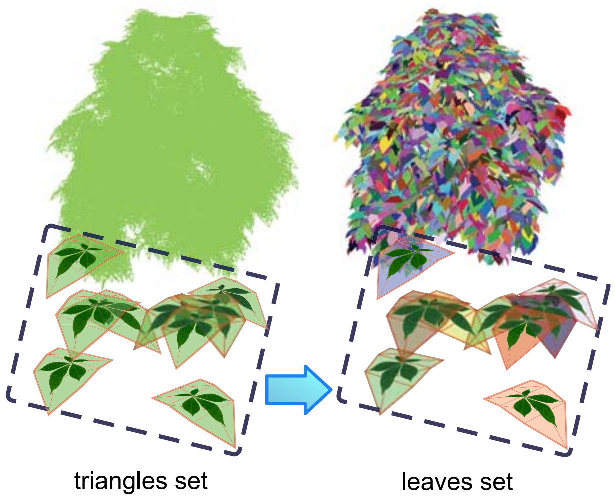


Figure 3.2: A polygon connectivity check is performed to find the leaf components in the triangle soup.

3.1.2 Clustering

In order to reduce the geometric complexity of a tree, our work is based in an extremal simplification method with automatic detection of leaf groups that lay approximatelly on a plane. Each plane will represent a billboard, defined as a quadrilateral having a semi-transparent leaves texture. Since the billboards are not rotated when the camera changes, the expected parallax effects of the classic geometry are provided, as in Decoret et al. [DDSD03].

One of the key points of the method is the generation of these billboards. We should first observe that a 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 it and is roughly parallel to the plane, then the billboard representation results in an acceptable error. Based on this recognition, we form clusters of the leaves in a way such that all leaves belonging to a cluster lie approximately on the same billboard plane and we can replace the whole group by a single billboard (see figure 3.3).

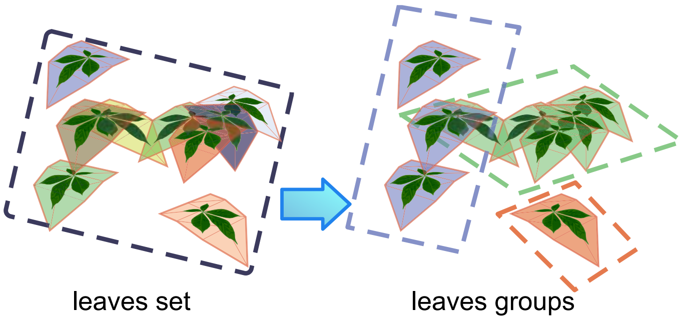


Figure 3.3: The clustering strategy finds groups of leaves that are approxyimatelly around polygon planes.

In our first experiments, we started using two strategies a standard Kd-Tree algorithm and a K-means clustering:

The Kd-Tree is a direct method that starting from the initial set of all the leaves, group them recursively subdividing a group in 2 new ones. The recursivity is controled by the minimisation of an error measure defined between the leaf and the planes. 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 between the plane and leaf normals. However, with the Kd-Tree algorithm we obtained a suboptimal set of billboards, with clusters with approximately the same number of leaves, because the decision taken at the level $i-1$ directly affects the next level, and there is no way to correct them like in iterative methods. These problems motivated us to search a more general iterative clustering method, such as K-means [Mac67].

In our case, the points to cluster are defined by each one of the leaves of the tree. Each leaf is defined by its central point p_i. The centroids of the standard algorithm of K-Means will be the billboard planes. The algorithm starts with a predefined number of random planes, and iterates the following steps:

```
For each iLeaf
  //Cluster each leaf into one of the groups defined by planes
  For each iPlane
    planesErrors = computeErrorMeasureLeafPlane(leaves[iLeaf], clusterPlanes[iPlane])
  End For
  p = findPlaneWithMinError(planesErrors)
  clusterPlanes = setLeafInClusterP(leaves[iLeaf], clusterPlanes[p])
End For

For each iCluster
  // Recompute the plane minimizing total error for leaves of it's cluster
  clusterPlanes[iCluster] = recomputePlane(clusterPlanes[iCluster], leaves)
End For
```

The K-means clustering method achieved the best results. In the following sections we will examine the definition of the error measure and these steps in details. We refer the reader to chapter 6 to have more details about the preprocessing times and the evaluation of the obtained the results.

3.1.3 Error measure for a leaf and a plane for K-means

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

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

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, a leaf i is defined by its average normal n_i of unit length and its center p_i, that is, by a pair (n_i, p_i). A 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 $\mathbf{n}_i^T \cdot \mathbf{n}$ is also small (see figure 3.4). Thus an appropriate error measure for plane $[\mathbf{n}, d]$ and leaf $(\mathbf{n}_i, \mathbf{p}_i)$ is:

$$E([\mathbf{n}, d] \leftrightarrow (\mathbf{n}_i, p_i)) = D^2_{[\mathbf{n},d]}(\mathbf{p}_i) - 2\alpha \cdot \mathbf{n}_i^T \cdot \mathbf{n}, \tag{3.1}$$

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

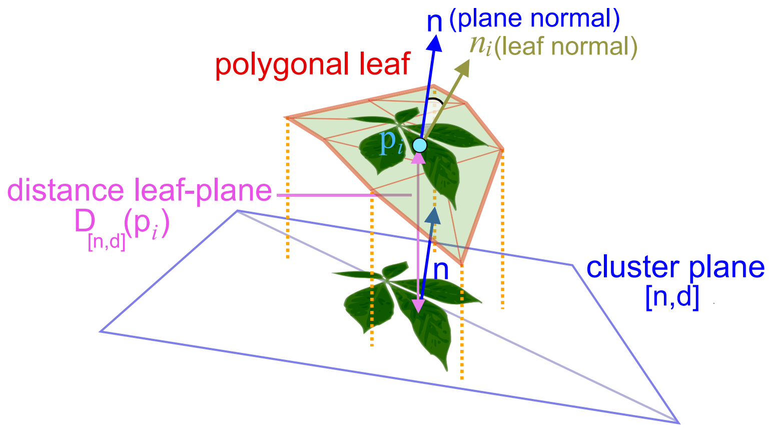


Figure 3.4: Error measure between a plane and a leaf for the K-Means clustering method.

3.1.4 Finding a good impostor plane for a leaf cluster

To find an optimal plane for a group of leaves, we have to compute plane parameters $[\mathbf{n}, d]$ in a way such 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

$$\sum_{i=1}^{N} E([\mathbf{n}, d] \leftrightarrow (\mathbf{n}_i, \mathbf{p}_i)) - \lambda \cdot (\mathbf{n}^T \cdot \mathbf{n} - 1) = \sum_{i=1}^{N} (\mathbf{p}_i^T \cdot \mathbf{n} + d)^2 + \alpha \cdot \sum_{i=1}^{N} (1 - (\mathbf{n}_i^T \cdot \mathbf{n})^2) - \lambda \cdot (\mathbf{n}^T \cdot \mathbf{n} - 1) \tag{3.2}$$

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

$$d = -\mathbf{c}^T \cdot \mathbf{n}. \tag{3.3}$$

where

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

is the centroid of leaf points. Computing the derivatives according to n_x, n_y, n_z, and substituting formula 3.3 into the resulting equation, we get:

$$\mathbf{A} \cdot \mathbf{n} = \lambda \cdot \mathbf{n}, \tag{3.4}$$

where matrix $\mathbf{A}$ is

$$\mathbf{A} = \sum_{i=1}^{N} (\mathbf{p}_i - \mathbf{c}) \cdot (\mathbf{p}_i - \mathbf{c})^T - \alpha \cdot \sum_{i=1}^{N} \mathbf{n}_i \cdot \mathbf{n}_i^T. \tag{3.5}$$

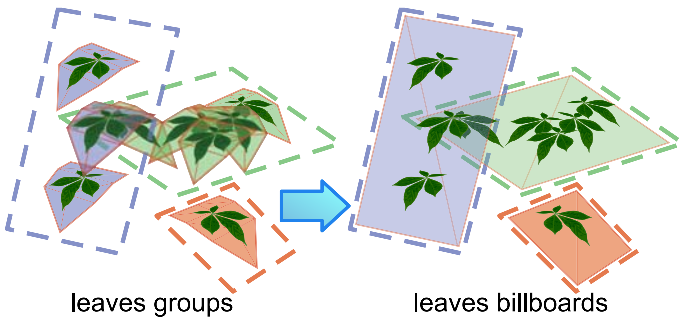
Note that the extremal normal vector is the eigenvector of the symmetric matrix **A**. In order to guarantee that this extremum is a minimum, we have to select that eigenvector which corresponds to the smallest eigenvalue. We could calculate the eigenvalues and eigenvectors, which requires the solution of the third order characteristic equation. On the other hand, we can take advantage that if **B** is a symmetric 3×3 matrix, then iterating $\mathbf{B}^n \cdot x_0$ converges to a vector corresponding to the largest eigenvalue from an arbitrary, nonzero vector x_0 [WEI03]. Thus if we select $B = A^{-1}$, then the iteration will result in the eigenvector of the smallest eigenvalue.

3.1.5 Leaves billboards generation

Unlike Fuhrman et al. [FUM05] who generated the billboards setting the clip planes around the polygonal model in the billboard plane position, taking all the triangles that are within the slightly separated clip planes to generate the billboard. We take each leaf cluster to build the corresponding billboard containing the grouped polygonal leaves, because, they rendered also faces that actually belong to other planes too. Instead, our preprocessing classifies each leaf in a single cluster, and hence to only one billboard. This lead us to a more accurate representation using the texture model we will introduce in chapter 7.1.

In order to create the billboard cloud, we find the minimal bounding box around the leaves of each group, as seen in figure 3.5. Then, for each billboard, a texture is generated by rendering into an off-screen buffer with an orthogonal projection of the polygonal faces of the clustered leaves.

(a)



(b)

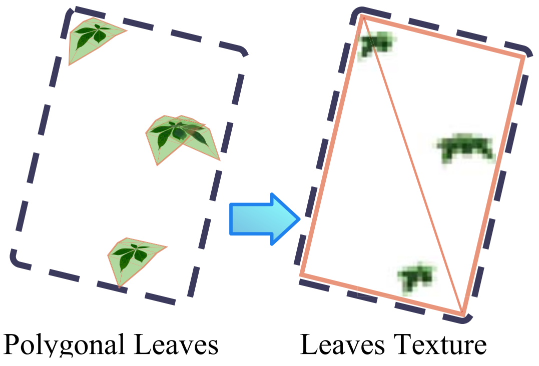


Figure 3.5: (a) Leaf groups are used to create the billboard planes . (b) A single texture map contain all leaves of one group.

The number of clusters can be specified by the user to control the number of billboards generated by the algorithm. The number of needed billboards depends on the size and complexity of the plant to be modelled. In general, the usage from 60 to 300 billboards for trees, or even fewer for smaller plants as bushes, is enough for visualization purposes. Another concern we took into account is the preprocessing costs, since image-based techniques typically involve extensive preprocessing and offline generation, and the k-means clustering converges to a suboptimal solution surprisingly fast, providing effective foliage representations (see figure 3.6), without the high computation cost and numerical inestability shown in other approaches, like Decoret et al. [DDSD03].

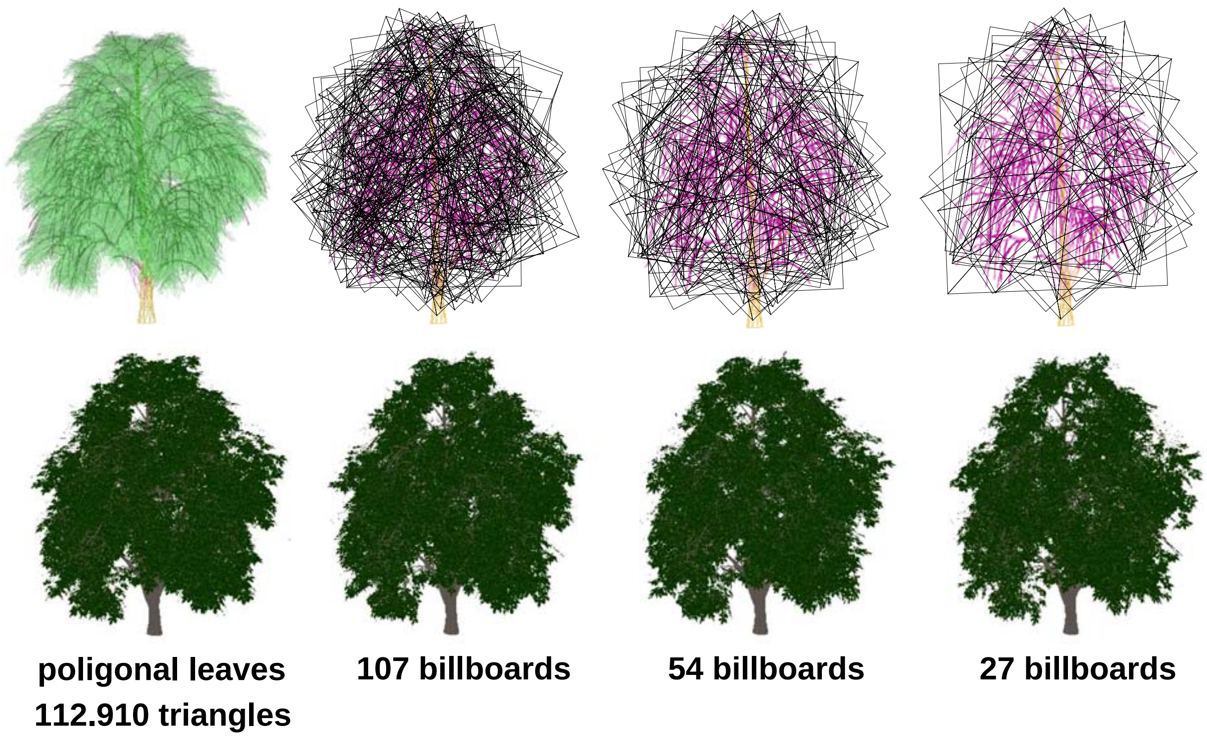


Figure 3.6: Chestnut (*Castanea Sativa*). Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

Chapter 4

Image-based vegetation representation

TEXTURES are useful for adding visual detail to geometry, but when extended to cover large areas such as billboards representing tree foliage, they require a very large amount of data. This approach also fails to preserve the small details that must be captured in the texture generation, unless a very large texture resolution is used, as shown in figure 4.1.

Unfortunatelly, all the image-based methods used in natural scenes have this drawback (e.g. [Jak00, MNP01, LCV04, FUM05, BCF^{+}05]). Some authors [LCV04, FUM05] tried to establish a LOD technique switching to the polygonal representation for close views, but the parallax problem perceived when changing the representation required to switch to the polygonal model quite far away, losing the performance benefit of using the image-based representations.

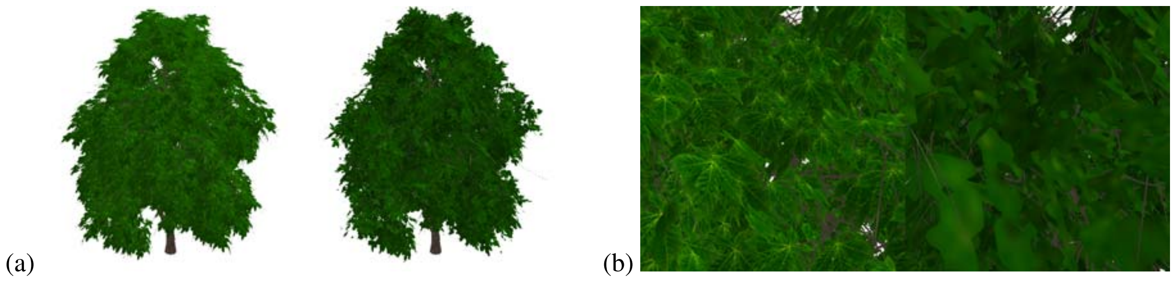


Figure 4.1: Chestnut (*Castanea Sativa*). Visual comparisons of the original polygonal tree (right) and a billboard cloud rendered with texture maps generated at 2048 × 2048 (left). (a) Shows a comparison of both representations from a far observer distance. (a) Shows a comparison of both representations from a closer observer distance, the billboards texture is not detailed enough, failing to replace the original tree.

4.1 Indirect texturing with billboard clouds

In order to keep the leaves details, in [GSSK05b] we introduced a procedural technique that replaces polygonal leaves by leaf patterns. This technique is called "indirect texturing", applying the new capabilities of current graphics hardware, which allows per-pixel computed values from one texture to be used as texture coordinates for an additional texture fetch [OL98].

In a preprocessing step the leaves positions are stored in a texture map to be used for placing each leaf on the billboard surface. In order to generate this texture, the method takes the rectangular projections of the polygonal leaves and using an off-screen buffer each rectangle is rendered. Each rectangle pixels encode the leaf position and orientation. The leaf position is defined by the rectangle upper left coordinate and an orientation index. The orientation index is defined by the angle of the leaf inside this rectangle.

We also generate another texture map with the leaves rotated at different angles, the so called Leaves Texture. In addition, if all the leaves have the same size we add a global scale parameter (due to clustering the billboard planes are roughly parallel to the leaves thus leaf sizes are approximately kept constant by the projections), further parameters can be added to our texture model, see section 4.2.2.

One of the key points of this simple technique is that all rectangle pixels represent the same leaf in the billboard, and all of them contains the same information, the 2D upper left coordinates and the leaf orientation index. Then

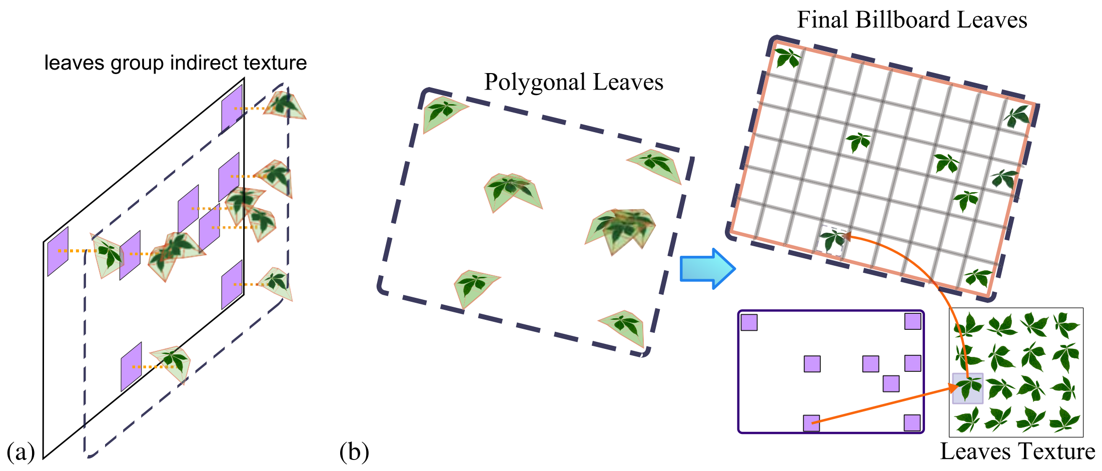


Figure 4.2: (a) Preprocessing step where leaves positions are stored in a texture map. (b) In the interactive visualization, the indirect texturing technique replaces the polygonal leaves by the billboard using the values in the texels of the indirect texture to fetch the corresponding leaf color image.

we are able to down-sample the texture size. In fact, this down-sampling could replace each rectangle pixels of the leaf projection by a single one. This extremal down-sampling leads to a regular cell based texture that tells us if there is a leaf in each of the corresponding billboard surface cells.

During rendering, we address this texture with the (u, v) texture coordinates to fetch the leaves information values, that are directly used as an indirect texture to fetch a leaf image from the Leaves Texture and render it above the billboard surface, as shown in Figure 4.2.

As explained, this approach can provide as highly detailed leaves as in the polygonal model. However, it can only use one leaf image per cell, which leads to a regular-looking pattern that fails to preserve the original leaves positions and orientations and cannot preserve the original overlapping, resulting in sparser-looking trees, as seen in figure 4.3.

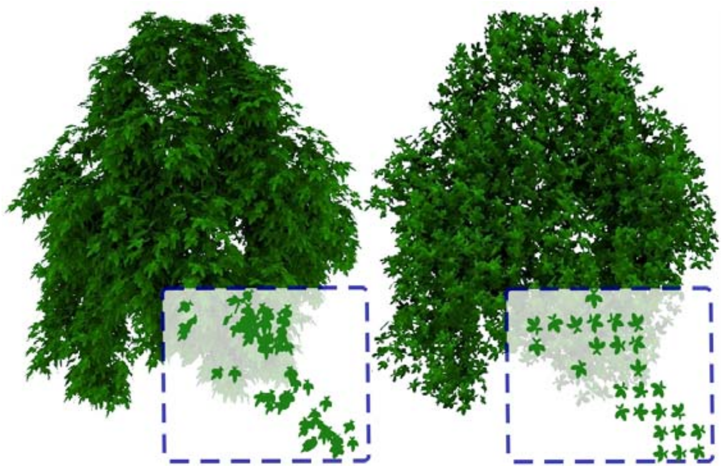


Figure 4.3: Comparison and close up of the polygonal tree and billboards with simple indirect texturing [GSSK05b].

4.2 Multi-layered indirect texturing

To evaluate the importance of preserving positioning and overlapping between leaves, let us look at a real example. Depending on the tree specie and age, a mature tree can have between 30000 and 200000 leaves. This intrinsic complexity leads to hundreds of leaves per billboard when a billboard cloud of about 260 quadrilaterals is used. Figure 4.4 shows one such billboard plane with 116 leaves, showing that the number of overlapping leaves can be as large as 10.

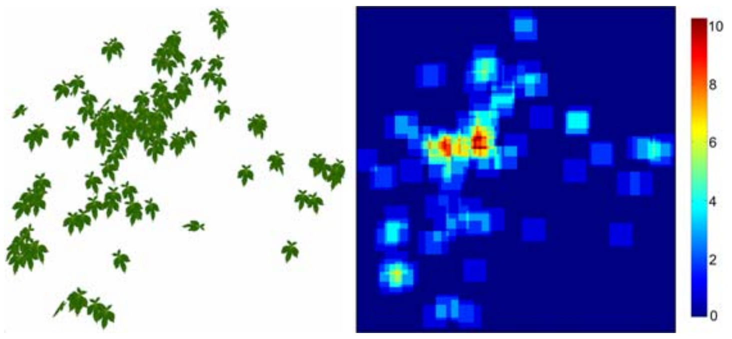


Figure 4.4: Group of 116 polygonal leaves clustered by a single billboard plane. The leaves distribution (right) leads to a up to 10 leaf fragment overlaps.

To properly handle the overlapping leaves within the same billboard, multiple texture maps would be needed, and the indirect texture should be encoded as a layered one, as shown in figure 4.5. However, even with this representation the leaves positions will remain discrete, still leading to a regular-looking pattern with respect to the number of cells of the indirect texture.

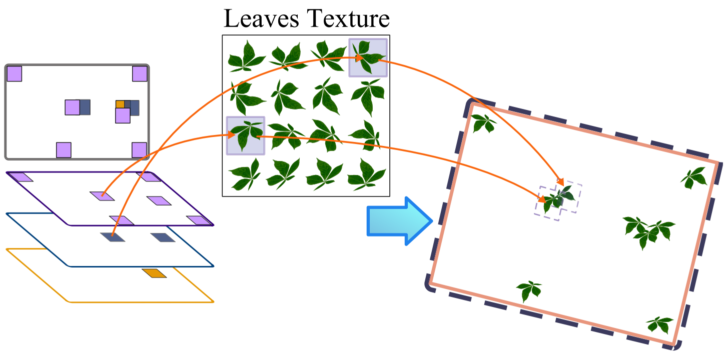


Figure 4.5: Overlapped leaves must be encoded with a layered indirect texture to be properly rendered.

In order to improve storage, look-up efficiency and the leaves positions accuracy, one indirect texture can be used to address *lists* of overlapping leaves, encoded in a second texture. Each entry in the lists contains a leaf position and an orientation index to address a leaf color image, as shown in figure 4.6.

The proposed method takes information from each group of leaves, generated by the billboard simplification algorithm, and prepares a set of textures that are effective at different scales of simulation. From close views of individual branches and leaves to bird's-eye flights the providing a viewing quality is very close to the one of the original complex polygonal model, as shown in chapter 6.

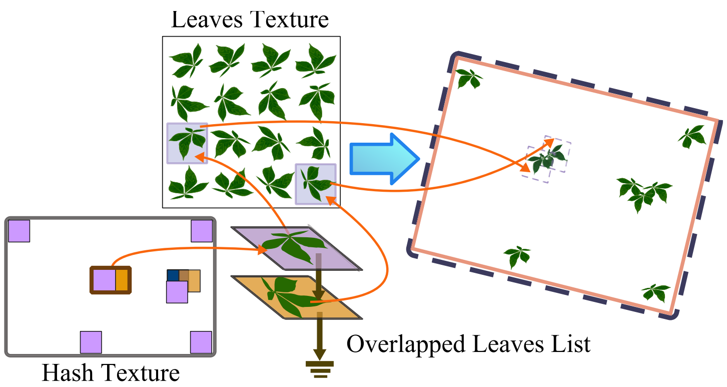


Figure 4.6: Indirect texturing with overlapped leaf fragments lists reduce texture requeriments.

4.2.1 Texture generation

In order to achieve an accurate overlapping while keeping the look-up efficiency, multiple layers must be used. So we subdivide the original billboard into a regular grid of cells, getting for each grid cell a set of cell-sized layers that may contain nothing (transparent) or a leaf fragment. Then we link those cell-sized layers into a list of overlapping leaf fragments as shown in Figure 4.7. We can consider that this grid of cells acts as a sort of spatial hash that accelerates fetches into this multilayered texture, so it is called the Hash Texture. The lists of leaf fragments are consecutively stored in a second separate texture, which we call the Lists Texture (see figure 5.2). Each list entry is a record that contains a reference to the third texture (the Leaves Texture), the leaf position vector, a scale factor, and any additional information that needs to be stored at the leaf level (see section 4.2.2). It is important to mention that the Leaves Texture is a high-resolution color texture including images of a tree leaf in all different orientations, and whose pixels get a transparent value in the areas not covered by the leaves.

One of the key points of the proposed method to preserve accurate foliage placement, is the generation of the Hash and Lists Textures. After the clustering process, we send each leaf as a point sprite to an off-screen buffer, encoding the leaf position and orientation and storing this information in the Lists Texture. To create the cell-sized layers, we use a depth-peeling technique [Eve99] that sorts the *leaf fragments* in front-to-back order, as shown in figure 4.7. This process is done for all leaves in a given billboard. It is important to mention that depth-peeling preserves the relative depth ordering of the leaf fragments with respect to the original polygonal model.

The size of the point sprite that holds the leaf information is defined with respect to its container billboard. This size is defined as the ratio between the 2D bounding box of the projection of the original leaf, and the size of the billboard (computed as $billboard_{pixels} = HashTex_{width} * HashTex_{height}$). In order to get the exact size of the point sprite in *pixels*, this size is multiplied by the total number of texels in the Hash Texture, following the expression:

$$leaf_{pixels} = \frac{area(leaf_{quad}) \cdot billboard_{pixels}}{area(billboard_{quad})}$$

leading to

$$leaf_{width} = leaf_{height} = \sqrt{leaf_{pixels}}$$

The last preprocessing step builds the Hash Texture, where each texel addresses one entry in the Lists Texture, or transparent when the cell does not hold any leaf fragments. It is important to mention that the billboard polygons not only have uv texture coordinates associated with them, but also encode the Hash Texture resolution, for later use. We use the average size of the leaves stored in with the billboard, if all leaves have the same size. If they have different sizes, this information is stored at the leaf level, in the Lists Texture, as mentioned before.

4.2.2 Information levels and extensions

Our image-based representation encodes the information for rendering the foliage in different structures, with different levels of granularity depending if the information will be the same when addressed from a set of trees, from each single tree, each group of leaves, or even from each independent leaf.

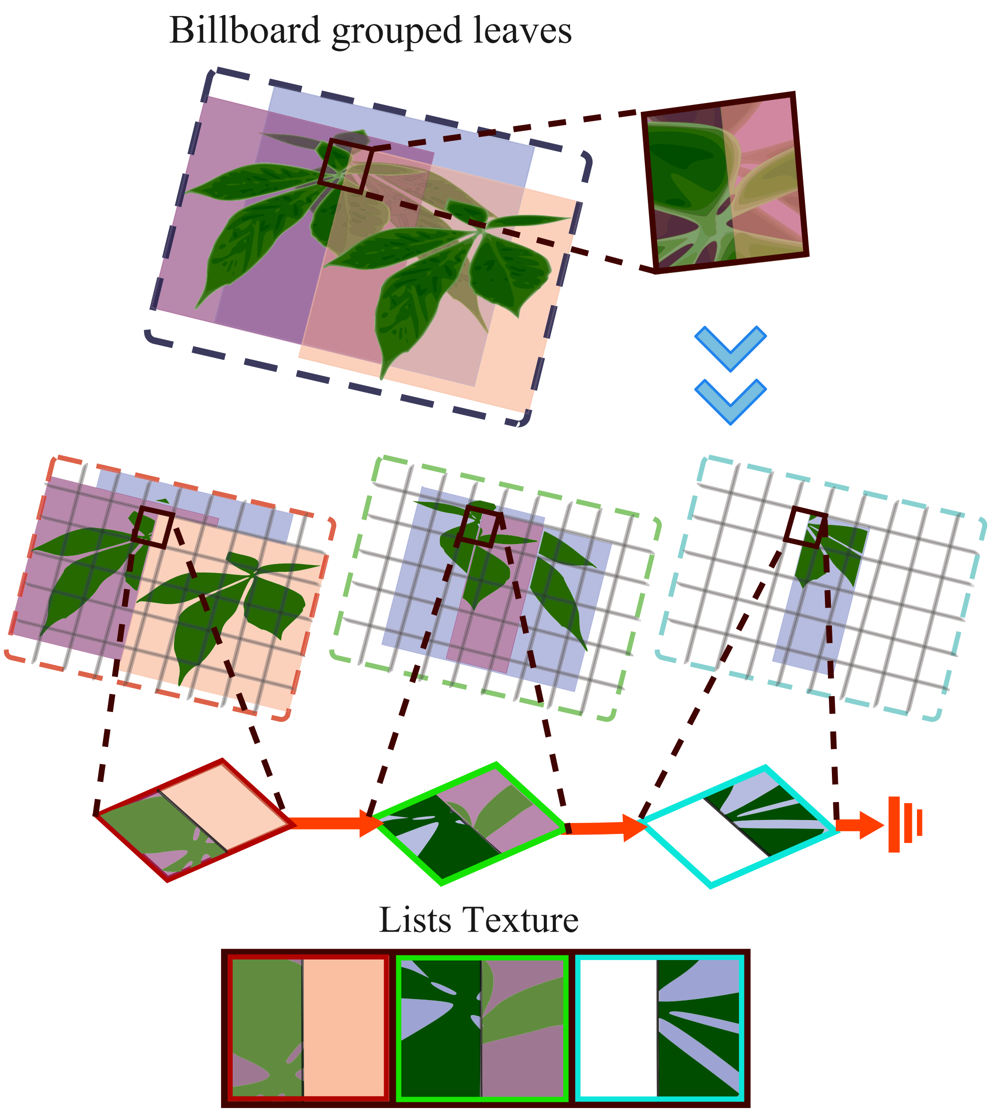


Figure 4.7: Each leaf is processed as a point sprite. The point sprite can appear in more than one grid cell of the Hash Texture. The point sprites are processed with depth peeling [Eve99] to build an overlapping leaf fragments list for each grid cell.

- **Batch level**: The information stored in the *batch level* is that information that is shared by a set of trees that will be place in the same region of the scene. This information is added when a preprocessor build the instancing groups to optimally render the final scene.
- **Tree level**: The information at the *tree level* is shared by all the foliage of a single tree, and can be stored in different ways depending on the final scene implementation, normally can be added in the billboard cloud texture coordinates, or in a look-up texture, enabling to use it when rendering each single tree.
- **Leaf group level**: The information stored in this level is shared only by the group of leaves contained in the same billboard plane, and is normally stored in the vertex texture coordinates of each billboard. For instance, the billboard polygons not only have the (u, v) texture coordinates associated with them, but also encode the Hash Texture resolution, used for the Shader-LOD detailed in section 5.1.1. In addition, stores the average size of the leaves in the billboard, if all leaves have the same size.
- **Leaf level**: This is the finest level of information, and is stored in the Lists Texture, enabling us with the indirect texturing to control specific parameters about how to render each single leaf of the tree. If the polygonal model had different leaves sizes, this information would be stored at the leaf level, in the Lists Texture.

This texture-based model lead us to include further extensions, specially at the *leaf level*, to enhance the rendering quality. For instance, the ambient occlusion term for the lighting illumination is defined for each leaf (see chapter 5) to achieve a inexpensive global illumination term. One of the benefits of our representation is the low memory costs needed, compared to complex polygonal models or volumetric representations. We would like to refer the reader to chapter 6 for further information in our method memory costs.

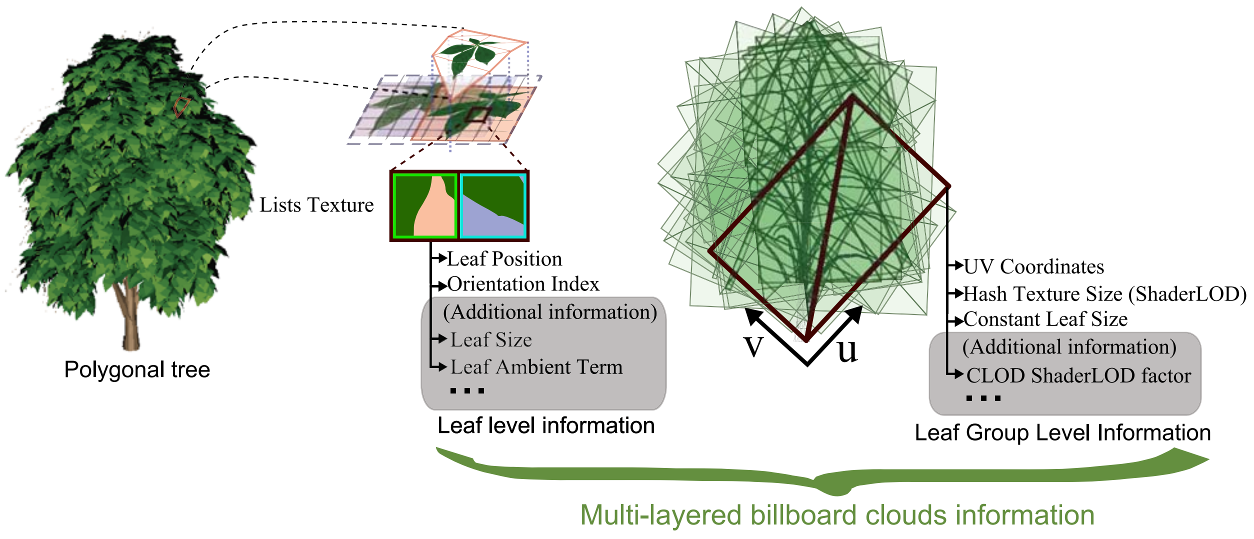


Figure 4.8: In a preprocessing step with the complex polygonal model we store the indirect texturing information in different structures. The leaf level information is stored in the Lists Texture, and the tree or leaf group information is encoded in the billboard cloud mesh.

Chapter 5

Real-time vegetation visualization

IN previous chapters we described a indirect texture-model, and how can we generate those indirect textures to be used with the billboard cloud representation extracted from a complex polygonal model. This chapter provide the details concering of how this method is used during the visualization, enabling us to provide realistic real-time visualizations from bird's eye to insect scale of plants and trees, using a Level-of-Detail technique for the billboard geometry and the texture-based representations. Finally in section 5.2, we present the realistic lighting model specially definded for such representations.

5.1 Visualization algorithm

When the graphics hardware draws a billboard, a single pass fragment shader is called with the corresponding uv texture coordinates. These coordinates are used to fetch the respective cell in the Hash Texture. As mentioned earlier, this cell can be either transparent, so the shader will also return a transparent fragment, or it can contain a reference to a list in the Lists Texture. Then, in an iterative process, each entry in the list is fetched until an opaque leaf fragment is found, or there are no more leaf fragments to evaluate. In the latter case a transparent color is sent as output. This process is illustrated in Figure 5.1.

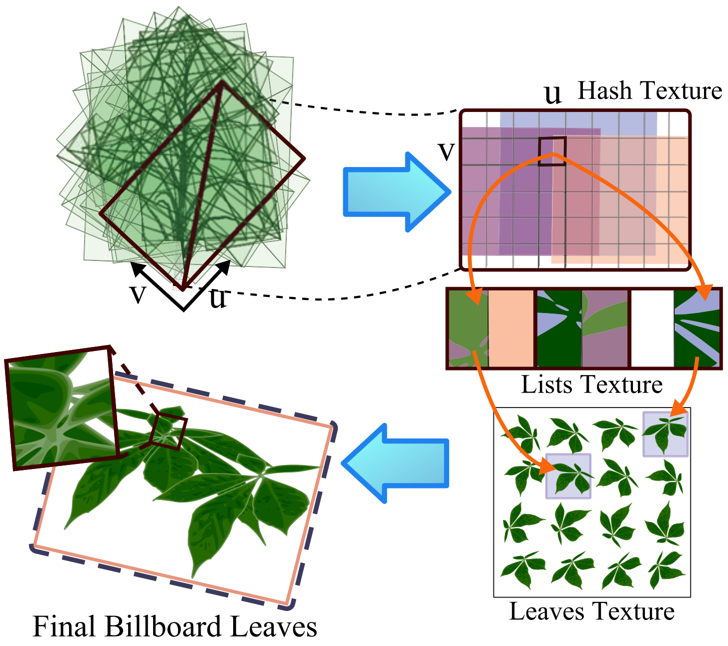


Figure 5.1: Multi-layer indirect texturing evaluation in the single pass fragment shader.

The leaves are positioned with a continuous representation: each leaf has an associated vector that represents its position in local billboard coordinates (see Figure 5.2).

When evaluating each entry in the list, the reference to a leaf image into the Leaves Texture and the 2D vector giving the exact position of the leaf are retrieved. The position is subtracted from the uv coordinates the shader received to obtain a vector we call the Local Position vector. On the other hand, the leaf reference into the Leaves Texture is used as a vector pointing to the upper left corner of the corresponding image of the leaf. This vector is added to the Local Position vector to retrieve a texel from the Leaves Texture which either contains an opaque pixel with a color from the leaf, or transparent, indicating we indexed outside the leaf and we should continue with the next entry, as shown in Figure 5.2. If we add size information in the leaf entries, the Local Position vector is scaled accordingly. As mentioned before, if the Leaves Texture evaluation gives a transparent value, the next entry in the list is evaluated. If it is not transparent, it means a leaf was indexed, and its color is used as output from the shader.

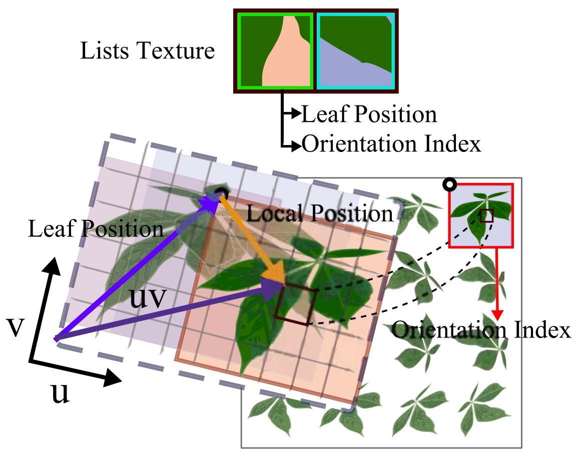


Figure 5.2: Information stored at an entry in the list of leaves for a hash cell.

Another important factor to take into account is that leaves overlap with a different order depending on the viewing direction. For instance, the polygonal leaves shown in figure 5.3 presents a clustered group of leaves where this effect can be seen. To incorporate this feature, the Hash Texture also stores the size of the list it is indexing, so we can access the list by both ends, just by looking at the direction of the billboard normal with respect to the viewer's direction. This also offers the advantage of allowing to index *two* Leaves Textures, one with the front sides of the leaves and the other with the back sides. These two textures can be unified in a single texture, and thus, a single indexing framework, to avoid further conditional branches.

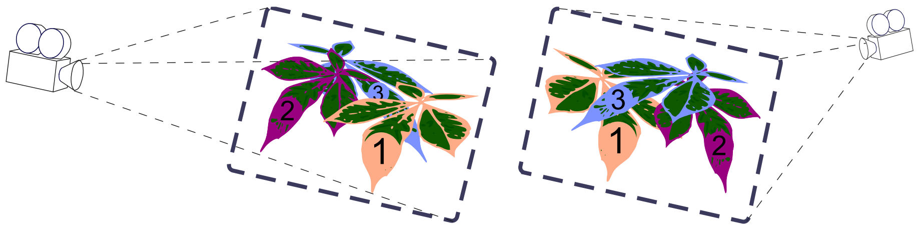


Figure 5.3: Polygonal overlapped leaves from different view directions. Given the left view direction, the leaf *leaf_1* overlaps the *leaf_2* and *leaf_3* ones. On the other hand, from the opposite view direction, the user expects to see the inverse overlapping order, where the *leaf_3* is above the *leaf_2* and *leaf_1* ones.

5.1.1 Level-of-detail with indirect texturing layers

Although this method was designed for close or medium range views, it works well if mip-mapping is used for the Leaves Texture, as it would produce the right effect at large distances. However, the shader computations described above would be executed no matter of the viewing distance, resulting in an unnecessary overhead that impacts on the fill-rate without representing a significative improvement in the visualization quality. To alleviate this problem, we introduce a progressive Level of Detail technique at the shader level.

For the above described method to work well, a strategy for progressively reducing the number of evaluated list entries should be thought of. The core of the problem is that, when a billboard is far from the observer, each texel in the Hash Texture will cover few pixels in the screen.

In order to apply the Level-of-Detail we need to determine the screen-space distance between adjacent texels of the billboard at runtime. Fortunatelly, modern GPUs include instructions (ddx, ddy) that compute the derivatives of their arguments with respect to screen-space x and y. Evaluating those functions using the billboard (u, v) texture coordinates multiplied by the Hash Texture size (stored in the billboard as described in section 4.2.1) enable us to compute the number of screen pixels per billboard texel, or equivalently the size of the Hash texel in screen-space.

Once upon we know the size of the billboard Hash texel in screen-space, its diagonal length multiplied by an user-defined factor establish the number of list entries evaluations needed. In addition, this expression is also used to choose the mip-map level of the Leaves Texture. The user-defined factor is useful to tune the trade of between performance and visual quality in our Level-of-Detail technique.

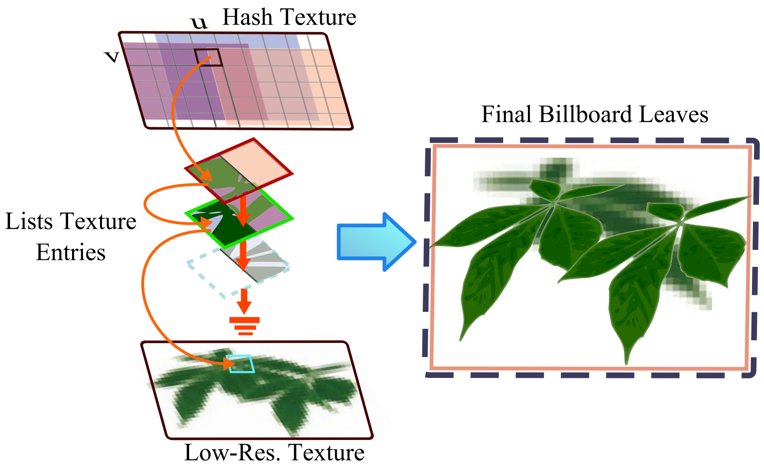
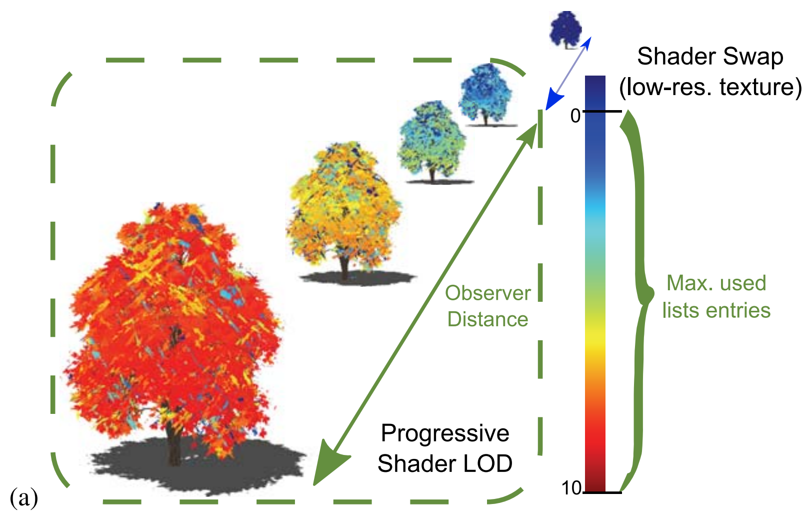


Figure 5.4: As distance increases, the shader-LOD technique evaluates less list entries, and the missing ones are substituted by a low-resolution texture.

Basically, this technique consists of reducing the number of evaluated list entries when the distance from the observer increases or the angle of the billboard plane with respect to the camera plane is almost perpendicular (see figures 5.5 and 5.7). Under such circumstancies, the Shader-LOD formula reduces progressively the number of lists entries being used. However, evaluating less list entries means evaluating less leaf fragments, which will generate holes in the tree that get bigger as the viewing distance increases. This is solved by introducing a regular low-resolution texture that represents the missing leaves, generated in a preprocessing pass from the layered texture. So, as distance increases, less elements are evaluated and the missing ones are replaced by a single texture fetch into the low-resolution texture. Finally, when the viewing distance is high enough, no list entries are used any more and only the low-resolution texture is evaluated.

Our experiments show that the fact that the leaves that are evaluated from a list entry are also present at the low-resolution texture does not produce any noticeable artefact. A comparison can be found in Figure 5.6. Finally, when the tree is at a very large distance, the technique would show only the low-resolution texture, without fetching any list. So, more evaluations can be avoided by a switch in the shader context to a shader that only evaluates the low-resolution texture without further computations.



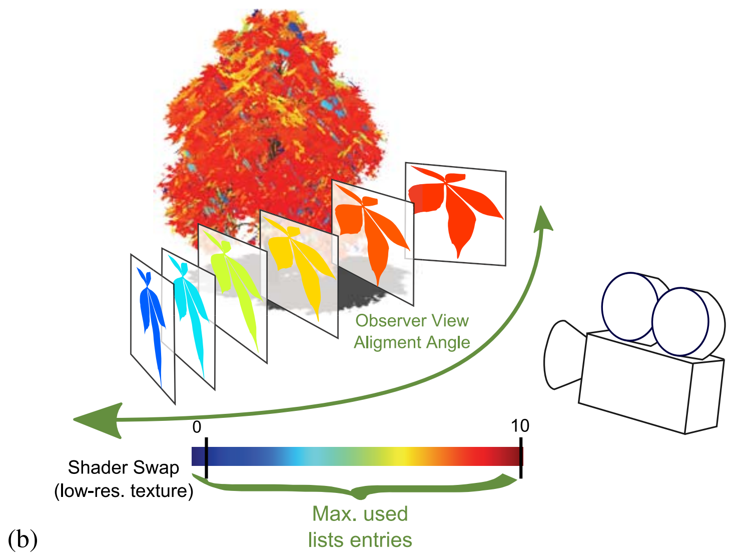


Figure 5.5: Description of the shader-LOD technique. Color red means the maximum level of detail. From red to blue represent the progressive decreasing level of detail. (a) Shader-LOD reduces the number of evaluated list entries when the distance from the observer increases. And the missing ones are substituted by a low-resolution texture. (b) In addition, when the angle between the billboard plane and the camera plane are almost perpendicular, the number of list entries is also progressively reduced. In both cases (a) and (b), the more red the foliage appeares is the notion that we are evaluating almost all the lists entries with respect to the maximum overlaps of the billboard. The full blue color means fragments that are only evaluating the low-res texture.

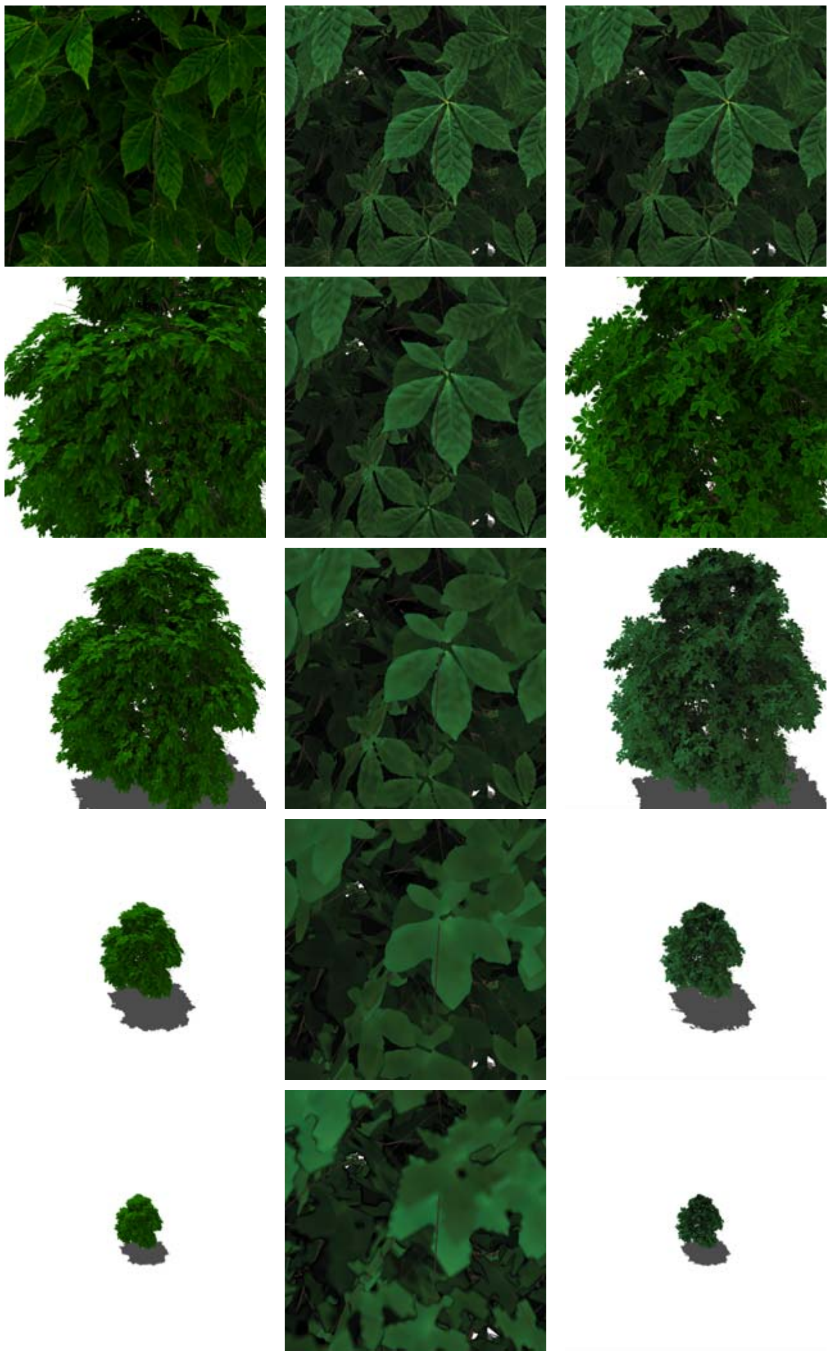


Figure 5.6: Comparison between the polygonal tree on the left and the multi-layer technique on the right, preserving accurate alignment of the leaves. The middle column shows the progressive shader LoD. The small holes are due to slight changes in the leaves alignment.

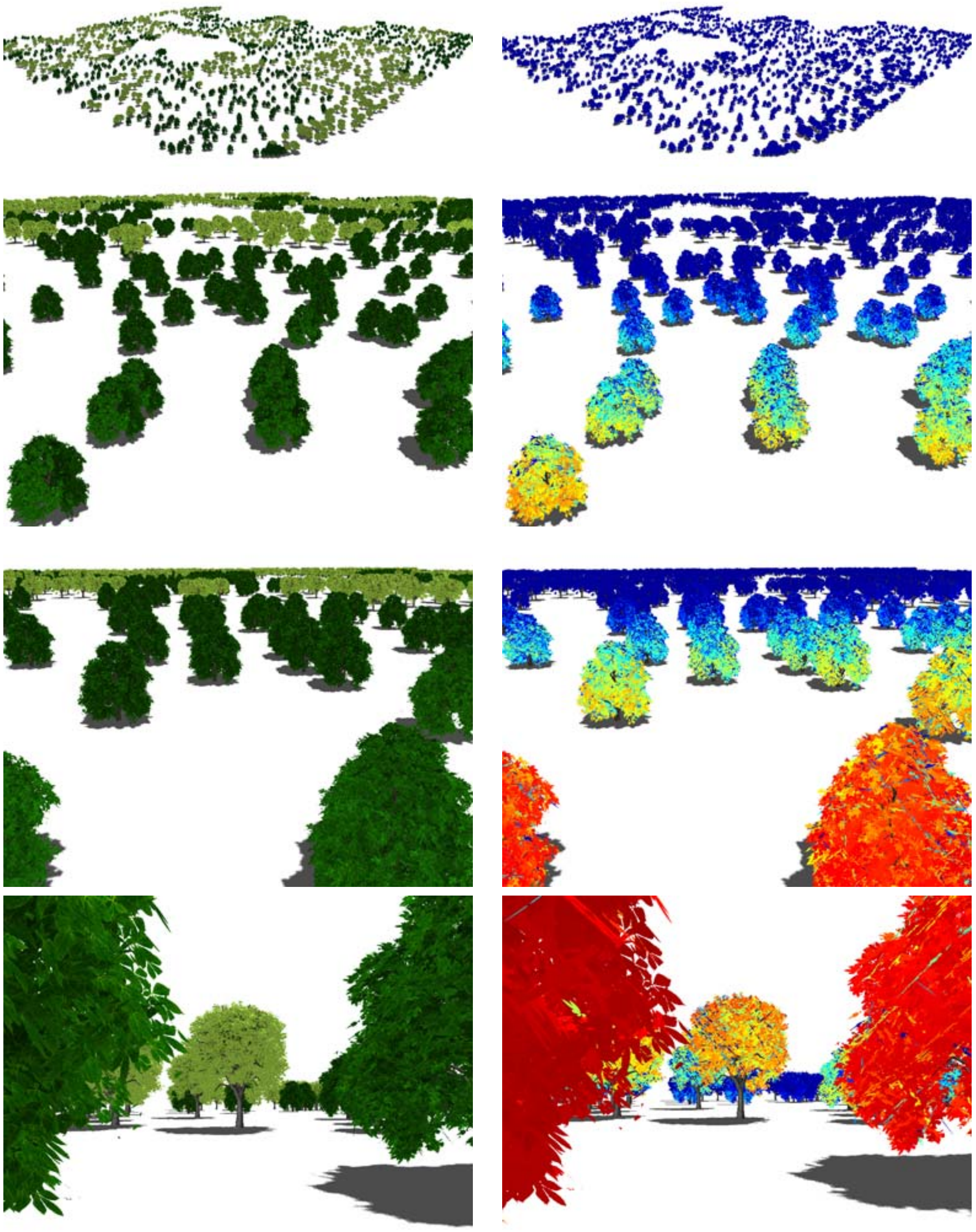


Figure 5.7: Forest scene with 2000 trees. On the left row we show the realistic rendering computing the multi-layer indirect texturing and the lighting model detailed in section 5.2. On the right we show the maximum number of lists entries evaluated for each billboard depending on the observer distance. The furthest views only use the low-res texture, progressively being replaced by high definition leaves, as the observer gets closer to the foliage.

5.2 Realistic lighting model

Lighting plays an important role for the appearence of trees and plants. So we introduced an adapted lighting model to the indirect texture-based model presented here. In the next section we describe the direct lighting computations, such as, the double-sided lighting of billboards, and an extension to have effective soft-shadows with our indirect texturing method. In addition we have added a precomputed ambient term, in order to simulate the effect of the indirect illumation at the tree level, detailed in section 5.2.2.

5.2.1 Direct lighting computation

The shading model used for the trees can be split into two techniques. On the one hand, trunk and branches simplified models are rendered with bump mapping, and shadow mapping. On the other, the foliage lighting is obtained as the combination of the proposed multi-layer technique with an adapted shadow mapping technique.

Billboard clouds approximately preserve the original geometry, thus they can also replace the original tree when shadows or global illumination effects are computed. Unfortunately, each billboard has a single normal for all leaves belonging to it. So, when direct illumination is calculated, a single normal for all leaves would make the planar billboard substitution too obvious for the observer. To avoid this unpleasant effect for direct lighting, we propose to use the original normals of the leaves. To be able to do this with our multi-layer indirect texturing technique, we encode a normal map in a similar way as the Leaves Texture, where each different leaf orientation image encodes its own local normal map.

If needed, a correction for the billboard plane normal could also be stored in the entries of the Lists Texture, and used in conjunction with the local normal retrieved from the normal map to obtain a final normal to compute the shading. In this case, we would end up having a normal for the billboard plane, one correction given at the leaf level from an extra field in the Lists Texture entries, and a final local normal correction retrieved from the leaves normals map.

To compute soft-shadows, i.e. the visibility from point and directional lights (like the sun), we use Light Space Perspective Shadow Maps [WSP04]. To generate the shadow map, we adapted its usage in combination with the multi-layer method: for close-up views, where shadows need to be highly detailed, we use a full evaluation of the layers as described before. But, as soon as the observer gets further enough, as described above, we switch to the low-resolution evaluation shader which provides good results at a fraction of the evaluation cost. We cannot use the progressive level of detail technique with respect to the light source position, as described in previous sections, because the observer could get very close to the shadows and notice the low-resolution quality. As can be seen in figure 5.8, the lighting and the shadows are approximately the same in our billboard cloud representation than in the original complex model.

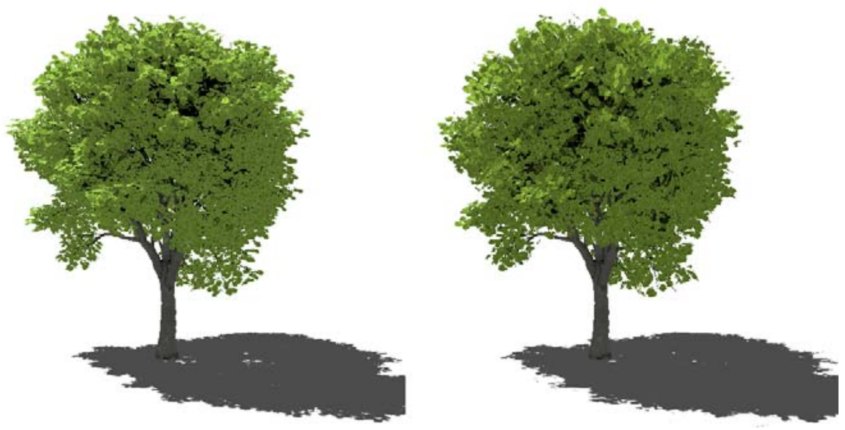


Figure 5.8: Visual comparison: Polygonal cork tree with 20144 leaaves simplified on the left to 121 multi-layered billboards on the right. As can be seen, the lighting and even shadows generated by the polygonal model and the billboard clourd representation are approximately the same.

5.2.2 Ambient occlusion term precomputation

In order to complete the lighting model, we incorporate a precomputed ambient occlusion term [GFS07] to take into account global illumination effects, computed for the original polygonal tree. To have a compact representation, the ambient occlusion term is adapted to the shader LoD technique described in Section 5.1.1. As we mention in Section 4.2.1, each entry of the Lists Texture can contain additional information to be stored at the leaf level. We encode the ambient occlusion term as one new field in each entry in the Lists Texture. Because of the limitations of current graphics hardware for packing into single precision RGBA channels, we decided in our implementation to store the ambient term in a separate texture, indexed the same way as the Lists Texture. Of course, another option would be to store this information in four integers packed into a floating point value, but it would provide more or less the same performance results.

To be compatible with our high-resolution representation of leaves into the Leaves Texture, we extrapolate the ambient term computed for the polygonal leaves to the four vertices of the 2D leaf bounding box. This is computed by setting a simple over-determined system of linear equations that relate the ambient term values at each projection of a polygonal vertex onto the billboard plane, with the corners that define the 2D bounding box of the projected leaf. See Figure 5.9. This system can be solved with any numerical method, or it can be approximated by taking only the four vertices of the original leaf that include the maximum value, the minimum value, and two extra values of the ambient term, and by solving only a system of 4 equations with 4 unknowns.

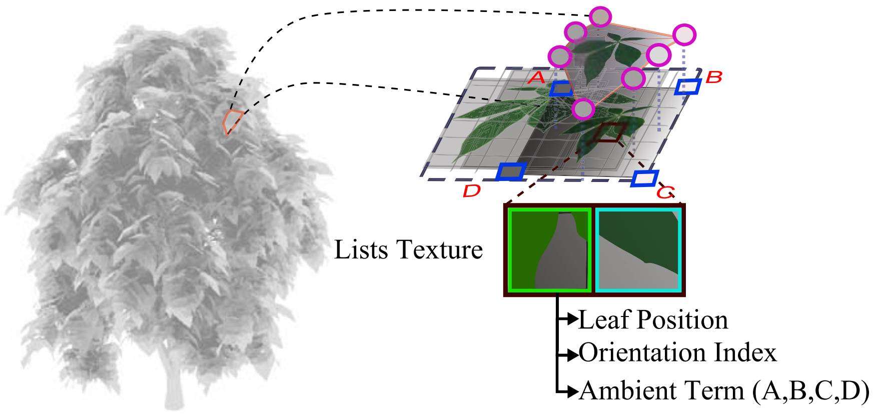


Figure 5.9: Ambient term generation for the multi-layer leaves representation is encoded as a new field of the List Texture at the leaf level. The small squares represent the actual ambient occlusion values at the corresponding vertices.

The resulting values are encoded as described in section 4.2.1: The ambient term fields in the entries of the Lists Texture store the four values corresponding to the corners of the leaf 2D bounding box. In the visualization, this ambient term is linearly interpolated in the fragment shader to provide the final ambient occlusion factor for the fragment on screen. One optimization would be, when the ambient term is stored in a separate texture, to use two consecutive rows to store the information for a single list, and using a 2×2 block of texels to store the ambient occlusion values. Then, the ambient values would be interpolated by the GPU when queried. For this to provide good results, the Local Position vector mentioned in section 5.1, should be used in combination with the texture coordinates used retrieved from the Hash Texture. For the low-resolution texture, we use an ambient occlusion term encoded in the alpha channel of the low-resolution leaves texture.

5.3 Implementation Details

The method presented so far requires one Hash Texture for each billboard, and in normal cases, there could be 260 and more billboards, largely exceeding current hardware capability of indexing textures at once. To handle this, several context switches would be needed, severely impinging on the overall performance. To solve this problem, all Hash Textures are condensed in a single large Hash Texture Atlas. In the same way, the different lists of the Lists Textures for each billboard are stored together in a single Lists Texture Atlas. In our experiments, the largest textures needed to store the Hash Textures and Lists Textures is about 1024^2 texels.

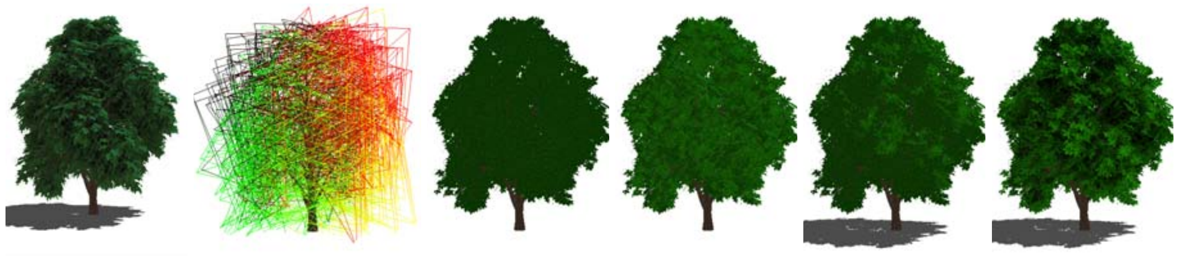


Figure 5.10: Comparison between the polygonal tree and the multi-layer technique. From left to right: Original polygonal tree fully lit, the wire frame billboards, billboards with diffuse lighting, direct lighting, shadows and, finally, the full illumination model including the ambient occlusion term.

In order to save memory on the graphics card, we decided to use simple precision integers for the Hash Texture. The current implementation uses the four-channel texture in the following manner: the first two channels encode an address into one of the 16×16 subregions of the Lists Texture. Then, the next two channels encode the local displacement within each subregions where the lists are stored. In this way, we can index any coordinate into the texture, up to 4096^2 texels. It is important to mention that the new Shader Model 4 GPUs incorporate extensions that enable the encoding of full integer values in each texture channel, and allow textures up to 8192^2 texels.

In addition to the shader level-of-detail technique described in section 5.1.1, we used different billboard clouds for the foliage with a decreasing number of billboards, and switched among them as the distance to the observer increased. We stored three different billboard clouds that were stored in a single Vertex Buffer Object and indexed as necessary. The simplest billboard cloud is used in conjunction only with the low-resolution texture as described above, and the changes in billboard levels are ensured to be consistent with the corresponding shader LoD at the time of the switch. With respect to the tree trunk and branches, we used a progressive geometry-based Level-of-Detail technique, as described in [Hop96].

Chapter 6

Results

THIS chapter presents the final results obtained with our techniques. In order to evaluate them we have generated extremal simplifications for a variety of trees and bushes with different foliage densities. In table 6.1 we show a description of each one of the polygonal models and the simplified billboard clouds with their preprocessing times. The benefit of our billboard clustering method is that it's simple and effective, being also between 10 to 20 times faster than the greedy billboard generation algorithm presented by [FUM05]. However, the texture generation times for their method are unknown, and could not be compared, but in our case the low-resolution and the multi-layer textures are generated within 1 or 2 minutes. Both preprocessing algorithms were compared in a 2GHz Pentium 4 with 2GB RAM machine.

Tree Species	**Leaves**	**Polys/ leave**	**Billboards**	**Billboards gen.(s)**	**Std. tex. gen.(s)**	**Multi-layer tex. gen.(s)**
Oak[2]	23660	2	194	35	9	127
Cork[1]	20144	4	121	28	7	125
Ivy Ground[3]	18274	2	359	40	15	140
Chestnut[1]	11291	10	278	30	11	120
Horse Chestnut[1]	9366	18	202	22	9	90
Ivy Sculp.[3]	8234	2	420	42	15	135
Shrub[2]	2388	2	198	18	10	68
Maple[1]	3662	2	175	13	8	64
Alder[1]	1049	2	228	17	10	92

Table 6.1: Description of different species showing the precomputation times for the billboards generation, the standard textures generation, and the multi-layer textures (in seconds). Labels mean: [1]Generated by Xfrog [Xfr06], [2]Generated by Forester, [3]Generated by Ivy Generator.

Tree Species	**Leaves**	**Polys/ leave**	**Billboards**[4]	**Memory**[4]
Oak[2]	23660	2	194	3119734
Cork[1]	20144	4	121	1462870
Ivy Ground[3]	18274	2	359	3189635
Chestnut[1]	11291	10	278	3486762
Horse Chestnut[1]	9366	18	202	3376653
Ivy Sculp.[3]	8234	2	420	3682312
Shrub[2]	2388	2	198	1349091
Maple[1]	3662	2	175	1381600
Alder[1]	1049	2	228	1332837

Table 6.2: Description of different species showing the number of leaves, polygons per leaf, billboards and memory usage (in bytes) with our method. Labels mean: [1]Generated by Xfrog [Xfr06], [2]Generated by Forester, [3]Generated by Ivy Generator and [4]Using texture compression DTX1-5, which provides a 1:6 compression ratio.

The billboard cloud preprocessor have been integrated in Autodesk Maya, in order to provide our tool to 3D content creation designers that would be able to compose the large natural scenes at the same time. Figure 6.1 show the main interface of our Maya plug-in.

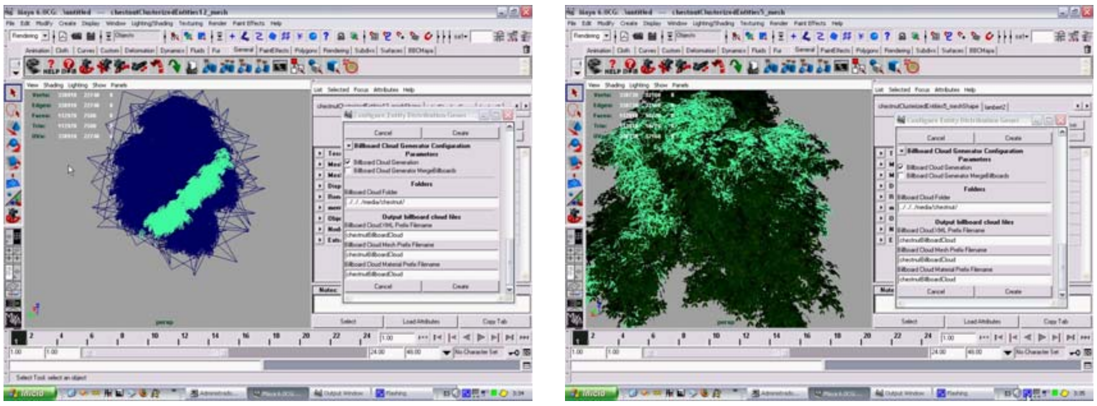


Figure 6.1: Billboard cloud preprocessor Maya plug-in interface.

To evaluate the results of some of the tree models listed in table 6.1, we include a comparison between the complex polygonal model, and three billboard cloud levels of details used in some of the final scenes shown in figures 6.10 and 6.11.

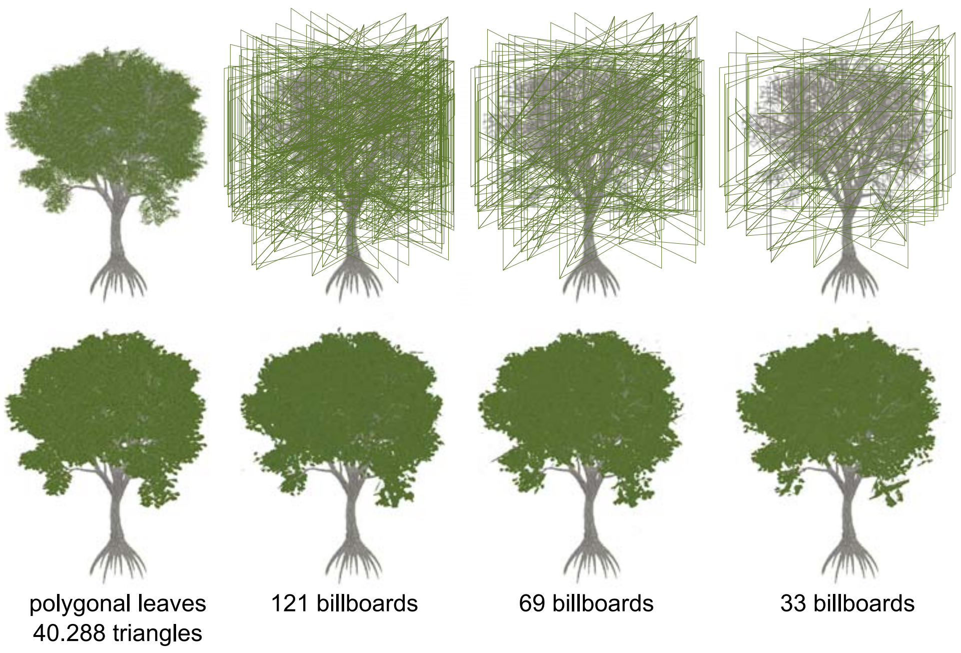


Figure 6.2: Amur Corktree (*Phellodendron amurense var. Japonica*) in adult age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

When compared to the classic billboard techniques with standard texture mapping, we can see in figure 6.8 that the multi-layered indirect texturing compares well with the original leaves for one billboard. The method is able to keep accurate leaves positions and orientations as opposite of the standard indirect texturing, as we shown in chapter 7.1, and largely surpassing the quality of a low-resolution, single texture approach.

The usage of the shader level-of-detail in combination with the progressive reduction in the number of billboards and the progressive LoD change for the trunk and branches geometry, allow large forest scenes to be visualized at interactive frame rates, as shown in figures 6.10 and 6.11.

All scenes were rendered in OpenGL/Cg integrated into the Ogre3D engine, and running on a 3GHz Pentium 4, 3GB of RAM and a NV8800GTS graphics card with a resolution of 1280x1024 pixels. The resulting frame rates of the system for varying number of trees are presented in table 6.3. Figure 6.10 show those complex scenes with tens of thousands of trees under very different lighting conditions and viewing distances.

Our experiments show that the visualization of one or few single complex trees, is more efficient with the

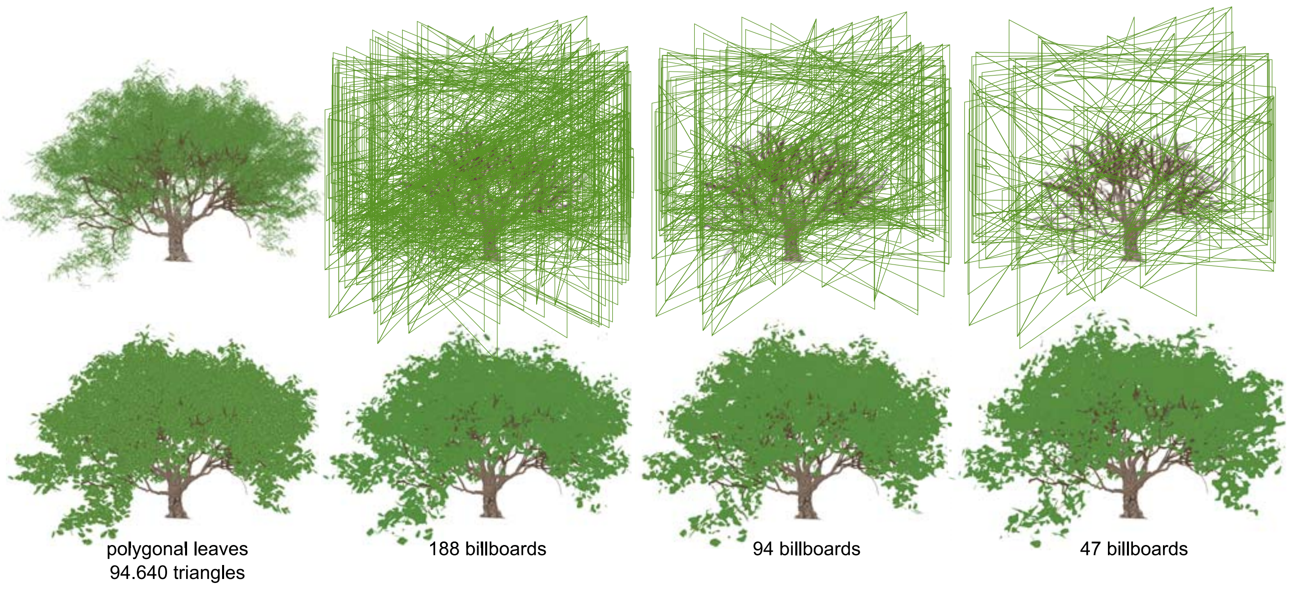


Figure 6.3: Oak (*quercus*) in adult age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

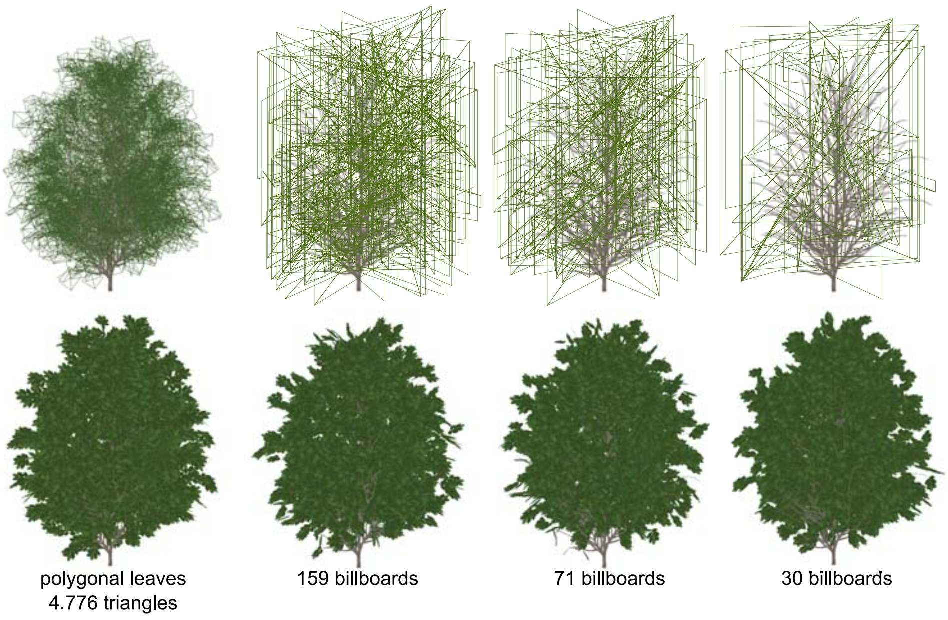


Figure 6.4: Hornbeam (*Carpinus betulus*) in adult age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

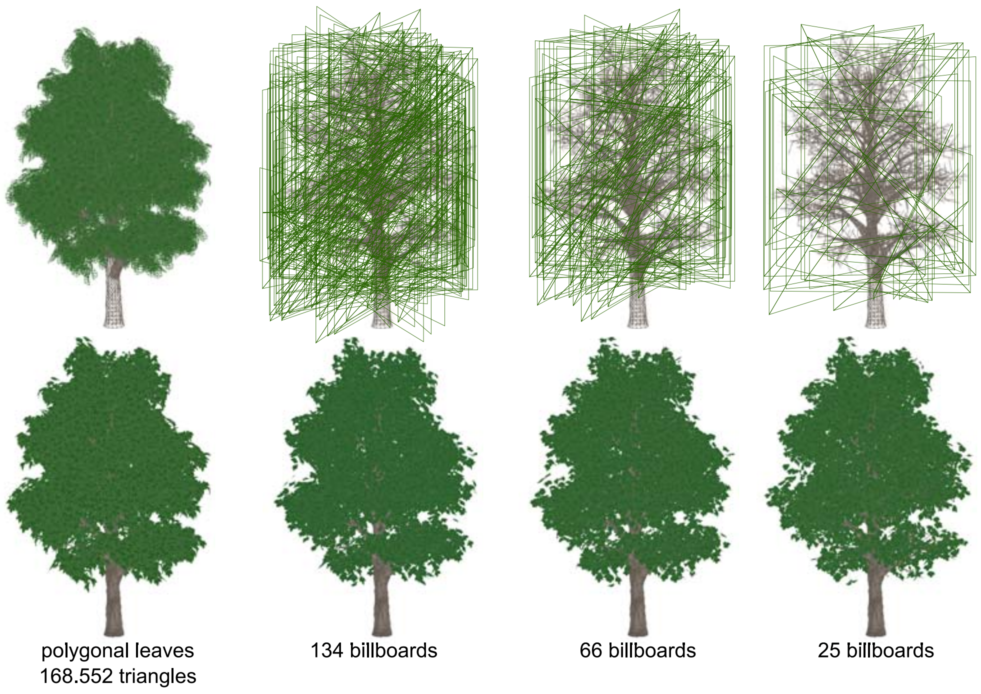


Figure 6.5: Horse chestnut (*Aesculus hippocastanum*) in adult age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

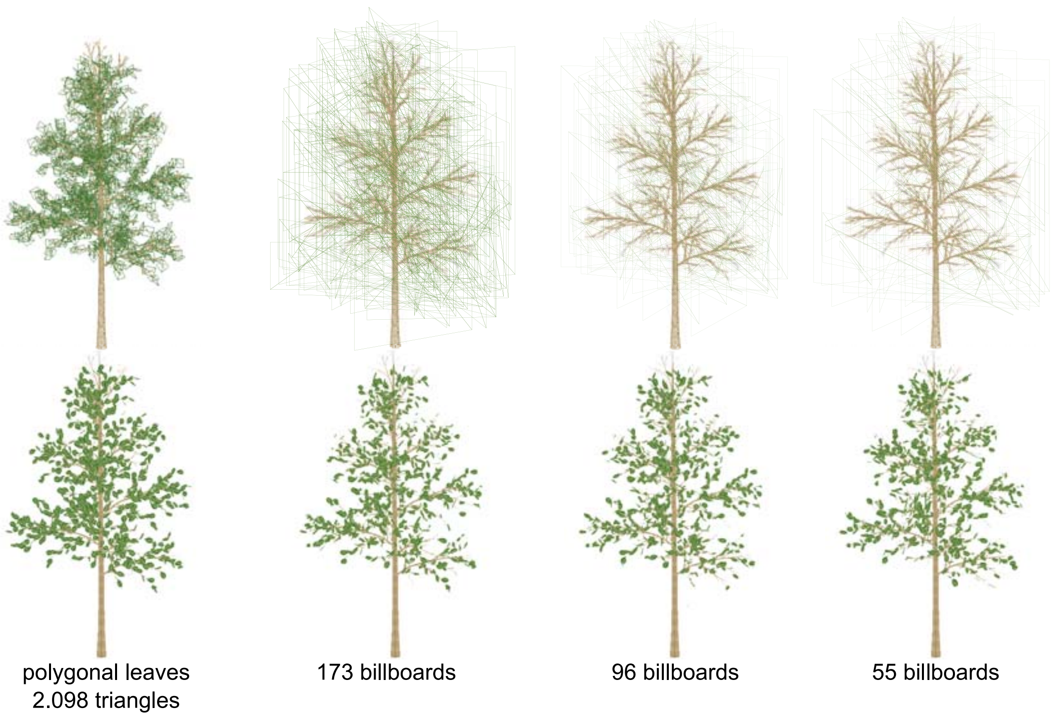


Figure 6.6: Alder tree (*Alnus glutinosa*) in young age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

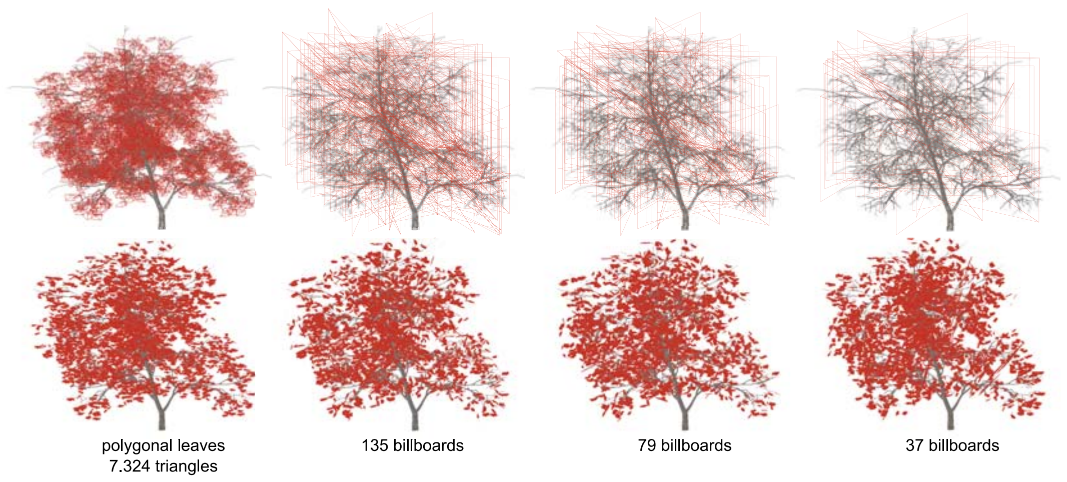


Figure 6.7: Japanese Maple (*Acer palmatum*) in young age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

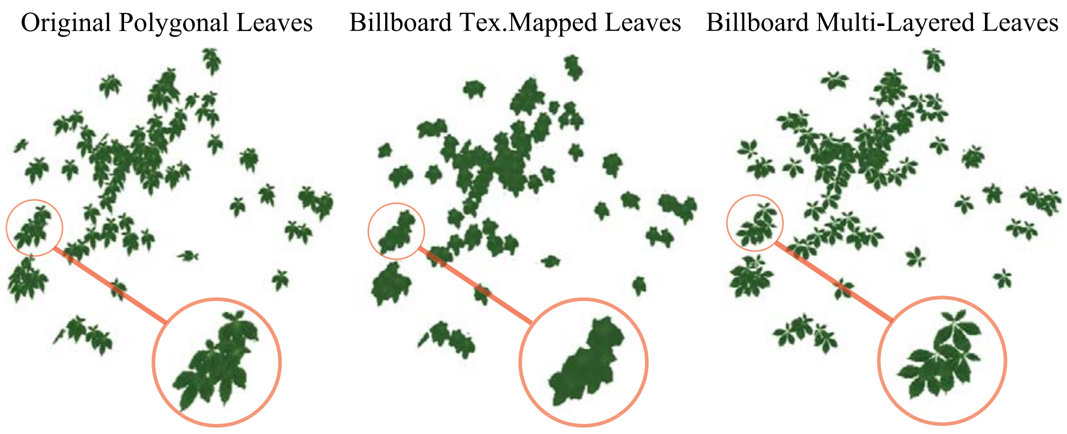

Original Polygonal Leaves
Billboard Tex.Mapped Leaves
Billboard Multi-Layered Leaves

Figure 6.8: Comparison between the group of polygonal leaves (left), with respect to the standard texture mapping result (middle) and the multi-layer method (right).

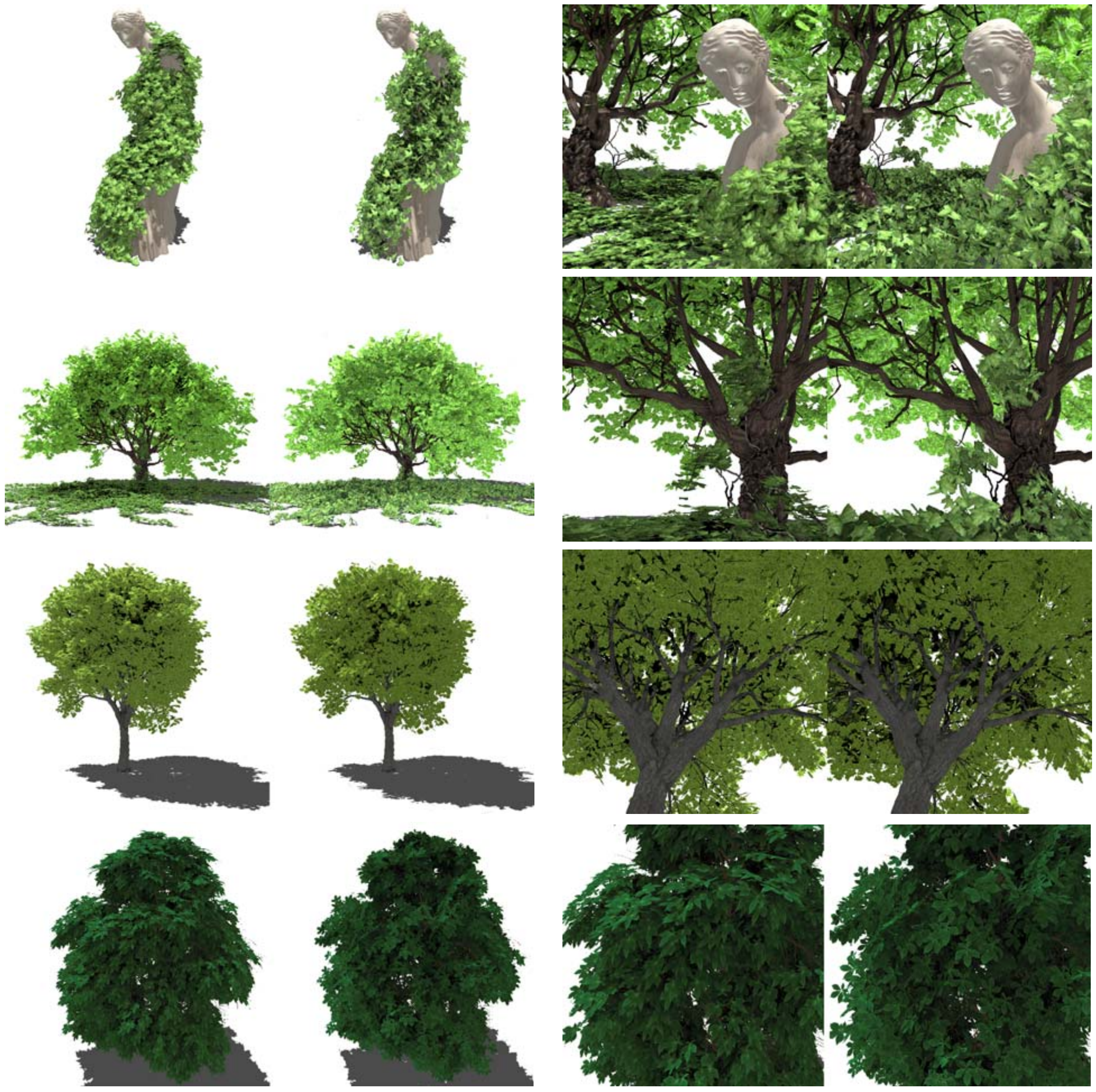


Figure 6.9: Visual comparisons between the polygonal plant/trees representation on the right and the multi-layered billboard clouds on the left. Including dynamic lighting and an precomputed ambient term. **First row:** Ivy plant composed by 8234 leaves simplified representation with 835 multi-layered billboards. **Second row:** Oak tree with 23660 leaves and a simplification with 194 multi-layered billboards, in the groud there is an ivy plant with 18274 leaves simplified to 359 billboards. **Third row:** Cork tree with 20144 leaaves simplified to 121 multi-layered billboards. **Fourth row:** Chestnut tree with 11291 leaves simplified to 278 multi-layered billboards.

original geometry if it has a moderate polygon count. However, if we have scenes with more than tens of trees, we obtain a noticeable performance gain with our approach as can be seen in table 6.3. And the evaluated scenes are shown in figures 6.10 and 6.11.

Trees	**Polygonal**	**Billboard**
1	180	80
50	47	46
100	15	39
1000	5	28
10000	n/a	20

Table 6.3: FPS for varying numbers of trees, for the original tree and the new method. Distances range from very close views to very far trees (covering only about 32^2 pixels).

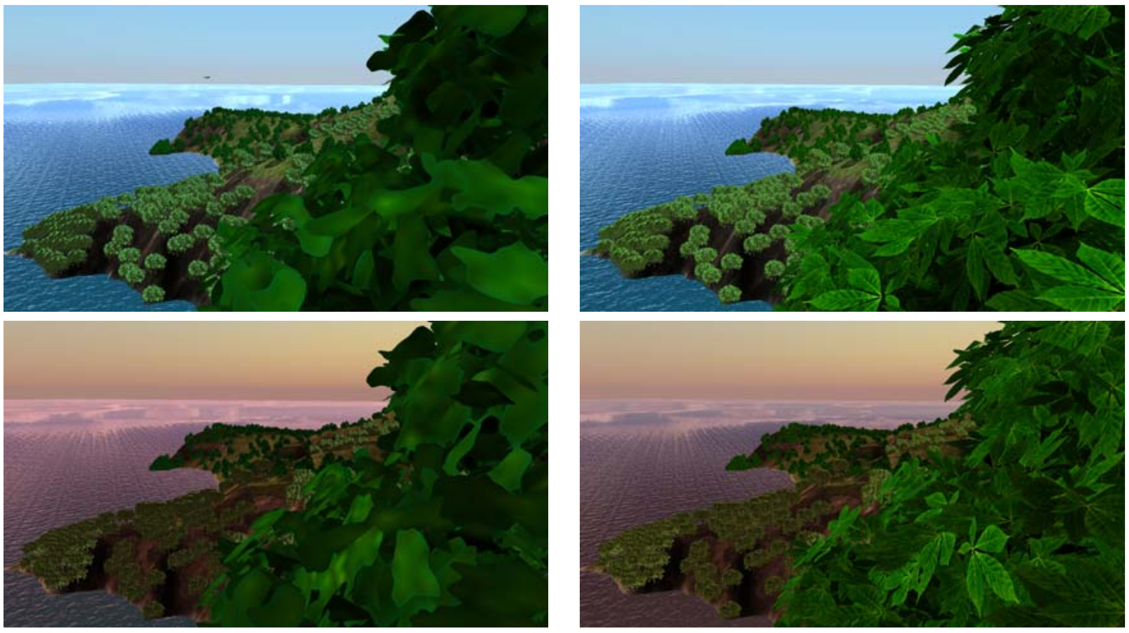


Figure 6.10: A complex sample scene under different lighting conditions. Left column shows the scene with the classic method [BCF+05, FUM05] using the same memory footprint as the images on the right, using the new method.

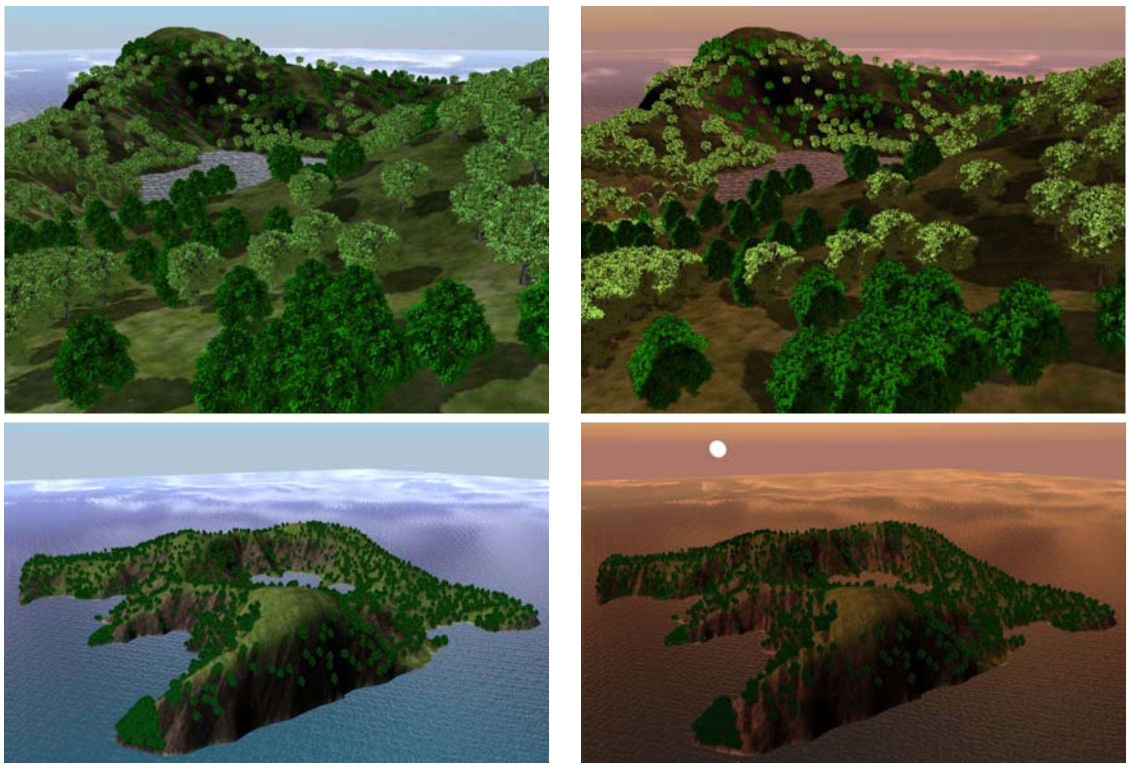


Figure 6.11: Two complex sample scene under different lighting conditions, using multi-layered billboard cloud for the leaves and progressive level of detail for the trunks.

Chapter 7

Conclusions and future work

Image-based rendering is one of the oldest approaches in generating computerized images, but had its rise in three-dimensional computer graphics only over the last years, as increasing hardware power made texture mapping inexpensive. We have shown that image-based rendering can be used to generate huge forests at interactive frame rates.

The new method is the combination of a simple but effective billboard cloud simplification method with an indirect texturing technique, in combination with a texture used as a hash for fast indexing and retrieval, and a layered representation of the overlapping leaf fragments. One of the key points of our method is the accurate preprocessing that generates the layered representation. For instance, previous methods require extremely large texture memory space to preserve small details in the leaves, while the presented method keeps a low memory footprint and needs only a few extra texture accesses. As an example, the chestnut model, shown in section 6, would require about 246MB of texture space to achieve the same visual quality as our method, which only needs 3.7MB. For comparison, the original polygonal model for the leaves needs 23Mb.

Level-of-detail techniques have specifically been developed, not only at the geometric level (reducing the number of billboards and simplifying the geometry of the trunk and branches), but also at the shader level, drastically reducing the computational costs associated with rendering trees far from the observer. Another of the key points of our method is that is able to provide a quasi-continuous smooth transition between the close and medium distance views without the parallax problem of the LOD techniques available in several of the image-based methods [Jak00, MNP01, LCV04, FUM05, BCF+05] when switching from the image-based representation to the polygonal representation for the closest views, losing the performance benefit of using the image-based representations.

It is important to note that detail in the leaves is preserved even at very short distances, something that was not done before with billboard clouds. These results hold even for any kind of trees or shrub independently of the foliage density. In addition, our technique is highly suitable for instancing, because the use of few polygons in the billboard cloud enable us to set thousands of instances in large natural scenes.

The visualization of the trees is enriched with a realistic lighting model including direct lighting computations, soft-shadows (by using a modified shadow map algorithm) and an ambient occlusion term.

Furthermore, the multi-layer indirect texturing technique could be used with any of the existing methods for billboard clouds generation, such as [FUM05, BCF+05, SPE05]. Those methods were limited to use a reduced number of billboards, in order to not increasing the texture memory.

However, some considerations should be taken into account, because the use of thousands of billboards for one single tree won't be effective at all. The main reason for this restriction is that in order to render the billboard cloud with proper depth-sortng, is only possible enabling the alpha test set to pass opaque texels only. In this situation, having too many billboards for a single tree, in the same region, we will exceed the fragments fill-rate of the graphics hardware because of the dense depth complexity has to be evaluated.

Include animation in our current billboard representation has some restrictions in the hierarchical structure motion expected in trees and plants, of all the leaves with respect to their branches, and new strategies must be found.

In addition, it must be mentioned that this algorithm is resolution dependant: depending on the screen-size of the trees and the screen resolution, different frame rates would be obtained. Nevertheless, as mentioned in chapter 6, we are able to present about ten thousand trees in a complex scene in resolutions of 1280×1024 with an acceptable frame rate.

We should also take into account that the use of a progressive geometric level of detail for the trunk is not providing the best performance. In close views of dense natural scenes, the number of polygons for the branches exceed current graphics hardware for interactive frame rates, therefore we had to switch to a simpler representation sometimes even closer to the observer.

7.1 Future work

We should take also into account that our billboard representation cannot be applied effectively to special kinds of highly curved leaves without some compromises, like for example in palm trees. Therefore, it can be interesting to find simple curved surfaces, able to contain one or multi-layers of foliage. This approach can be good to simulate highly curved leaves representations or special foliage placed around curved surfaces that would require considerable billboard planes to reach a good approximation, as can be in the case of ivy leaves around a the sculpture or the tree trunk, shown in figure 6.9.

In the case of animation, we will study the use of weigheted billboards with respect to their respective branches, in order to provide an approximate bending effect.

One line of promising immediate research is to change the progressive level-of-detail technique described in section 5.1.1 to a continuous LoD technique. This technique would perform an interpolation between shader levels, resulting in a smoother switch between LoD levels.

Another interesting line is to extend the presented technique to include small branches, which would reduce the complexity of the geometry of the trunk, which then would be more easily handled with more traditional techniques. Finally, the incorporation of branch movements seems an interesting challenge to be modelled with the presented algorithm, as the Hash Texture would be slow to regenerate.

In addition, in our current implementation, the last level of the shader LOD deals with a simpler shader using only standard texture mapping for the leaves. The specific distance where this shader change is applied by an user-defined factor, being useful to tune the trade of between visual quality and performance. However, it can be feasible to implement an automatic method that under these quality and performance constraints, find the accurate distance to switch from the multi-layer shader to the texture mapping one, and the adjustment of the shader LOD depending on the tree model automatically.

Finally, an interesting extension would be find a fast ambient occlusion interactive computation addressed to billboard trees representations, because their low polygonal count and compact representation can made feasible some interactive global illumination computations.

Bibliography

[AMH02] Thomas Akenine-Moller and Eric Haines, editors. *Real-Time Rendering, second edition*. A. K. Peters, Ltd., 2002. 19

[BCF$^+$05] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum*, 24(3):507–516, 2005. 10, 12, 14, 29, 30, 43, 66, 67, 93

[DCSD02] Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 219–226, Washington, DC, USA, 2002. IEEE Computer Society. 9, 23, 24, 34

[DDSD03] Xavier Décoret, Frédo Durand, François X. Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. *ACM Trans. Graph.*, 22(3):689–696, 2003. 27, 29, 38, 41

[DN04] Philippe Decaudin and Fabrice Neyret. Rendering forest scenes in real-time. In *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*, pages 93–102, june 2004. 10, 27, 28, 29, 37

[EAS06] Bionatics, easynat 2.5, 2006. `http://www.bionatics.com/home.html`. 17

[Eve99] C. Everitt. Interactive order-independent transparency, 1999. White paper, NVIDIA Corporation. 11, 13, 46, 47, 81

[FUM05] Anton L. Fuhrmann, Eike Umlauf, and Stephan Mantler. Extreme model simplification for forest rendering. In *Eurographics Workshop on Natural Phenomena*, 2005. 10, 12, 14, 29, 38, 41, 43, 59, 66, 67, 93

[GFS07] Francisco González, Miquel Feixas, and Mateu Sbert. An information-theoretic ambient occlusion. In *International Symposium on Computational Aesthetics*, 2007. 56

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 10, 12, 37, 76

[Gla05] R. Steven Glanville. Texture bombing. In Randima Fernando, editor, *GPU Gems 1*, pages 323–338. Addison-Wesley, October 2005. 10, 32

[GMN05] Guillaume Gilet, Alexandre Meyer, and Fabrice Neyret. Point-based rendering of trees. In P. Poulin E. Galin, editor, *Eurographics Workshop on Natural Phenomena*, 2005. 9, 23, 24

[GPSSK07] Ismael García, Gustavo Patow, Mateu Sbert, and László Szirmay-Kalos. Multi-layered indirect texturing for tree rendering. In S. Mérillou D. Ebert, editor, *Eurographics Workshop on Natural Phenomena*, 2007. 35

[GSSK05a] Ismael García, Mateu Sbert, , and László Szirmay-Kalos. Tree rendering with billboard clouds. In *In Proceedings of Third Hungarian Conference on Computer Graphics and Geometry*, pages 9–15, 2005. 35

[GSSK05b] Ismael García, Mateu Sbert, and László Szirmay-Kalos. Leaf cluster impostors for tree rendering with parallax. In *In Proceedings of EG Short Presentations 2005*, pages 69–72, 2005. 11, 12, 35, 38, 43, 44, 79

[HKW07] Ralf Habel, Alexander Kusternig, and Michael Wimmer. Physically based real-time translucency for leaves. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, 2007. 10, 31, 32

[Hop96] Hugues Hoppe. Progressive meshes. *ACM Computer Graphics*, 30(Annual Conference Series):99–108, 1996. 37, 57

[HPAD06] Kyle Hegeman, Simon Premoze, Michael Ashikhmin, and George Drettakis. Approximate ambient occlusion for trees. In C. Sequin and M. Olano, editors, *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM SIGGRAPH, March 2006. 10, 31

[Jak00] A. Jakulin. Interactive vegetation rendering with slicing and blending. In A. de Sousa and J.C. Torres, editors, *Proc. Eurographics 2000 (Short Presentations)*. Eurographics, August 2000. 9, 25, 26, 38, 43, 67

[Lan02] H. Landis. Production-ready global illumination. In *ACM-SIGGRAPH Siggraph '02 Course Notes*. ACM SIGGRAPH, ACM Press, 2002. 31

[LCV03] Javier Lluch, Emilio Camahort, and Roberto Vivó. Procedural multiresolution for plant and tree rendering. In *AFRIGRAPH '03: Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa*, pages 31–38, New York, NY, USA, 2003. ACM Press. 9, 20, 21

[LCV04] Javier Lluch, Emilio Camahort, and Roberto Vivó. An image-based multiresolution model for interactive foliage rendering. In *Winter School of Computer Graphics*, 2004. 9, 26, 43, 67

[LHN05] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Texture sprites: Texture elements splatted on surfaces. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics (I3D)*. ACM SIGGRAPH, ACM Press, April 2005. 10, 32, 33

[LRMDM06] C. Linz, A. Reche-Martinez, G. Drettakis, and M. Magnor. Effective multi-resolution rendering and texture compression for captured volumetric trees. In *Game-On 2006*, pages 16–21, 2006. 27, 37

[LVF$^+$01] Javier Lluch, Maria J. Vicent, S. Fernandez, Carlos Monserrat, and Roberto Vivó. The modelling of branched structures using a single polygonal mesh. In *VIIP*, pages 203–207, 2001. 9, 20, 22

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *In Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, page 281–297, 1967. 39

[MDK99] Nelson Max, Oliver Deussen, and Brett Keating. Hierarchical image-based rendering using texture mapping hardware. In *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*, pages 57–62. Eurographics, Dani Lischinski and Gregory Ward Larson (eds.), 1999. 9, 25

[MFI97] Dana Marshall, Donald S. Fussell, and A. T. III, Campbell. Multiresolution rendering of complex botanical scenes. In *Proceedings of the conference on Graphics interface '97*, pages 97–104, Toronto, Ont., Canada, Canada, 1997. Canadian Information Processing Society. 9, 20, 34

[MN98] Alexandre Meyer and Fabrice Neyret. Interactive volumetric textures. In George Drettakis and Nelson Max, editors, *Rendering Techniques (Eurographics Workshop on Rendering - EGSR)*, pages 157–168, New York City, NY, Jul 1998. Eurographics, Springer Wein. ISBN. 26, 34

[MNP01] Alexandre Meyer, Fabrice Neyret, and Pierre Poulin. Interactive rendering of trees with shading and shadows. In *Rendering Techniques (Eurographics Workshop on Rendering - EGSR)*, Jul 2001. 10, 27, 43, 67

[MO95] N. Max and K. Ohsaki. Rendering trees from precomputed z-buffer views, 1995. 9, 25

[Ney98] Fabrice Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. In *IEEE Transactions on Visualization and Computer Graphics 4, 1*, pages 55–70, 1998. 27

[OL98] Marc Olano and Anselmo Lastra. A shading language on graphics hardware: the pixelflow shading system. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 159–168, New York, NY, USA, 1998. ACM Press. 43

[Opp86] Peter E. Oppenheimer. Real time design and animation of fractal plants and trees. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 55–64, New York, NY, USA, 1986. ACM Press. 9, 19, 20

[RB85] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 313–322, New York, NY, USA, 1985. ACM Press. 9, 22

[RCRB03] I. Remolar, M. Chover, J. Ribelles, and O. Belmonte. View-dependent multiresolution model for foliage, 2003. 9, 20, 21

[Rem05] Inmaculada Remolar. *Visualización interactiva de especies vegetales*. PhD thesis, 2005. Tesis Doctoral, Universitat Jaume I. 37

[RH94] John Rohlf and James Helman. Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394, New York, NY, USA, 1994. ACM Press. 38

[RMMD04] Alex Reche-Martinez, Ignacio Martin, and George Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Trans. Graph.*, 23(3):720–727, 2004. 10, 27, 28, 37

[SPE05] Speedtree, interactive data visualization inc., 2005. `http://www.idvinc.com/speedtree`. 10, 29, 30, 38, 67

[WEI03] E. WEISSTEIN. World of mathematics., 2003. `http://mathworld.wolfram.com/Eigenvector.html`. 41

[WP95] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128, New York, NY, USA, 1995. ACM Press. 9, 22, 23

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering 2004)*, pages 143–151, 2004. 55

[WWD$^+$05] 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. 10, 31, 32

[Xfr06] Greenworks: Organic software, 2006. `http://www.greenworks.de/`. 15, 17, 59

Color Plates

Color Plates

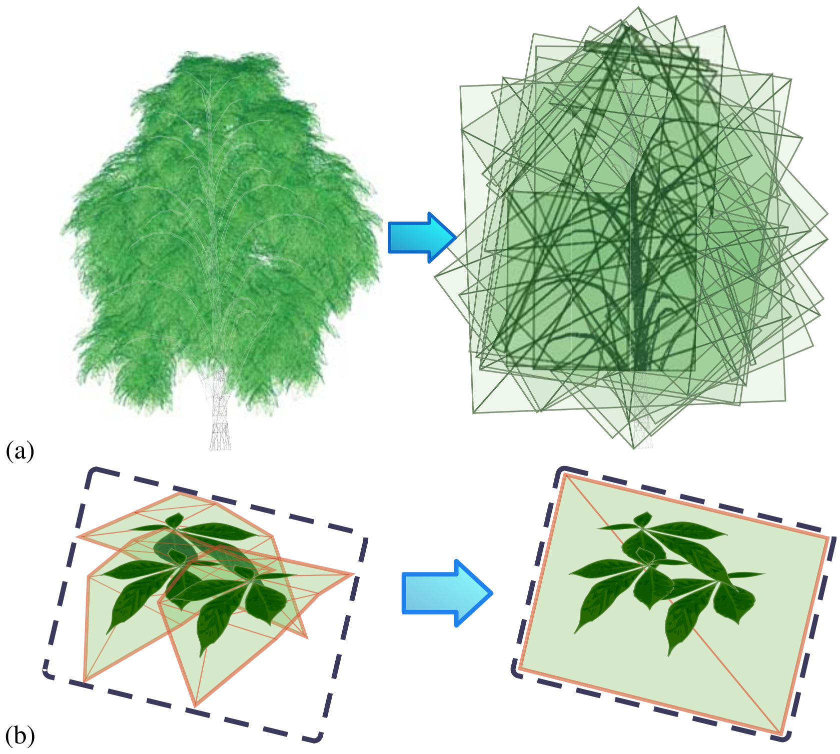


Figure 1.1: Replacing large groups of leaves by semi-transparent quadrilaterals with a texture map. (a) The polygonal leaves are replaced by a billboard cloud. (b) Each group of leaves is replaced by a billboard.

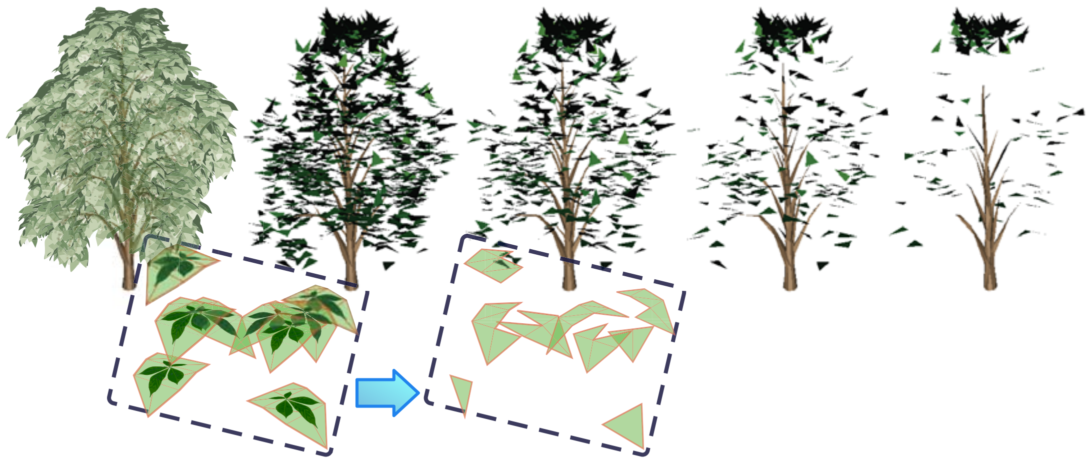


Figure 3.1: Example of geometry simplification using Garland's et al. [GH97] Quadric Error Metric method. This chestnut tree have been simplified in different multiresolution meshes. As can be seen, the trunk and branches are well preserved, compared to the original model. But the leaves are prunned drastically in each level losing their original dense appearence.

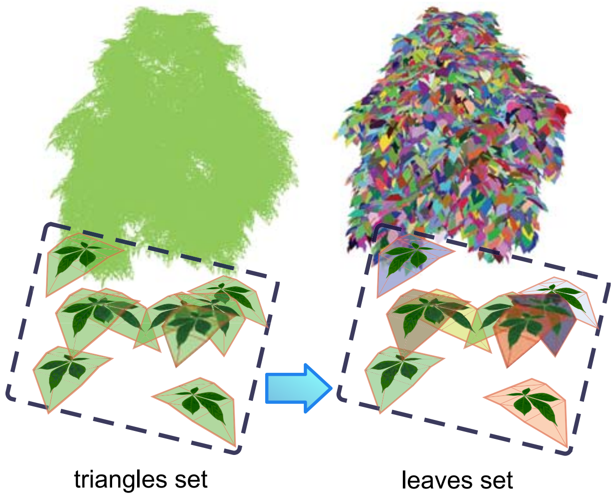


Figure 3.2: A polygon connectivity check is performed to find the leaf components in the triangle soup.

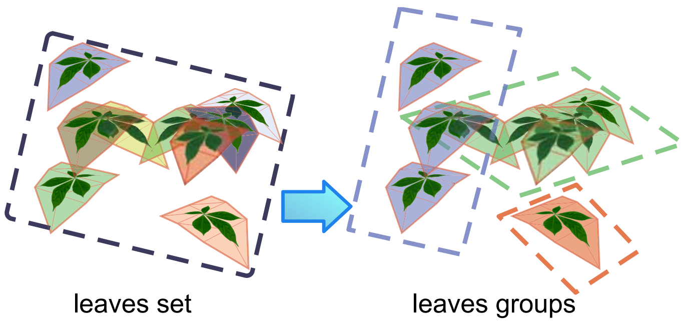


Figure 3.3: The clustering strategy finds groups of leaves that are approxyimatelly around polygon planes.

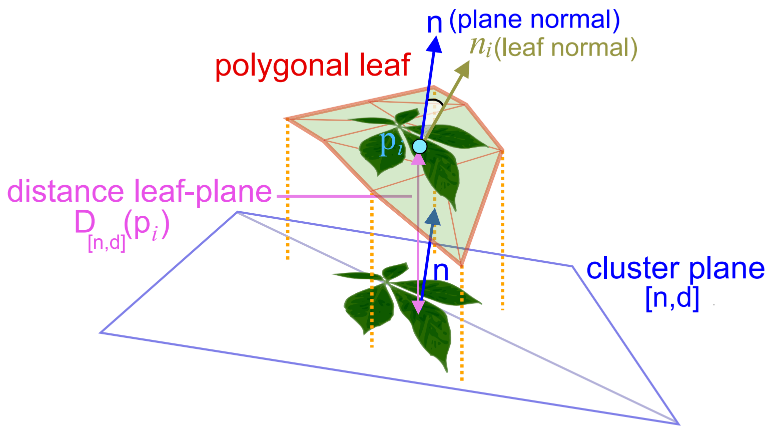


Figure 3.4: Error measure between a plane and a leaf for the K-Means clustering method.

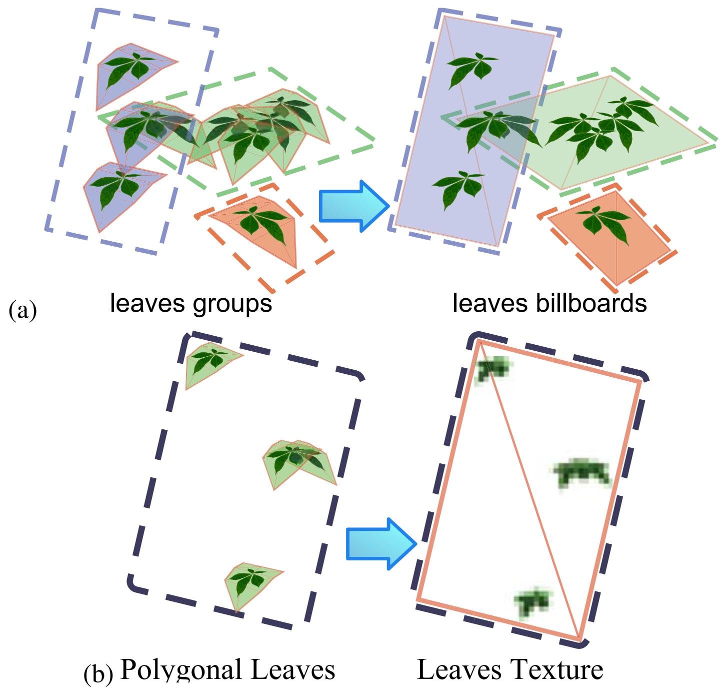


Figure 3.5: (a) Leaf groups are used to create the billboard planes . (b) A single texture map contain all leaves of one group.

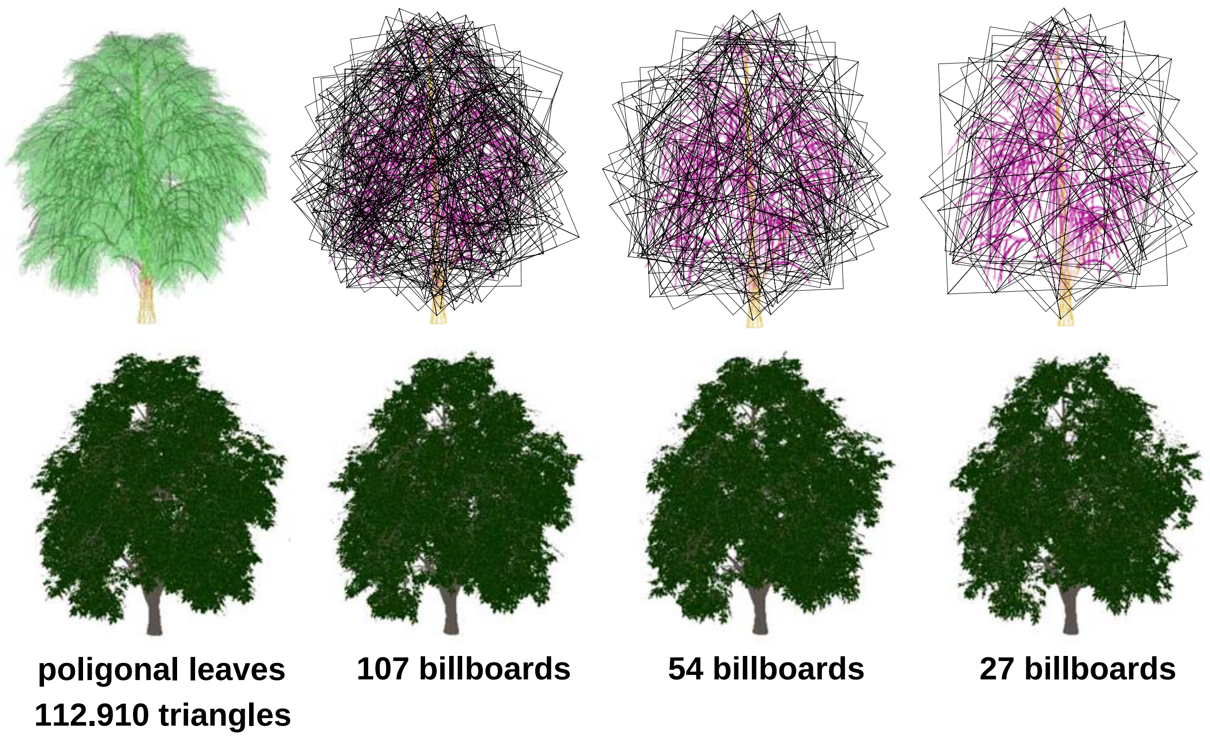


Figure 3.6: Chestnut (*Castanea Sativa*). Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

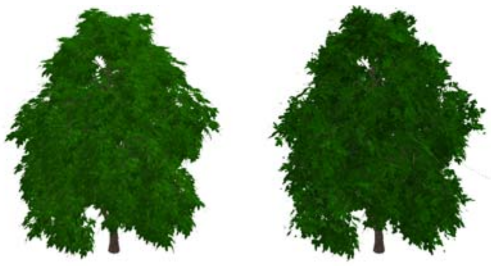
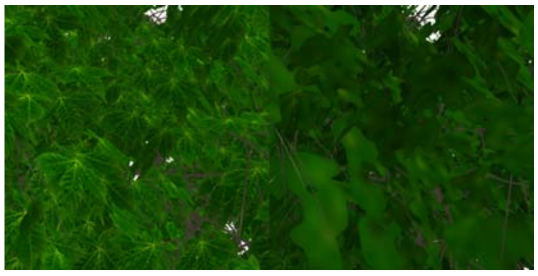
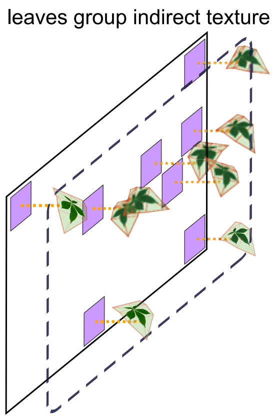
(a)  (b) 

Figure 4.1: Chestnut (*Castanea Sativa*). Visual comparisons of the original polygonal tree (right) and a billboard cloud rendered with texture maps generated at 2048×2048 (left). (a) Shows a comparison of both representations from a far observer distance. (a) Shows a comparison of both representations from a closer observer distance, the billboards texture is not detailed enough, failing to replace the original tree.

(a) 

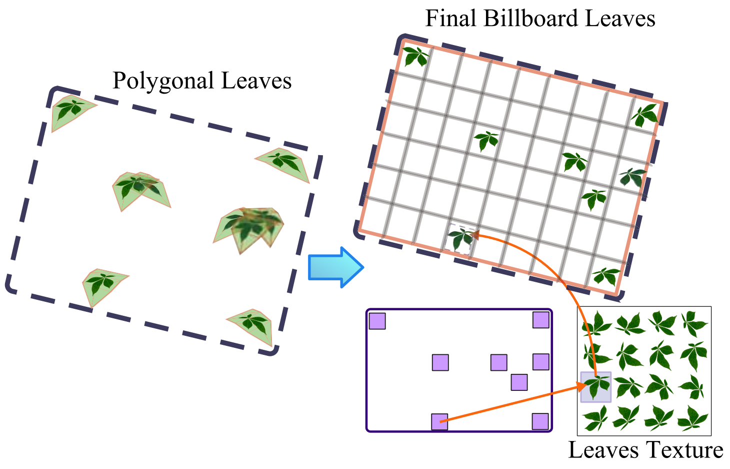
(b) 

Figure 4.2: (a) Preprocessing step where leaves positions are stored in a texture map. (b) In the interactive visualization, the indirect texturing technique replaces the polygonal leaves by the billboard using the values in the texels of the indirect texture to fetch the corresponding leaf color image.

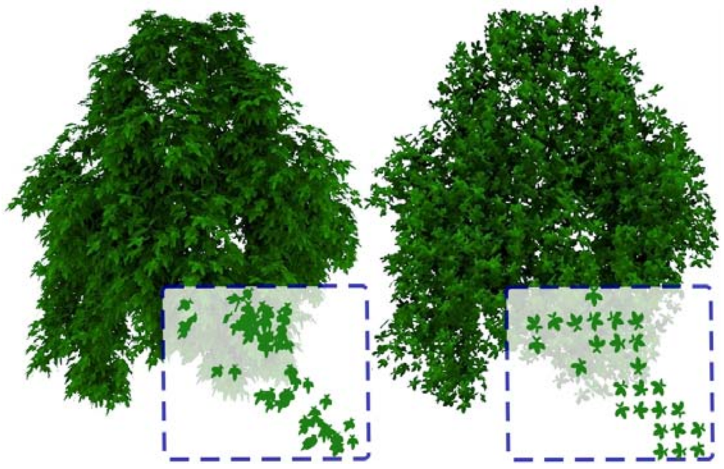


Figure 4.3: Comparison and close up of the polygonal tree and billboards with simple indirect texturing [GSSK05b].

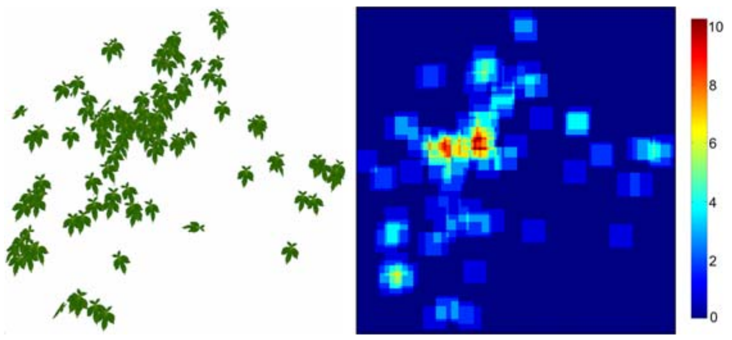


Figure 4.4: Group of 116 polygonal leaves clustered by a single billboard plane. The leaves distribution (right) leads to a up to 10 leaf fragment overlaps.

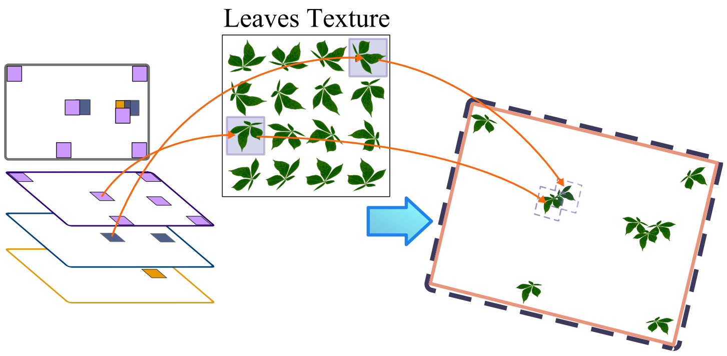


Figure 4.5: Overlapped leaves must be encoded with a layered indirect texture to be properly rendered.

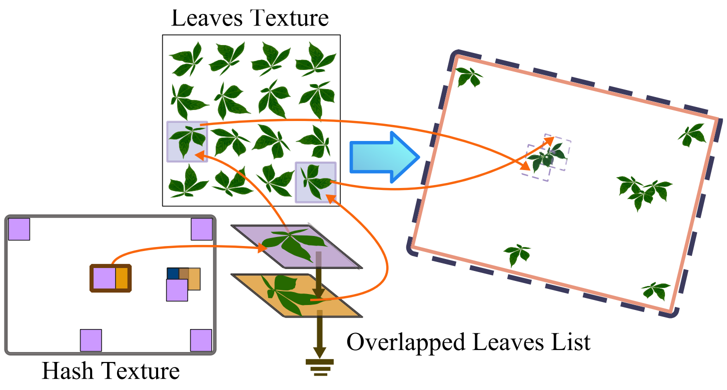


Figure 4.6: Indirect texturing with overlapped leaf fragments lists reduce texture requeriments.

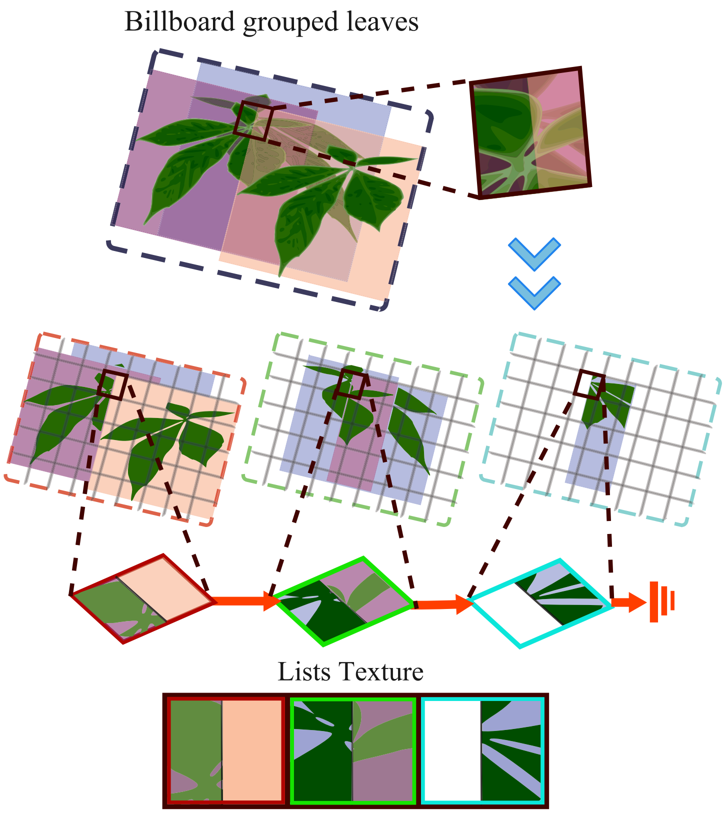


Figure 4.7: Each leaf is processed as a point sprite. The point sprite can appear in more than one grid cell of the Hash Texture. The point sprites are processed with depth peeling [Eve99] to build an overlapping leaf fragments list for each grid cell.

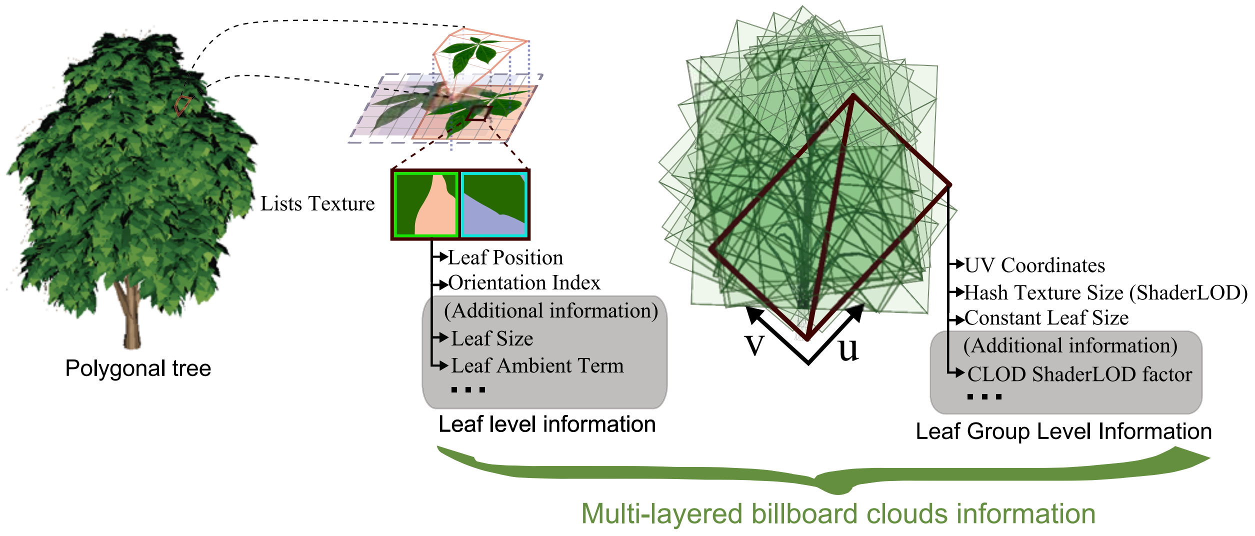


Figure 4.8: In a preprocessing step with the complex polygonal model we store the indirect texturing information in different structures. The leaf level information is stored in the Lists Texture, and the tree or leaf group information is encoded in the billboard cloud mesh.

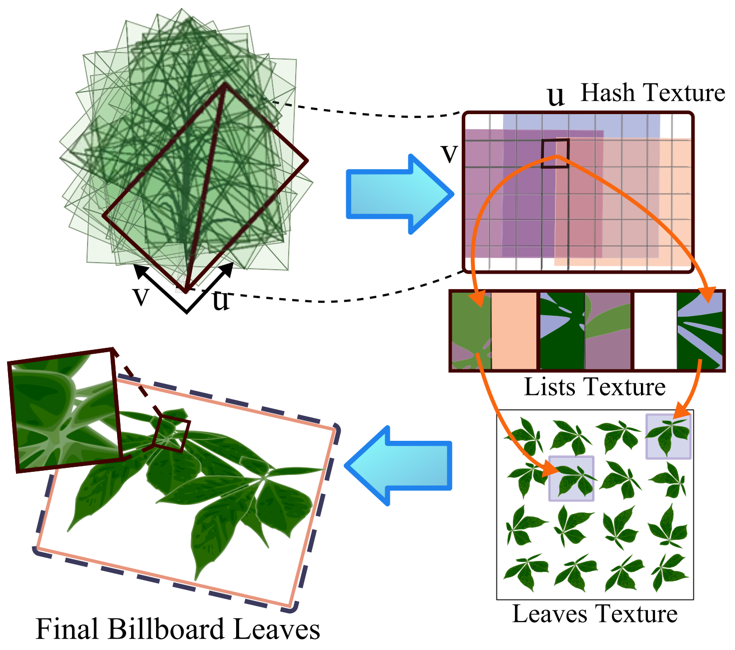


Figure 5.1: Multi-layer indirect texturing evaluation in the single pass fragment shader.

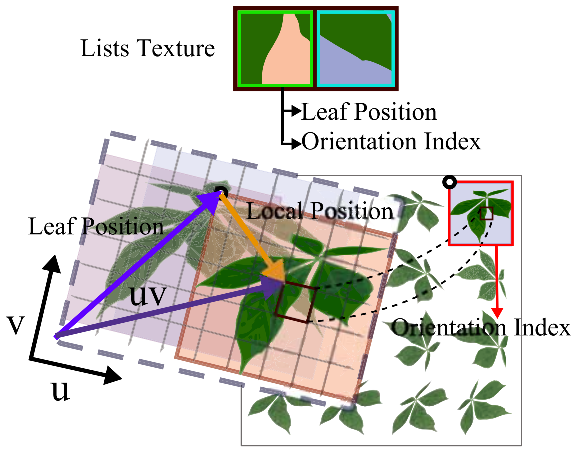


Figure 5.2: Information stored at an entry in the list of leaves for a hash cell.

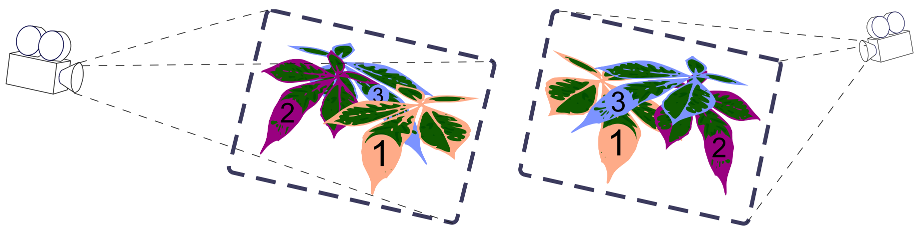


Figure 5.3: Polygonal overlapped leaves from different view directions. Given the left view direction, the leaf *leaf_1* overlaps the *leaf_2* and *leaf_3* ones. On the other hand, from the opposite view direction, the user expects to see the inverse overlapping order, where the *leaf_3* is above the *leaf_2* and *leaf_1* ones.

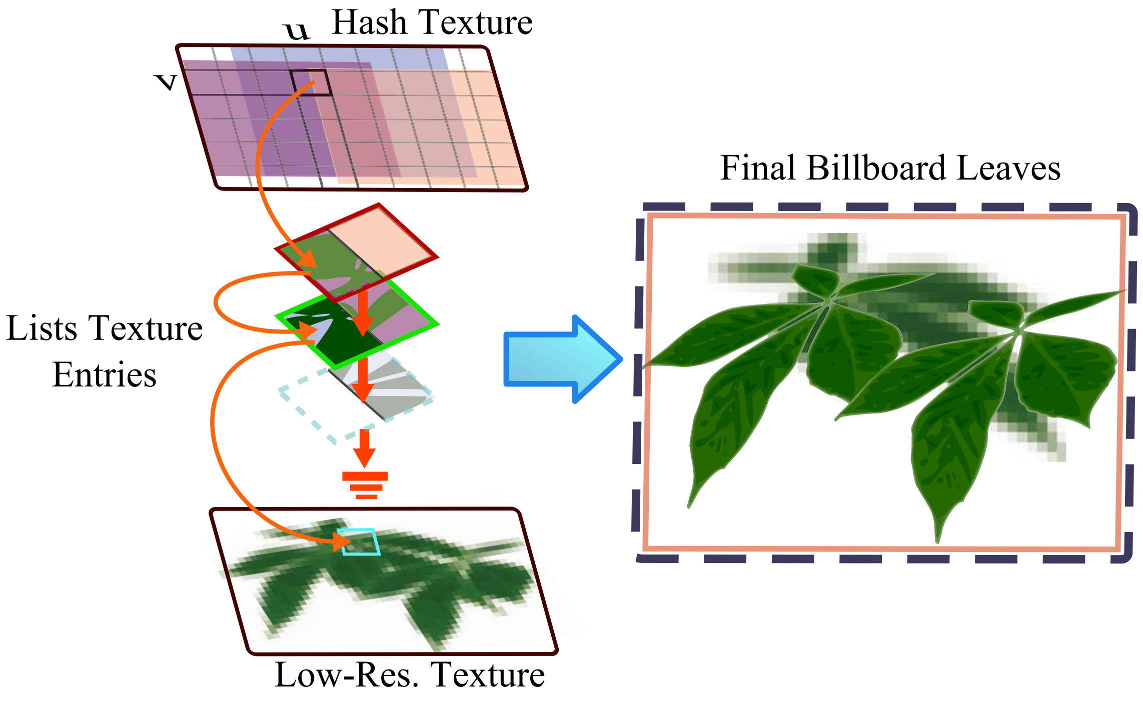
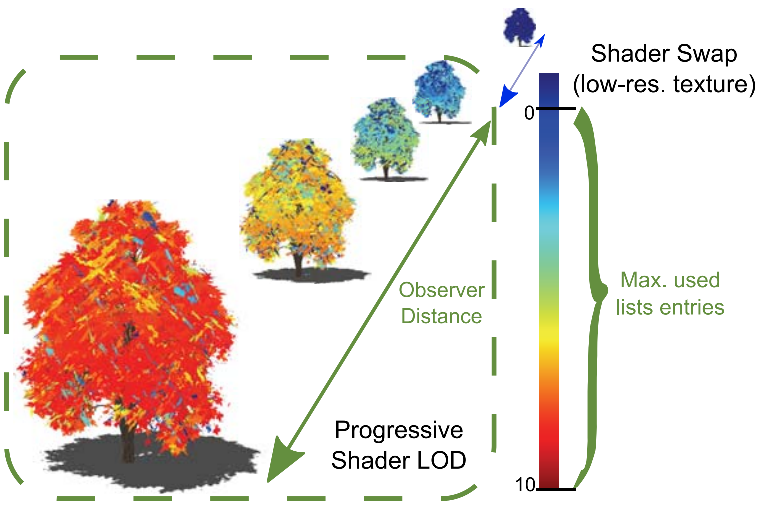
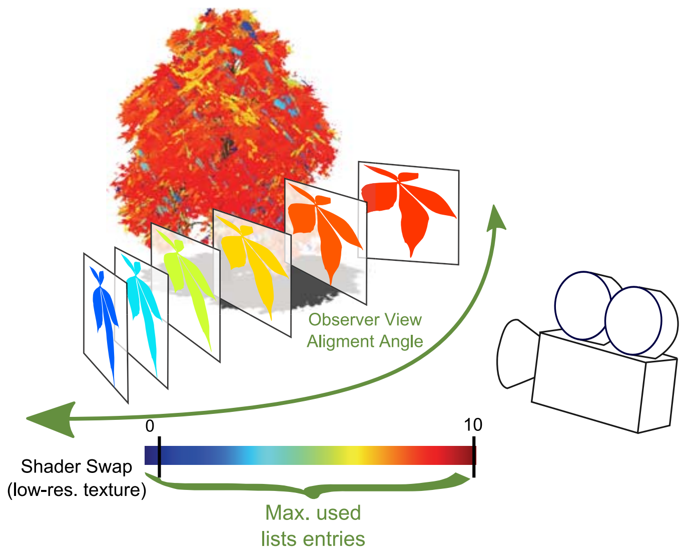


Figure 5.4: As distance increases, the shader-LOD technique evaluates less list entries, and the missing ones are substituted by a low-resolution texture.



(a)



(b)

Figure 5.5: Description of the shader-LOD technique. Color red means the maximum level of detail. From red to blue represent the progressive decreasing level of detail. (a) Shader-LOD reduces the number of evaluated list entries when the distance from the observer increases. And the missing ones are substituted by a low-resolution texture. (b) In addition, when the angle between the billboard plane and the camera plane are almost perpendicular, the number of list entries is also progressively reduced. In both cases (a) and (b), the more red the foliage appeares is the notion that we are evaluating almost all the lists entries with respect to the maximum overlaps of the billboard. The full blue color means fragments that are only evaluating the low-res texture.

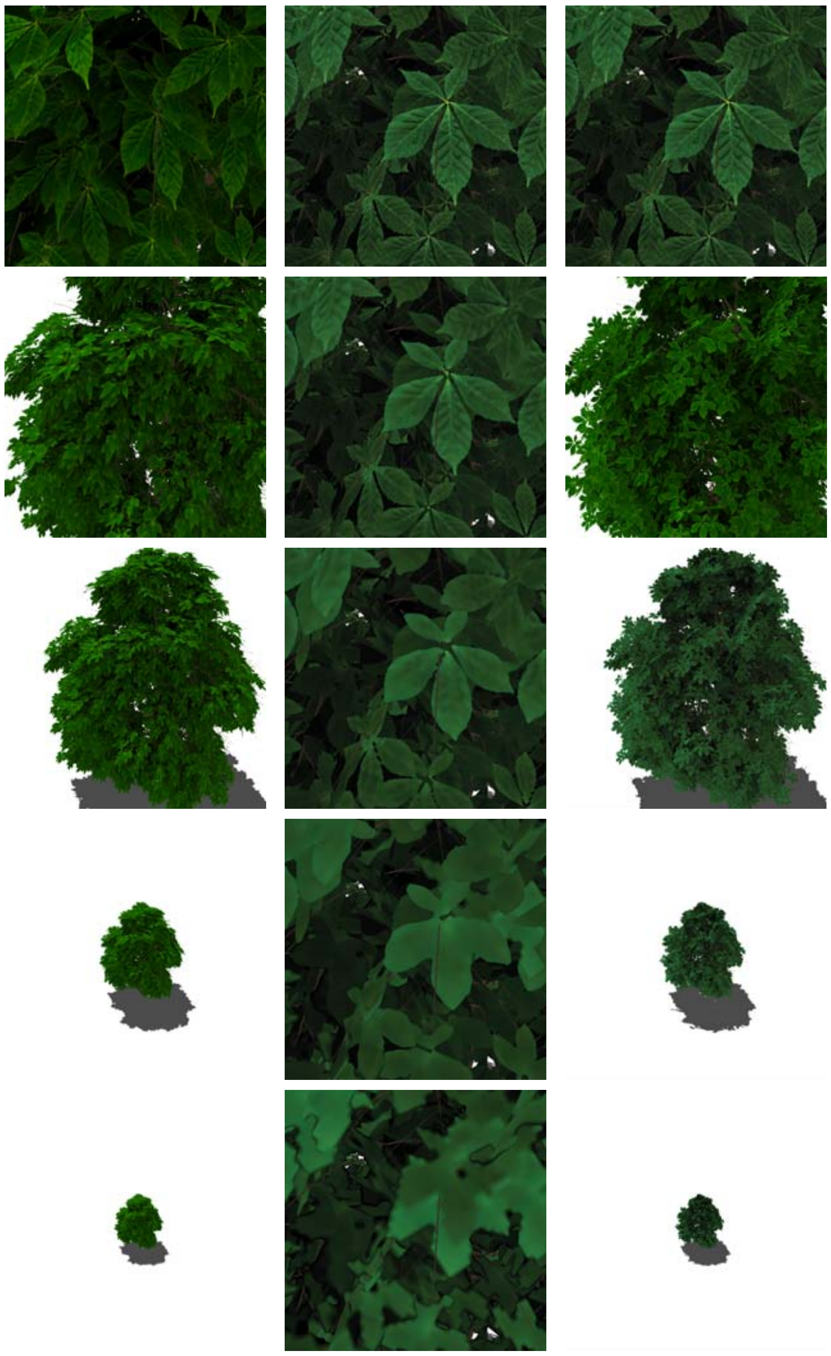


Figure 5.6: Comparison between the polygonal tree on the left and the multi-layer technique on the right, preserving accurate alignment of the leaves. The middle column shows the progressive shader LoD. The small holes are due to slight changes in the leaves alignment.

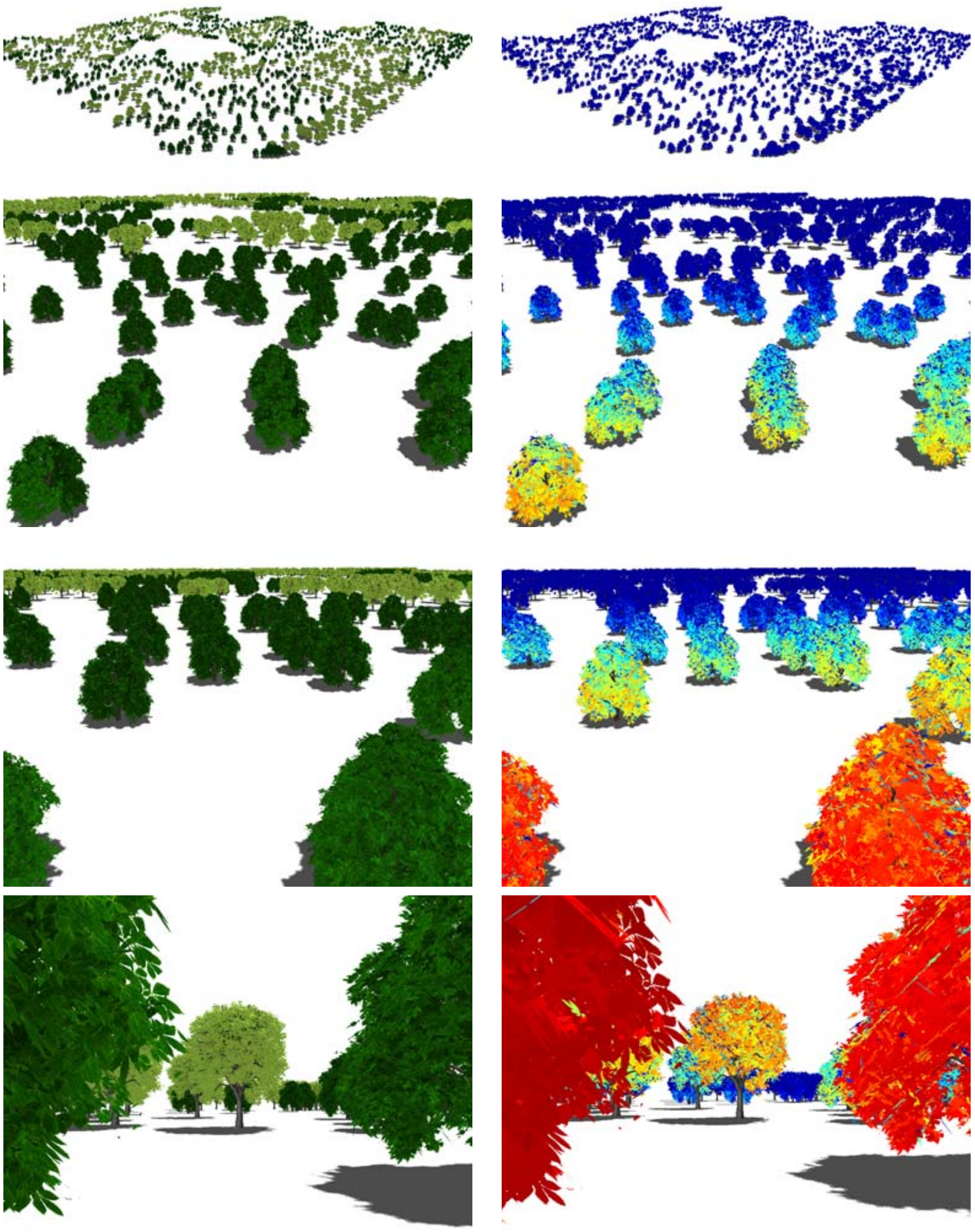


Figure 5.7: Forest scene with 2000 trees. On the left row we show the realistic rendering computing the multi-layer indirect texturing and the lighting model detailed in section 5.2. On the right we show the maximum number of lists entries evaluated for each billboard depending on the observer distance. The furthest views only use the low-res texture, progressively being replaced by high definition leaves, as the observer gets closer to the foliage.

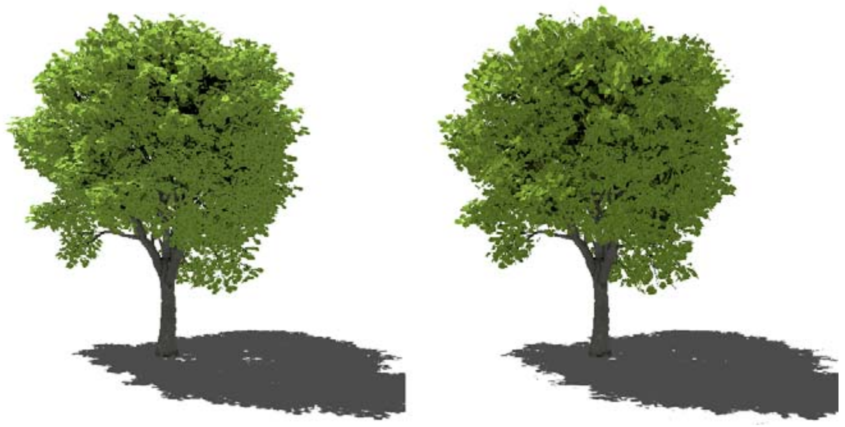


Figure 5.8: Visual comparison: Polygonal cork tree with 20144 leaaves simplified on the left to 121 multi-layered billboards on the right. As can be seen, the lighting and even shadows generated by the polygonal model and the billboard clourd representation are approximately the same.

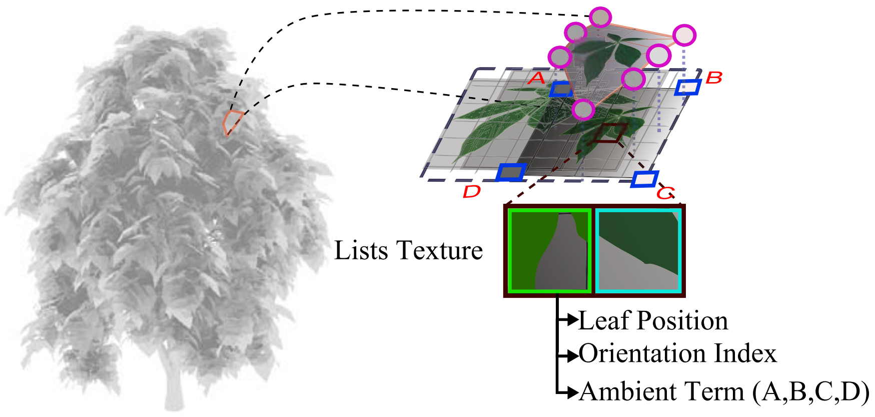


Figure 5.9: Ambient term generation for the multi-layer leaves representation is encoded as a new field of the List Texture at the leaf level. The small squares represent the actual ambient occlusion values at the corresponding vertices.

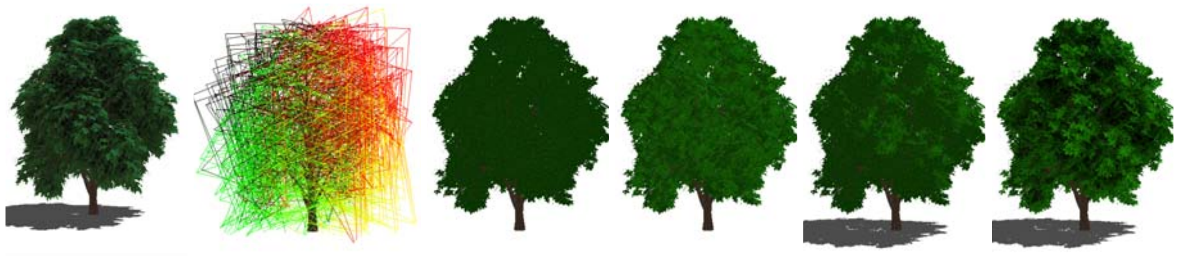
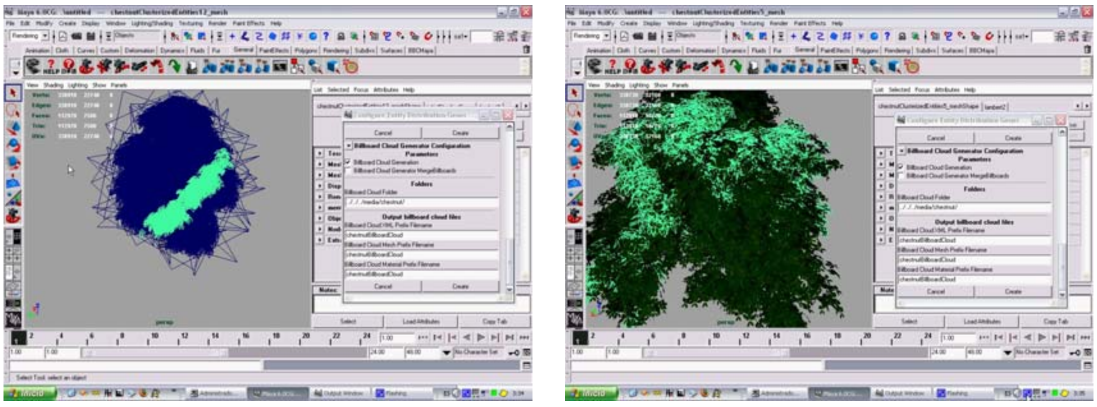


Figure 5.10: Comparison between the polygonal tree and the multi-layer technique. From left to right: Original polygonal tree fully lit, the wire frame billboards, billboards with diffuse lighting, direct lighting, shadows and, finally, the full illumination model including the ambient occlusion term.


Billboard Cloud Generator Configuration

Parameters

[✓] Billboard Cloud Generation

[ ] Billboard Cloud Generator MergeBillboards

Folders

Billboard Cloud Folder: ../../../media/chestnut/

Output billboard cloud files

Billboard Cloud XML Prefix Filename: chestnutBillboardCloud

Billboard Cloud Mesh Prefix Filename: chestnutBillboardCloud

Billboard Cloud Material Prefix Filename: chestnutBillboardCloud

Figure 6.1: Billboard cloud preprocessor Maya plug-in interface.

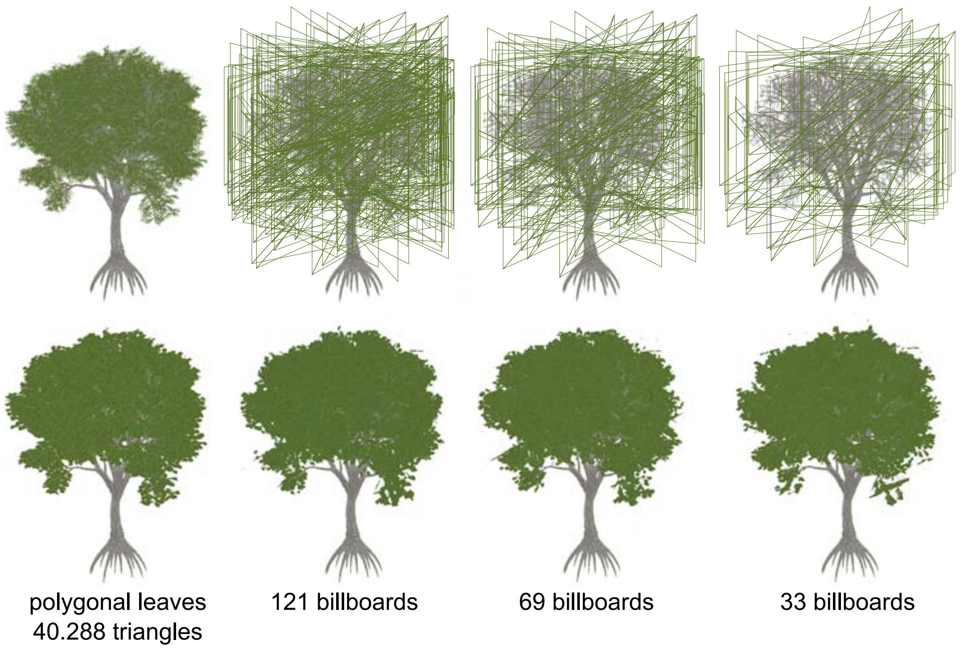


Figure 6.2: Amur Corktree (*Phellodendron amurense var. Japonica*) in adult age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

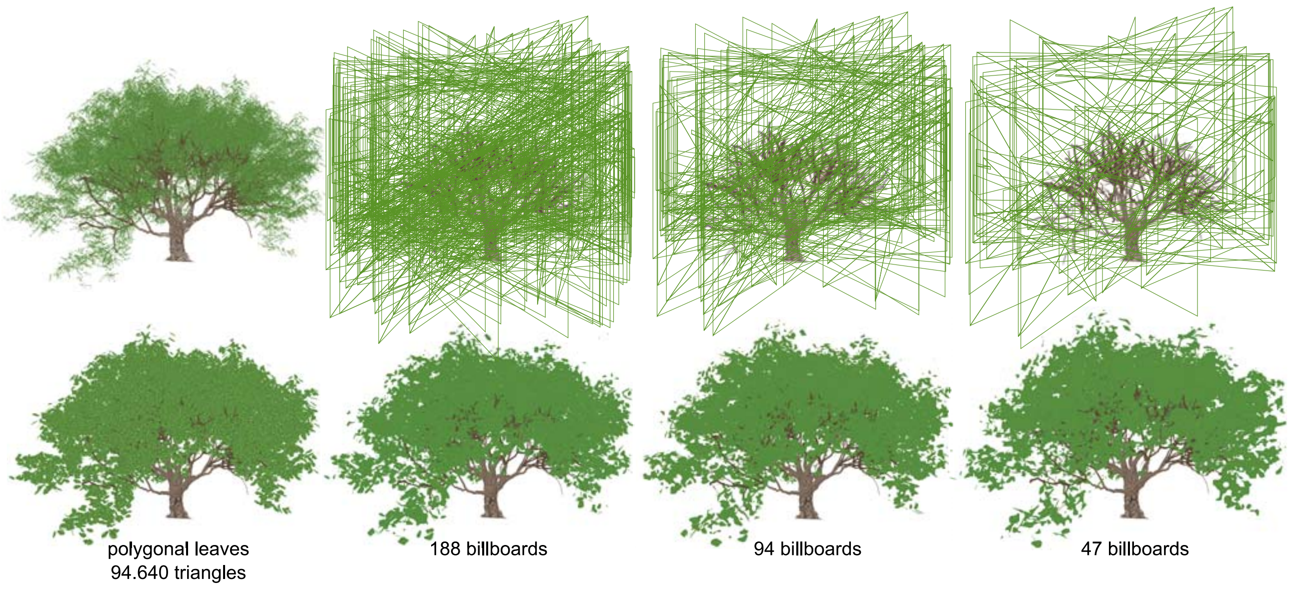


Figure 6.3: Oak (*quercus*) in adult age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

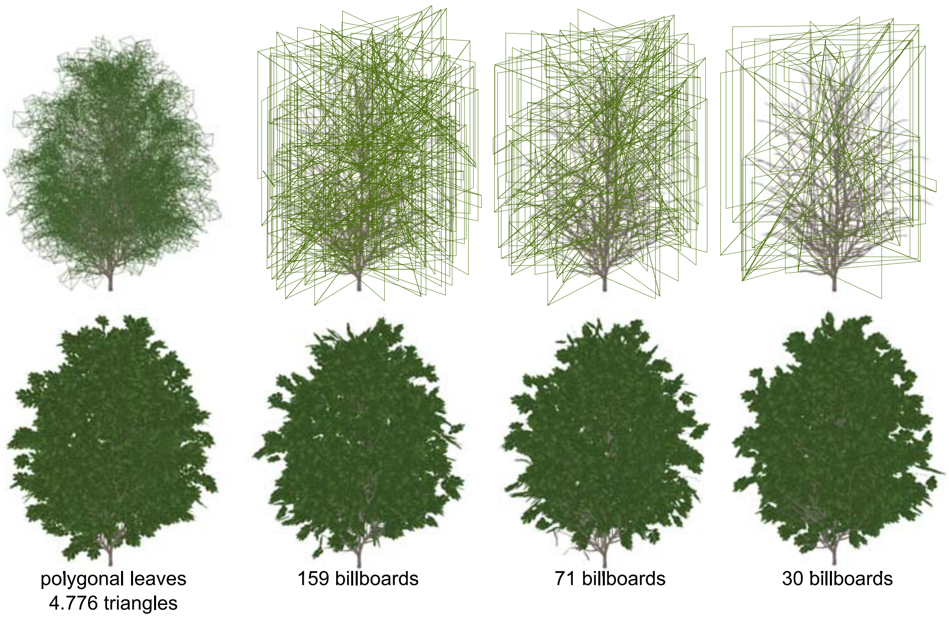


Figure 6.4: Hornbeam (*Carpinus betulus*) in adult age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

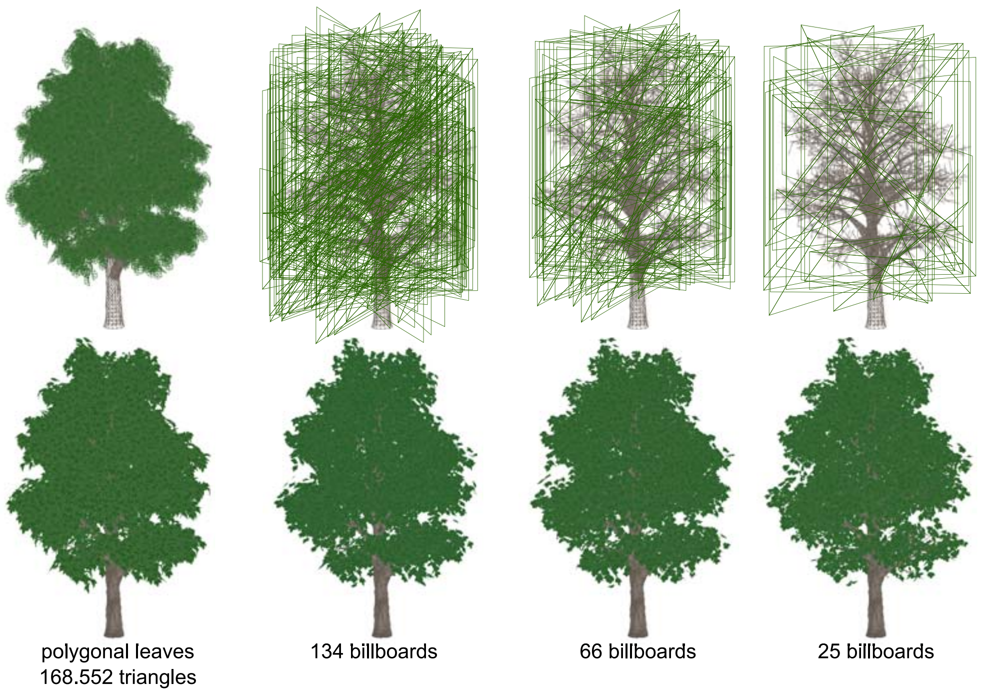


Figure 6.5: Horse chestnut (*Aesculus hippocastanum*) in adult age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

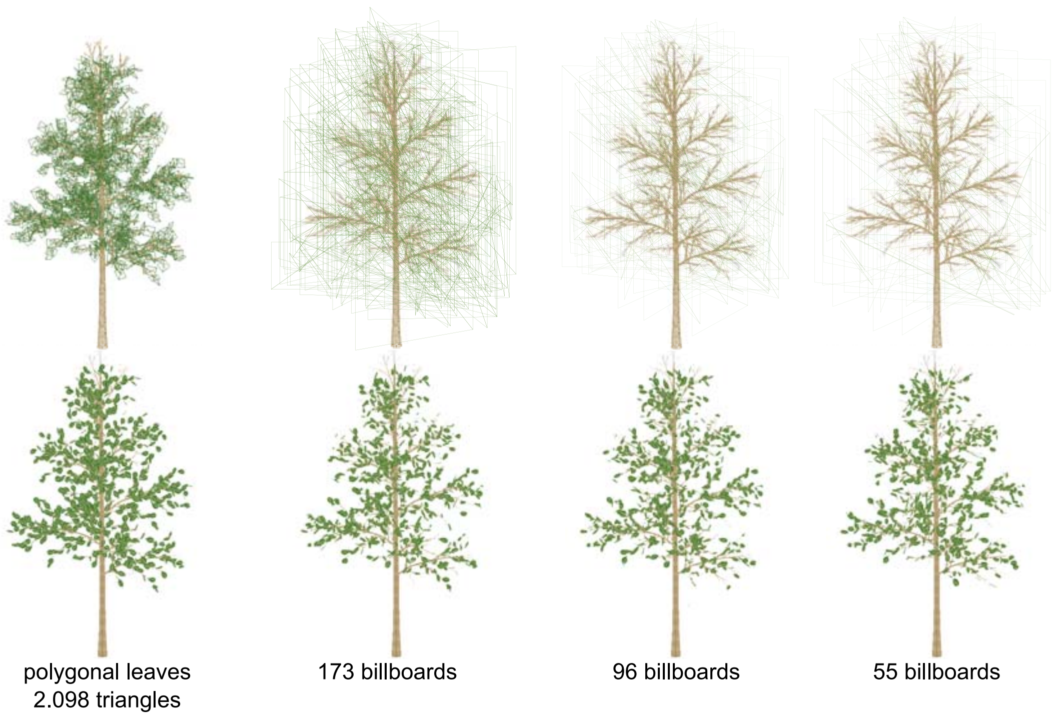


Figure 6.6: Alder tree (*Alnus glutinosa*) in young age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

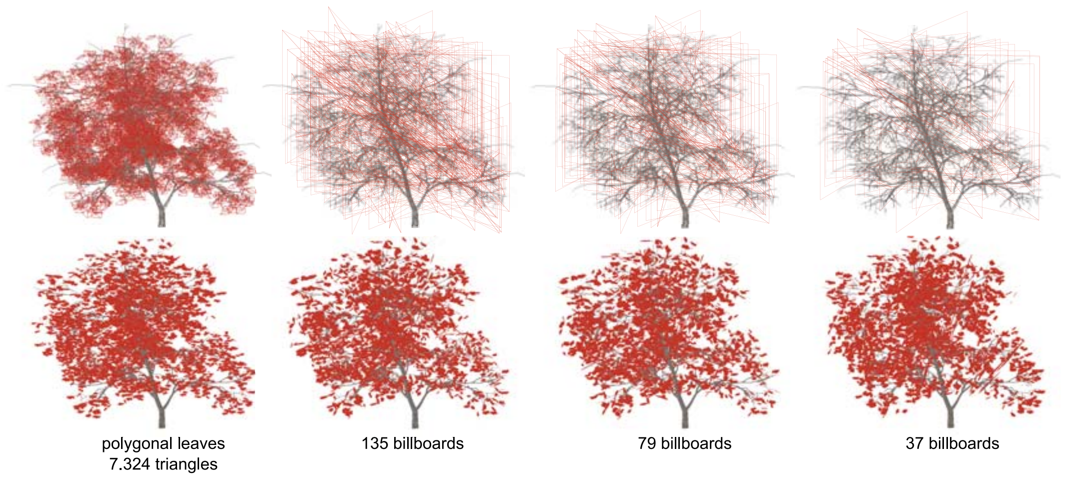


Figure 6.7: Japanese Maple (*Acer palmatum*) in young age. Visual comparisons of the original polygonal tree and the tree rendered with different number of billboards.

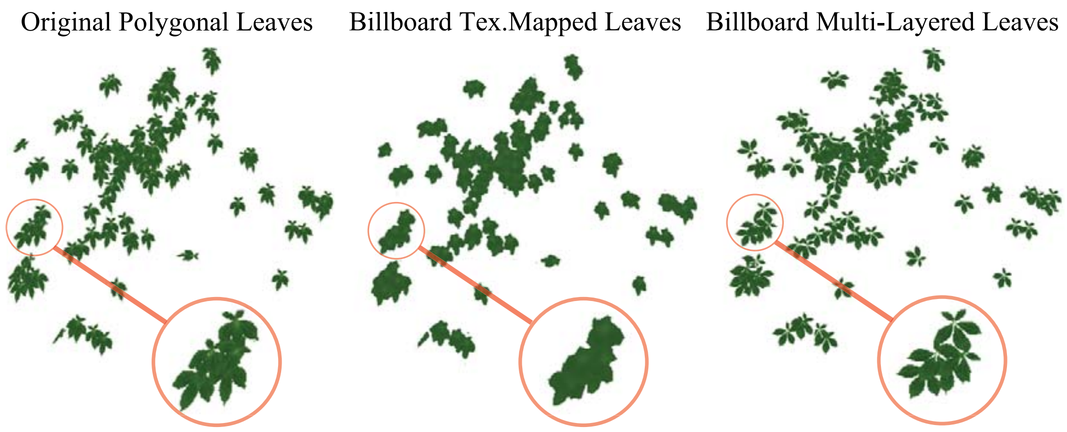

Original Polygonal Leaves
Billboard Tex.Mapped Leaves
Billboard Multi-Layered Leaves

Figure 6.8: Comparison between the group of polygonal leaves (left), with respect to the standard texture mapping result (middle) and the multi-layer method (right).

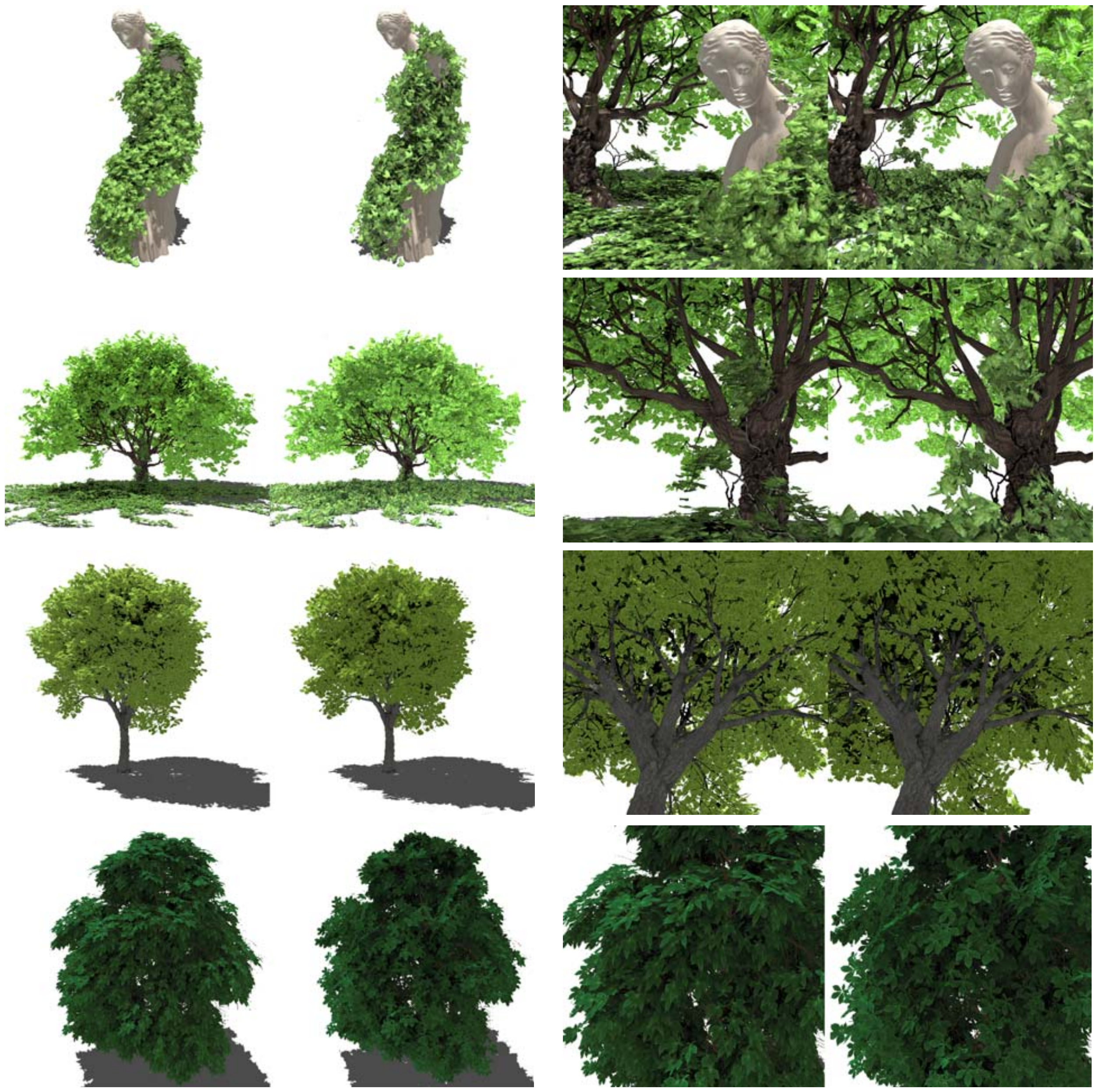


Figure 6.9: Visual comparisons between the polygonal plant/trees representation on the right and the multi-layered billboard clouds on the left. Including dynamic lighting and an precomputed ambient term. **First row:** Ivy plant composed by 8234 leaves simplified representation with 835 multi-layered billboards. **Second row:** Oak tree with 23660 leaves and a simplification with 194 multi-layered billboards. **Third row:** Cork tree with 20144 leaaves simplified to 121 multi-layered billboards. **Fourth row:** Chestnut tree with 11291 leaves simplified to 278 multi-layered billboards.

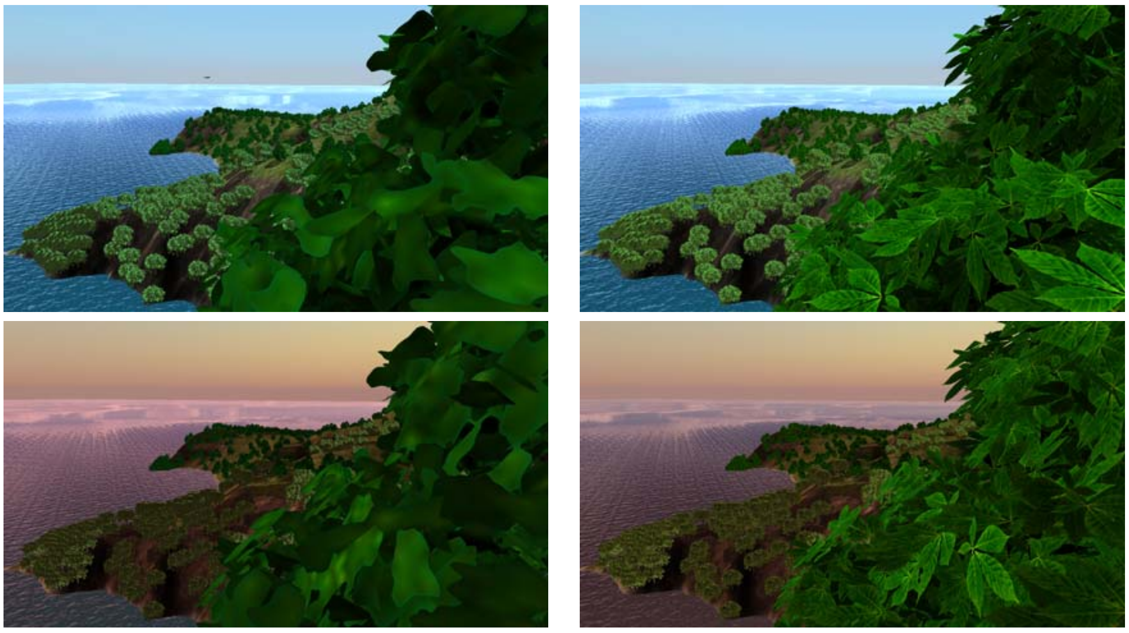


Figure 6.10: A complex sample scene under different lighting conditions. Left column shows the scene with the classic method [BCF+05, FUM05] using the same memory footprint as the images on the right, using the new method.

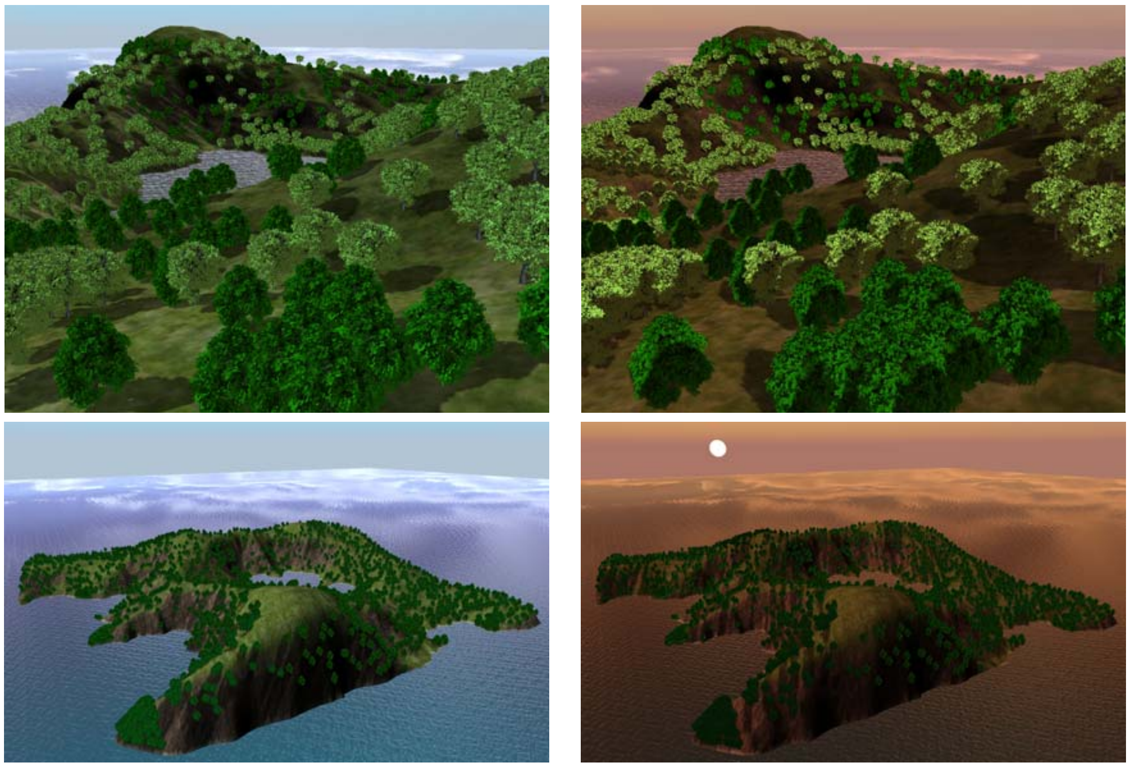


Figure 6.11: Two complex sample scene under different lighting conditions, using multi-layered billboard cloud for the leaves and progressive level of detail for the trunks.

Appendix A

Publications

Publications that support content of this master thesis:

- Ismael García and Mateu Sbert and László Szirmay-Kalos. Leaf Cluster Impostors for Tree Rendering with Parallax. In *Proceedings of Eurographics Short Presentations* 2005, pag. 69-72.
- Ismael García and Mateu Sbert and and László Szirmay-Kalos. Tree rendering with billboard clouds. In *Proceedings of Third Hungarian Conference on Computer Graphics and Geometry*, 2005, pag. 9-15.
- Ismael García, Gustavo Patow, Mateu Sbertand László Szirmay-Kalos. Multi-layered indirect texturing for tree rendering. In *Eurographics Workshop on Natural Phenomena*, 2007, Ed. D. Ebert, S. Mérillou.

Leaf Cluster Impostors for Tree Rendering with Parallax

Ismael García, Mateu Sbert, László Szirmay-Kalos

GGG
Girona Graphics Group

Abstract

● We present a simple method to render complex trees on high frame rates while maintaining parallax effects.

● Using clustering we find an impostor plane for each group of tree leaves that lie approximately in the same plane.

● Unlike billboards, these impostors are not rotated when the camera moves, thus the expected parallax effecs are provided.

● The replacement of large number of leaves by a single semi-transparent quadrilateral considerably improves rendering time.

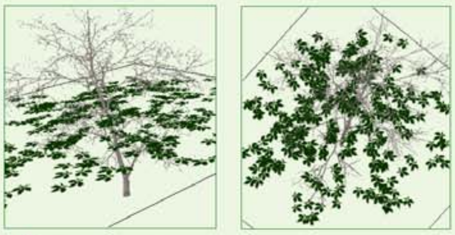
● Our impostors represent well the tree from any direction and provide accurate depth values.

Generating the impostors (I)

K-*means clustering algorithm*

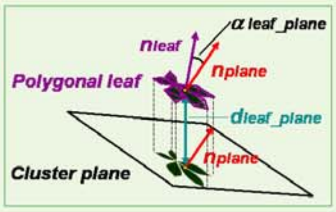
We form clusters of leaves in a way that all leaves belonging to a cluster lie approximately on the same on the same impostor plane and replace the whole group by a single impostor.

1. For each leaf we find that plane that minimizes the error. This step clusters the leaves into groups defined by the plane.
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.


▲ Two from 32 leaf clusters planes representing the tree.

Generating the impostors (II)

Error measure for a leaf and a plane



● The *error measure* evaluate the distance *d* between the center of the leaf and the cluster plane, and the angle α between the normals.

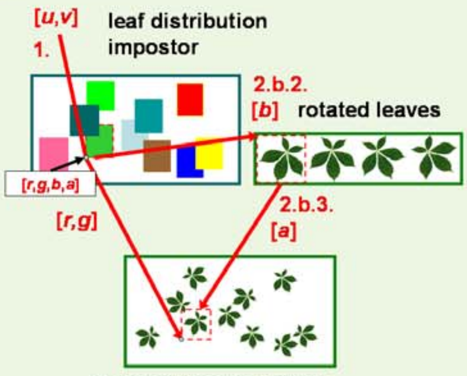
Finding a good impostor plane for a leaf cluster

● To find the optimal plane for a leaf group we minimize the total error using the Lagrange-multiplier.

Indirect texturing (rendering the impostors)

● The **leaf distribution impostor** stores the position in the *r*, *g* channels, the orientation of the leaves in channel *b* and the size of the leaf rectangle in the *a* channel.

● The **rotated leaves** at different angles are put in a separate texture.



1. During the rendering, we address this **leaf distribution impostor** with the texture coordinates *u*, *v*.

2.a. If the *r* value of the looked up texel is outside [0,1] then it's not addressing a leaf, and should be transparent.

2.b.1. Else the coordinates *r*, *g* are the texture coordinates of the lower left corner of the leaf rectangle.

2.b.2. The rotation angle selects the leaf texture that represets this rotation.

2.b.3. Finally the *u*, *v* coordinates are translated by *r*, *g* and scaled by size *a* of the leaf rectangle, using:

$$u' = (u - r) / a$$
$$v' = (v - g) / a$$

Results

● We present a tree rendering algorithm where clustering of leaves are represented by semi-transparent impostors.

● The automatic clustering algorithm fins an impostor in a way that the represented leaves lie approximately in its plane.

● The results are satisfactory in *vehicle scale*, *human scale* and *insect scale*.

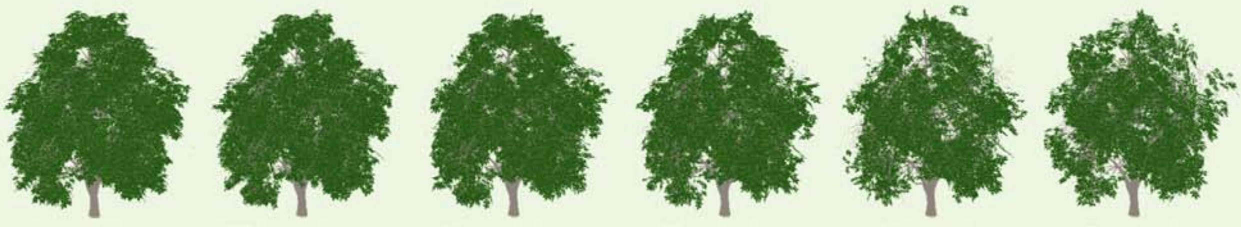
● The method is also good to generate shadows and can be extended to a hierarchical approach.

▲ Using level of detail we have 2000 trees rendered on 30 FPS without instancing.

▲ Applying indirect texturing we have the *insect scale* with nice results.

▲ This algorithm leads represent trees with only hundreds of polygons.

An european chestnut (Castanae Sativa) having 11291 leaves defined by 112910 faces. Visual comparisons using tree rendered with different number of impostors. ▶



Leaf Cluster Impostors for Tree Rendering with Parallax

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 simple method to render complex trees on high frame rates while maintaining parallax effects. Based on the recognition that a planar impostor is accurate if the represented polygon is in its plane, we find an impostor for each of those groups of tree leaves that lie approximately in the same plane. The groups are built automatically by a clustering algorithm. Unlike billboards, these impostors are not rotated when the camera moves, thus the expected parallax effects are provided. On the other hand, clustering allows the replacement of a large number of leaves by a single semi-transparent quadrilateral, which improves rendering time considerably. Our impostors well represent the tree from any direction and provide accurate depth values, thus the method is also good for shadow computation.

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 [DHL*98]. 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 [LRC*02].

Image-based methods [RMD04, ABC*04] represent a trade-off of consistency and physical precision in favor of

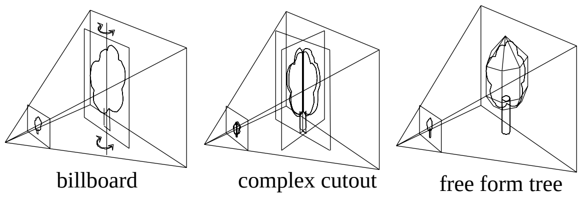


Figure 1: *A tree representation with images*

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. 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. In order to handle occlusions correctly, billboards can also be augmented with depth information [MO95, Szi05]. One of the most advanced methods actually implemented in commercial entertainment software is the basic *free-form textured tree model*, where the images are imposed on the approximated geometry.

The method of this paper falls into the class of complex cutouts, but it prepares them so carefully that the results are satisfactory not only on vehicle scale, but also on human scale, and with some compromises on insect scale. The method is also good to generate shadows [MNP01] and can be extended to a hierarchical approach [MDK99].

2. The new method

In order to reduce the geometric complexity of a tree, we use impostors having transparent textures to represent leaf groups. Since impostors are not rotated when the camera changes, the expected parallax effects are provided [DDSD03]. The key of the method is then the generation of these impostors. We should first observe that an impostor is a perfectly accurate representation of a polygon (i.e. leaf) if the plane of impostor is identical to the plane of the polygon. On the other hand, even if the leaf is not on the impostor plane but is not far from that and is roughly parallel to the plane, then the impostor 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 impostor plane and replace the whole group by a single impostor. 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* [Mac67] starts with a predefined number of random planes, and iterates the following steps:

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 $(\mathbf{n}_i, \mathbf{p}_i)$.

Leaf i approximately lies in impostor 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 impostor 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 $n_x, n_y, n_z, d, \lambda$, 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 $\mathbf{x}_0$ [Wei03]. Thus if we select $\mathbf{B} = \mathbf{A}^{-1}$, then the iteration will result in the eigenvector of the smallest eigenvalue.

2.3. 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. 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 2). 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.

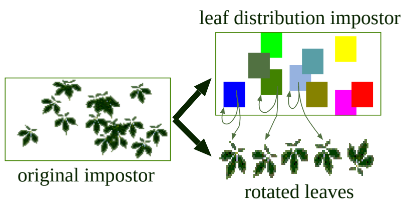


Figure 2: *Indirect texturing. The impostor is replaced by a leaf distribution impostor and the rotated images of the leaves.*

2.4. 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. 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 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 impostors using the proposed algorithm. When we did not apply indirect texturing, the impostor 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 leaf cluster impostors. The comparison of the images of the original polygonal tree and the impostor tree is shown in figure 4. Note that this level of similarity is maintained for all directions, and the impostor tree also provides realistic parallax effects.

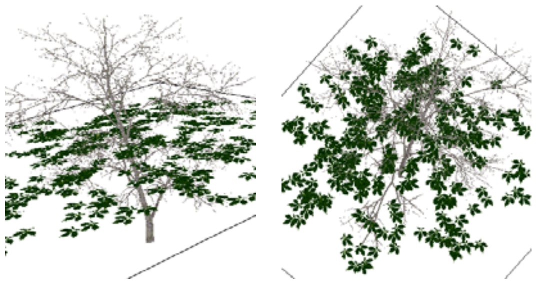


Figure 4: *Two from the 32 leaf clusters representing the tree*

Figure 5 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 from 32 to 16 and even to 8.

4. Conclusions

This paper presented a tree rendering algorithm where clusters of leaves are represented by semi-transparent impostors. 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 geometry information. Since

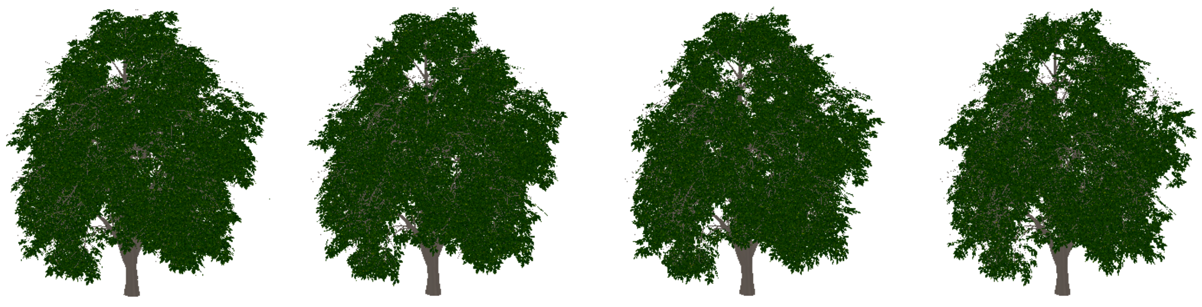


Figure 3: *Visual and speed comparisons of the original polygonal tree and the tree rendered with different number of impostors.*

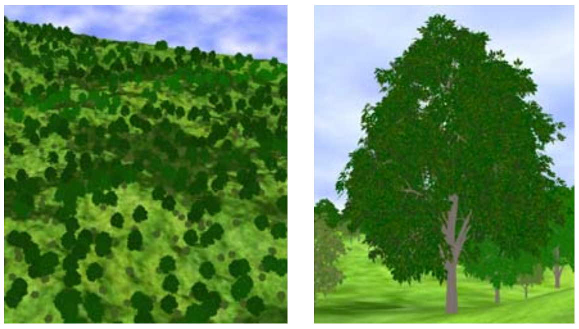


Figure 5: *Two snapshots from a 30 FPS animation showing 2000 trees*

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.

5. 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

[ABC*04] Andujar C., Brunet P., Chica A., Navazo I., Rossignac J., Vinacua J.: Computing maximal tiles and application to impostor-based simplification. *Computer Graphics Forum 23*, 3 (2004), 401–410. 1

[DDSD03] Décoret X., Durand F., Sillion F., Dorsey J.: Billboard clouds for extreme model simplification. In *SIGGRAPH '2003 Proceedings* (2003), pp. 689–696. 2

[DHL*98] Deussen O., Hanrahan P., Lintermann R., Mech R., Pharr M., Prusinkiewicz P.: Interactive modeling and rendering of plant ecosystems. In *SIGGRAPH '98 Proceedings* (1998), pp. 275–286. 1

[LRC*02] Luebke D., Reddy M., Cohen J., Varshney A., Watson B., Huebner R.: *Level of Detail for 3D Graphics*. Morgan-Kaufmann, 2002. 1

[Mac67] MacQueen J. B.: Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability* (1967), pp. 281–297. 2

[MDK99] Max N., Deussen O., Keating B.: Hierarchical image-based rendering using texture mapping. In *Eurographics Workshop on Rendering* (1999), pp. 57–62. 2

[MNP01] Meyer A., Neyeret F., Poulin: Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering* (2001). 2

[MO95] Max N., Ohsaki K.: Rendering trees from precomputed z-buffer views. In *Eurographics Workshop on Rendering* (1995), pp. 74–81. 2

[RMD04] Reche A., Martin I., Drettakis G.: Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics 23*, 3 (2004). 1

[Szi05] Szijártó G.: 2.5 dimensional impostors for realistic trees and forests. In *Game Programming Gems 5*, Pallister K., (Ed.). Charles River Media, 2005, pp. 527–538. 2

[Wei03] Weisstein E.: *World of Mathematics*. 2003. http://mathworld.wolfram.com/Eigenvector.html. 3

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].

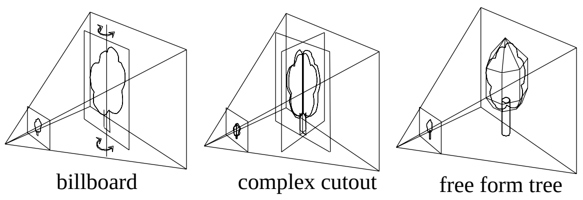


Figure 1: *A tree representation with images*

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:

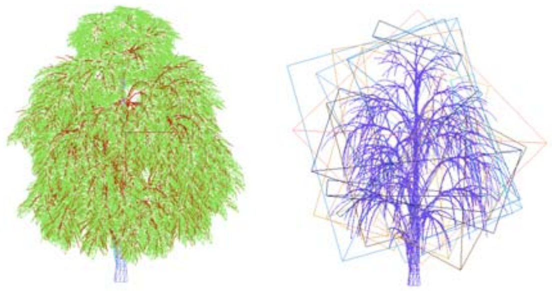


Figure 2: *The basic idea of the proposed method: the original tree defined by a hundred thousand polygons is approximated by a few fixed billboards, where a single billboard itself approximates many leaves.*

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 $(\mathbf{n}_i, \mathbf{p}_i)$.

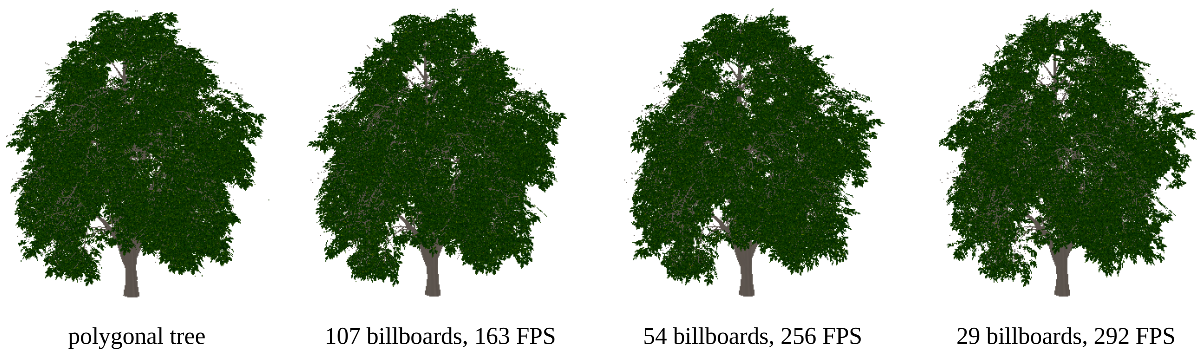


Figure 3: *Visual and speed comparisons of the original polygonal tree and the tree rendered with different number of billboards.*

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.

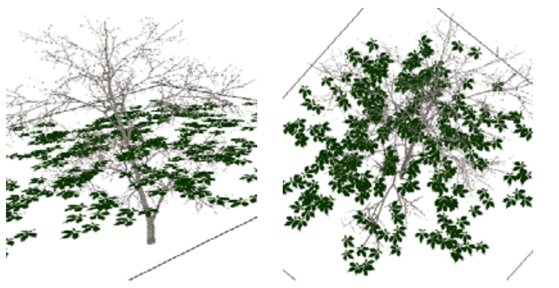


Figure 4: *Two from the 32 leaf clusters representing the tree*

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.

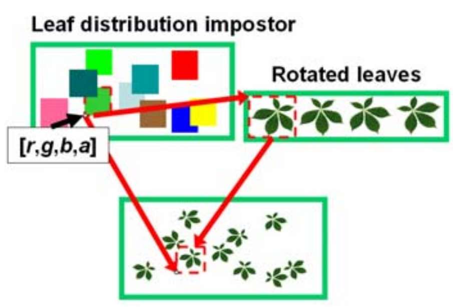


Figure 5: *Indirect texturing. The billboard is replaced by a leaf distribution impostor and the rotated images of the leaves.*

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.

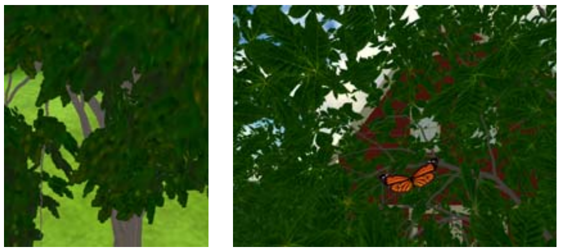


Figure 6: *Comparison of conventional filtered texturing and indirect texturing. Note that the high apparent texture resolution in the indirect texturing result.*

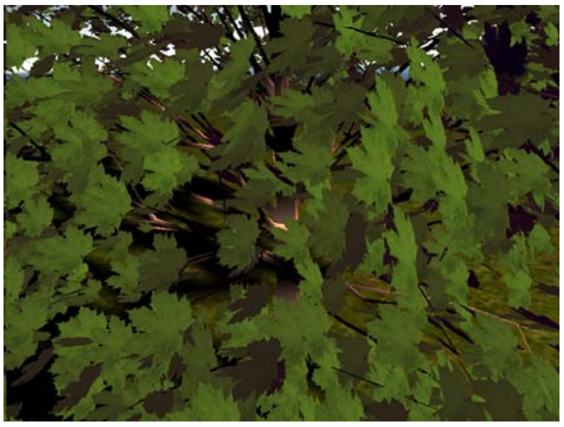


Figure 7: *Indirect texturing.*

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.

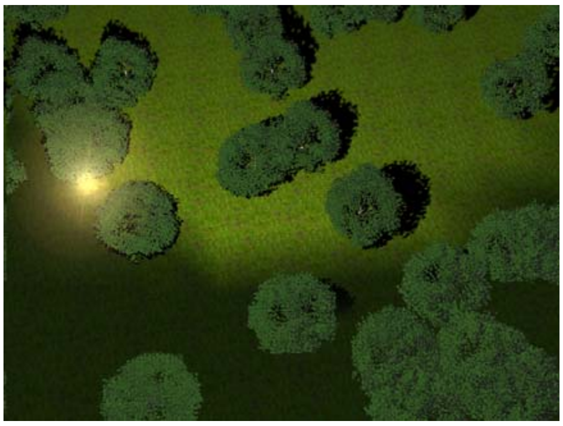


Figure 8: *Trees with illumination and shadow computation.*

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.

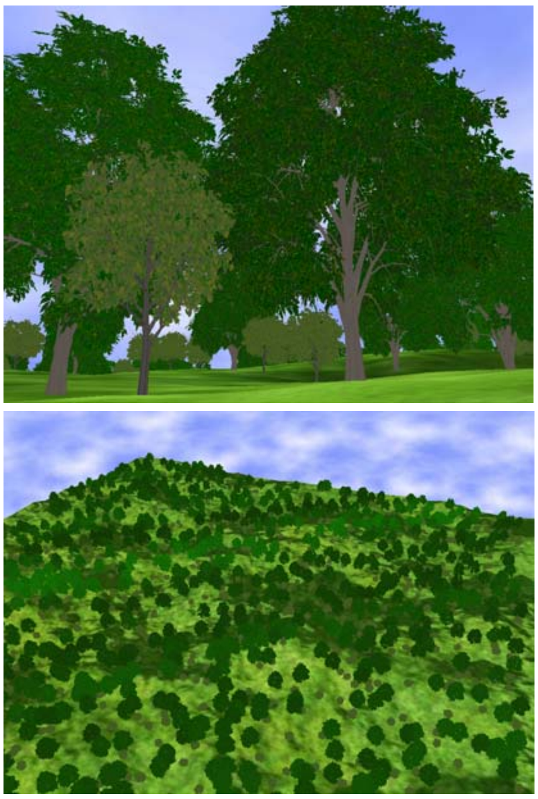


Figure 9: *Two snapshots from a 30 FPS animation showing 2000 trees without shadow and illumination computation*

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.

Eurographics Workshop on Natural Phenomena (2007)
D. Ebert, S. Mérillou (Editors)

Multi-layered indirect texturing for tree rendering

Ismael García[1], Gustavo Patow[1], László Szirmay-Kalos[2] & Mateu Sbert[1]

[1]University of Girona, Spain
[2]Budapest University of Technology, Hungary

Abstract

This paper presents a technique to render in real time complex trees using billboard clouds as an impostor simplification for the original polygonal tree, combined with a new texture-based representation for the foliage. The technique provides several new contributions with respect to previous approaches. The new algorithm allows progressive level of detail both at the geometric and at the shader levels. It also preserves the parallax effects of the original polygonal model keeping leaf positions, orientations, and preserving the overlapping of the leaves as seen from any view point. In addition, the texture-based representation provides high-definition close views without introducing high memory requeriments. We adapted a realistic lighting model with soft shadows and a global illumination precomputation, allowing to render highly complex scenes with thousands of trees in real time.

Categories and Subject Descriptors (according to ACM CCS): http://www.acm.org/class/1998/ I.3.3 [Computer Graphics]: Image generation, I.3.7 [Computer Graphics]: 3D Graphics and Realism

1. Introduction

In computer graphics, one of the current most challenging problems is the generation and interactive rendering of vast natural scenes, involving hundreds of thousands of trees and other vegetal species. This is specially true in forest scenes, whose complexity is impossible to model in detail without efficient level-of-detail (LoD) algorithms and data structures. The original polygonal foliage models provide the natural parallax effect, but cannot be used in real-time visualizations of natural scenes with thousands of trees. On the other hand, most of the previous image-based foliage representations fail either to keep the parallax effect, provide low resolution detail, or fall back to wrong foliage alignment.

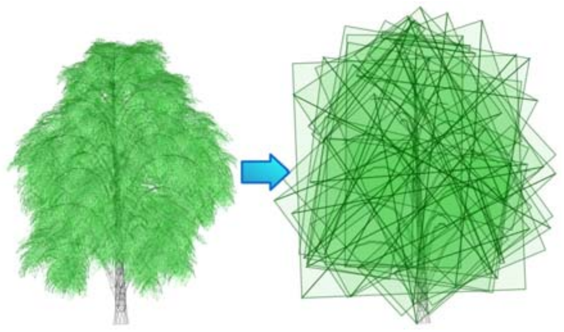
We propose a new texture foliage-based representation that provides high-definition from close views and progressive level-of-detail transitions with respect to the observer distance, keeping the appearance of the original complex geometry model and allowing realtime rendering of scenes with thousands of trees. In this sense, it outperforms existing techniques in the quality of close-views, with a much lower memory footprint. It also allows a progressive LoD technique that gracefully converges to the standard billboard cloud method for far-views, thus optimizing rendering efficiency. In addition, a lighting model including shadows, and ambient occlusion is incorporated to allow realistic, real-time visualization of large foliage sets.

The paper is organized as follows. In Section 2 we discuss previous work on interactive tree rendering. In Section 3 we present our foliage rendering approach. We describe the billboard cloud generation framework in Section 4 and we present in detail the visualization algorithm in Section 5. We give some implementation details in 6, demonstrate the results obtained using our techniques in Section 7 and discuss future work directions in Section 8.

2. Previous work

The problem of rendering complex natural scenes has already received a lot of attention. There are two general approaches for interactive realistic rendering for trees: geometry-based (see [HG97] and [RCB*02]) and image-based techniques. Our work is included in the second group.

Image-based methods represent a trade-off between consistency and physical precision in favor of more photorealistic visuals. Billboard rendering is analogous to using cardboard cutouts: the billboard plane is always turned to-


(a) The polygonal leaves are replaced by a billboard cloud.

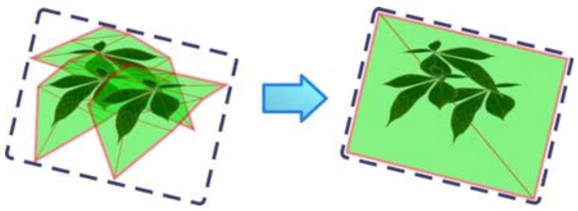

(b) Each group of leaves is replaced by a billboard.

Figure 1: *Replacing large groups of leaves by semi-transparent quadrilaterals with a texture map.*

wards the camera. Although this simple trick solves the parallax problem, the tree always looks the same no matter from where we look at it. Shade et al. [SGHS98] and Chang et al. [CBL99] used layered depth images (LDI) to render complex natural objects from pre-computed pixel-based representations with depth, with different levels of detail, but resulted in a low image quality for closeups. Max et al. [Max96, MDK99] modelled and rendered trees hierarchically from pre-computed images with Z-buffers, but the rendering times were not suited for interactive applications due to extensive texture transfer operations. Meyer and Neyret [MN98] converted complex natural objects into mip-mapped volumetric textures, which were then raytraced. On the other hand, in [RMMD04, LRMDM06] they estimated opacity in a volume, and then generated and displayed view-dependent textures attached to cells of that volume.

One of the most advanced methods actually implemented in commercial entertainment software [SPE05] is the basic free-form textured tree model, where the images are imposed on the approximated geometry. To represent tree models using billboards, the "billboard clouds" approach can be used [DDSD03], which represents a geometry by a set of arbitrarily oriented billboards. In our work we also use billboard clouds, but we generate the billboard set based on the work presented in [BCF*05] and [GSSK05], better suited for trees. We use a clustering algorithm on the basis of the leaf faces and vertices that enables us to find better billboard approximations. Then, leaves textures are used to add visual detail. In [GSSK05] it is also proposed an indirect texture approach, but it can only use one leaf image per cell, which leads to a regular-looking pattern that fails to preserve the original leaves positions and orientations and cannot preserve the original overlapping, resulting in sparser-looking trees, as seen in Figure 2. In [FUM05] they generated trees using super-sampling in an off-screen buffer, and then they applied downsampling with respect to a user-defined texture-size, trying to preserve the small details of the leaves. A drawback of this method is not being effective enough for close views.

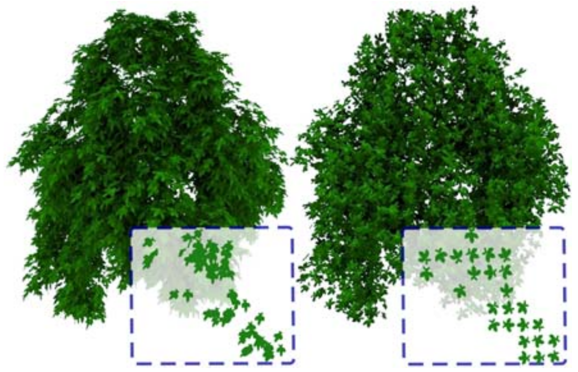


Figure 2: *Comparison and close up of the polygonal tree and billboards with simple indirect texturing [GSSK05].*

The techniques presented in this paper are related to the one presented in [Gla05], that placed small images at irregular intervals to help reducing the pattern artifacts of the free-form textured models, but did not handle the overlapping of the small images in a general case.

3. Overview

The presented method is based on replacing large groups of leaves by semi-transparent quadrilaterals defined as bill-

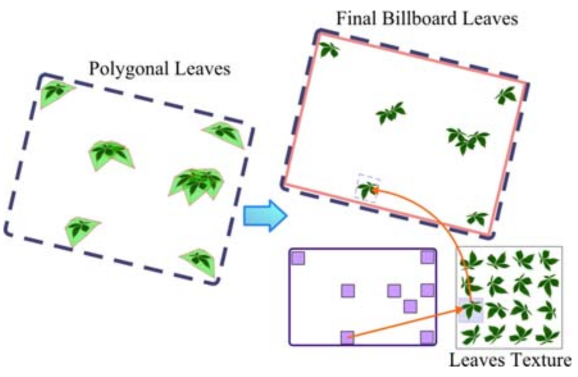


Figure 3: *Indirect texturing uses values in the texels of the indirect texture to fetch the corresponding leaf color image.*

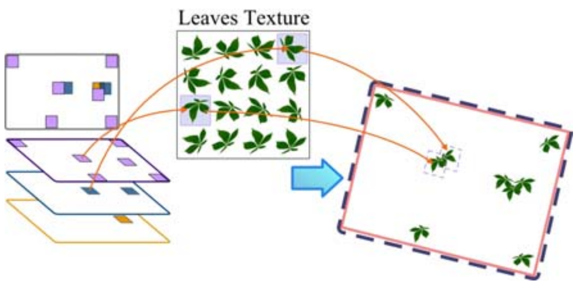


Figure 4: *Overlapped leaves must be encoded with a layered indirect texture to be properly rendered.*

boards (see Figure 1). Each billboard has a texture generated with an orthogonal projection of the polygonal faces of the leaves. In order to keep the leaves details, we use a procedural technique that replaces polygonal leaves by leaf patterns, called "indirect texturing" [OL98], which allows per-pixel computed values from one texture to be used as texture coordinates for an additional texture fetch. The leaves positions are used to place each leaf on the billboard surface, and an orientation index is used to fetch a leaf image from a color texture (called the Leaves Texture), as shown in Figure 3.

To properly handle the overlapping leaves within the same billboard, multiple texture maps would be needed, and the indirect texture should be encoded as a layered one, as shown in Figure 4. To improve storage and look-up efficiency, one indirect texture can be used to address lists of overlapping leaves, encoded in a second texture. Each entry in the lists contains a leaf position and an orientation index to address a leaf color image, as shown in Figure 5.

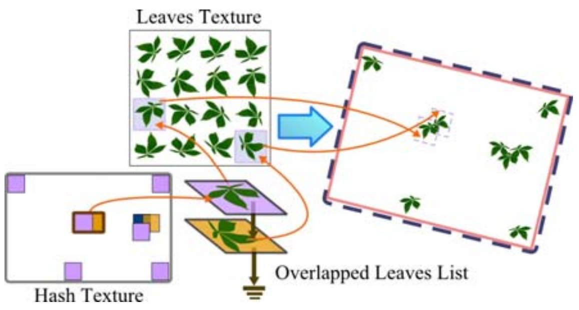


Figure 5: *Indirect texturing with overlapped leaf fragments lists reduce texture requeriments.*

The proposed method takes information from each group of leaves, generated by the billboard simplification algorithm and prepares a set of textures that are effective at different scales of simulation, from close views of individual branches and leaves, to bird's-eye flights, providing a viewing quality very close to the original complex polygonal model, as shown in Section 7.

4. Billboard cloud algorithm

As observed in Section 2, there are many automatic geometry simplification methods, but applying them to trees only provides acceptable results with the polygonal meshes that represent the trunk and the branches. Those methods do not work properly for the foliage. On the other hand, the billboard clouds approaches such as [BCF*05], can find nearly optimal sets of billboards for effective simplification, replacing sets of polygonal leaves by simple quadrilaterals.

4.1. Clustering

An algorithm like the one presented in [BCF*05] and [GSSK05] is used to group the leaves into clusters, where each cluster is represented by one billboard. One interesting property of the billboard cloud is that it completely bounds the shape of the original model, as shown in Figure 1.

The number of clusters can be specified by the user to control the number of billboards generated by the algorithm. The number of needed billboards depends on the size and complexity of the plant to be modelled. In general, the usage from 60 to 260 billboards for trees, or even fewer for smaller plants as bushes, is enough for visualization purposes.

4.2. Texture generation

Depending on the tree specie and age, a mature tree can have between 30,000 and 200,000 leaves. This intrinsic complexity leads to hundreds of leaves per billboard when a billboard cloud of about 260 quadrilaterals is used. As an example, one typical billboard plane, with 116 leaves, may have up to 10 overlapping leaves.

Our leaves are positioned with a continuous representation: each leaf has an associated vector that represents its position in local billboard coordinates (see Figure 8). In order to achieve an accurate overlapping, multiple layers must be used, so we subdivide the original billboard into a regular grid of cells, getting, for each grid cell, a set of cell-sized layers that may contain nothing (transparent) or a leaf fragment. Then, we link those cell-sized layers into a list of overlapping leaf fragments, as shown in Figure 6. We can consider that this grid of cells acts as a sort of spatial hash that accelerates fetches into this multilayered texture, so it is called the Hash Texture. The lists of leaf fragments are consecutively stored in a second, separate texture, which we call the Lists Texture (see Figure 8). Each list entry is a record that contains a reference to the third texture (the Leaves Texture), the leaf position vector, a scale factor, and any additional information that needs to be stored at the leaf level (see Section 5.2).

One of the key points of the proposed method to preserve accurate foliage placement, is the generation of the Hash and Lists Textures. After the clustering process, we send each leaf as a point sprite to an off-screen buffer, encoding the leaf position and orientation and storing this information in the Lists Texture. To create the cell-sized layers, we use a depth-peeling technique [Eve99] that sorts the *leaf fragments* in front-to-back order, as shown in Figure 6.

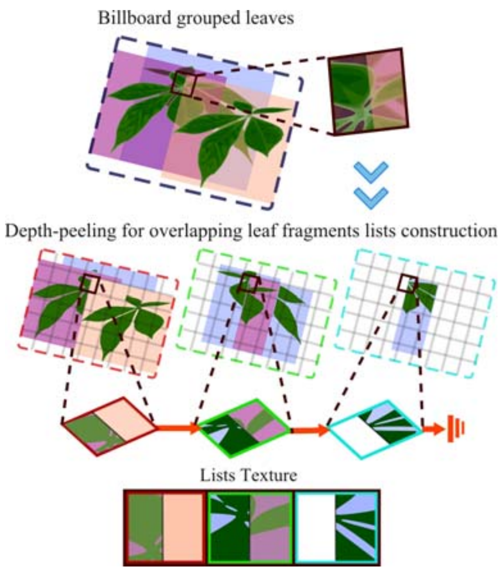

Billboard grouped leaves
Depth-peeling for overlapping leaf fragments lists construction
Lists Texture

Figure 6: *Each leaf is processed as a point sprite. The point sprite can appear in more than one grid cell of the Hash Texture. The point sprites are processed with depth peeling to build an overlapping leaf fragments list for each grid cell.*

The size of the point sprite that holds the leaf information is defined with respect to its container billboard. This size is defined as the ratio between the 2D bounding box of the projection of the original leaf, and the size of the billboard (computed as $billboard_{pixels} = HashTex_{width} * HashTex_{height}$). In order to get the exact size of the point sprite in *pixels*, this size is multiplied by the total number of texels in the Hash Texture, following the expression:

$$leaf_{pixels} = \frac{area(leaf_{quad}) \cdot billboard_{pixels}}{area(billboard_{quad})}$$

leading to $leaf_{width} = leaf_{height} = \sqrt{leaf_{pixels}}$

The last preprocessing step builds the Hash Texture, where each texel addresses one entry in the Lists Texture, or transparent when the cell does not hold any leaf fragments. It is important to mention that the billboard polygons not only have *uv* texture coordinates associated with them, but also encode the Hash Texture resolution, for later use.

5. Visualization algorithm

When the graphics hardware draws a billboard, a single pass fragment shader is called with the corresponding *uv* texture coordinates. These coordinates are used to fetch the respective cell in the Hash Texture. As mentioned earlier, this cell can be either transparent, so the shader also will return a transparent fragment, or it can contain a reference to a list in the Lists Texture. Then, in an iterative process, each entry in the list is fetched until an opaque leaf fragment is found, or there are no more leaf fragments to evaluate. In the latter case, a transparent color is sent as output. This process is illustrated in Figure 7.

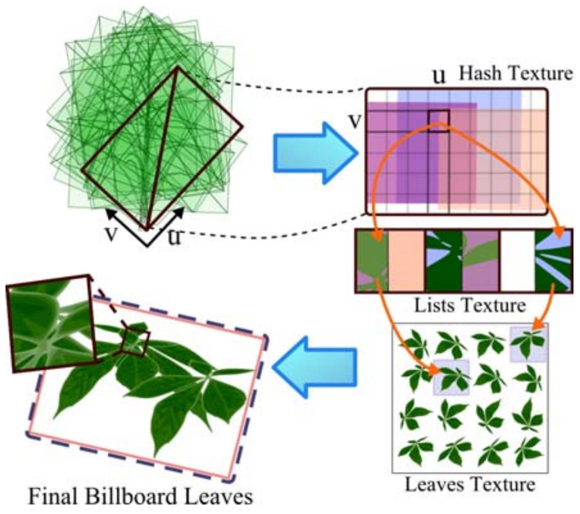


Figure 7: *Multi-layer indirect texturing evaluation in the single pass fragment shader.*

When evaluating each entry in the list, the reference to a leaf image into the Leaves Texture and the 2D vector giving the exact position of the leaf are retrieved. The position is subtracted from the *uv* coordinates the shader received to obtain a vector we call the Local Position vector. On the other hand, the leaf reference into the Leaves Texture is used as a vector pointing to the upper left corner of the corresponding image of the leaf. This vector is added to the Local Position vector to retrieve a texel from the Leaves Texture which either contains an opaque pixel with a color from the leaf, or transparent, indicating we indexed outside the leaf and we should continue with the next entry, as shown in Figure 8. If we add size information in the leaf entries, the Local Position vector is scaled accordingly.

Another important factor to take into account is that leaves overlap with a different order depending on the viewing direction. To incorporate this feature, the Hash Texture also stores the size of the list it is indexing, so we can access the list by both ends, just by looking at the direction of the billboard normal with respect to the viewer's direction. This also offers the advantage of allowing to index *two* Leaves Textures, one with the front sides of the leaves and the other with the back sides. These two textures can be unified in a single texture, and thus, a single indexing framework, to avoid further conditional branches.

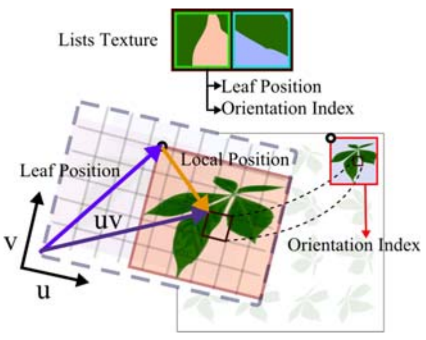


Figure 8: *Information stored at an entry in the list of leaves for a hash cell.*

5.1. Level-of-detail with indirect texturing layers

Although this method was designed for close or medium range views, it works well if mip-mapping is used for the Leaves Texture, as it would produce the right effect at large distances. However, the shader computations described above would be executed no matter the viewing distance, resulting in an unnecessary overhead that impacts on the fill-rate without representing a significative improvement in the visualization quality. To alleviate this problem, we introduce a progressive Level of Detail technique at the shader level.

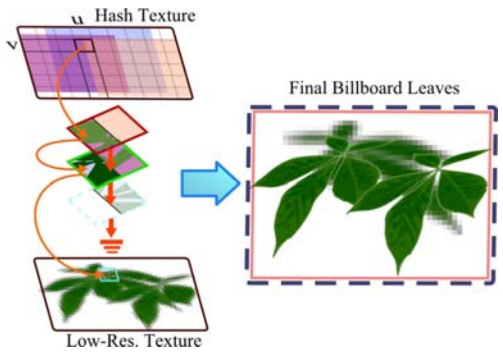


Figure 9: *As distance increases, the shader-LoD technique evaluates less list entries, and the missing ones are substituted by a low-resolution texture.*

Basically, this technique consists of reducing the number of evaluated list entries when the distance from the observer increases (see Figure 9). But evaluating less list entries means evaluating less leaf fragments, which will generate holes in the tree that get bigger as the viewing distance increases. This is solved by introducing a regular low-resolution texture that represents the missing leaves, generated in a pre-processing pass from the layered texture. So, as distance increases, less elements are evaluated and the missing ones are replaced by a single texture fetch into the low-resolution texture. Finally, when the viewing distance is high enough, no list entries are used any more and only the low-resolution texture is evaluated. Our experiments show that the fact that the leaves that are evaluated from a list entry are also present at the low-resolution texture does not produce any noticeable artifact. A comparison can be found in Figure 10. Finally, when the tree is at a very large distance, the technique would show only the low-resolution texture, without fetching any list. So, more evaluations can be avoided by a switch in the shader context to a shader that only evaluates the low-resolution texture without further computations.

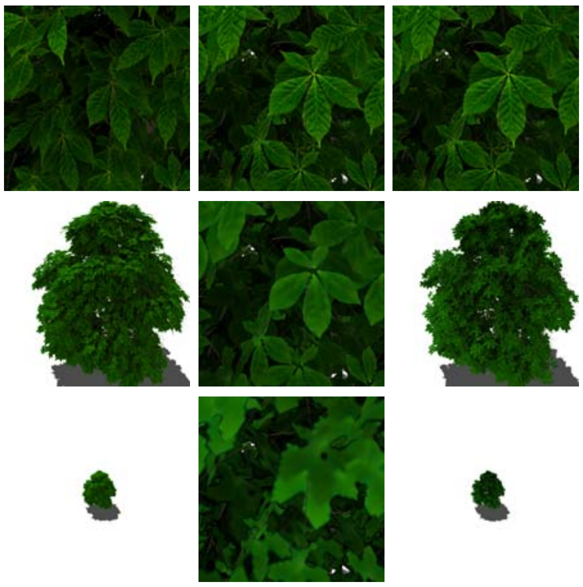


Figure 10: *Comparison between the polygonal tree on the left and the multi-layer technique on the right, preserving accurate alignment of the leaves. The middle column shows the progressive shader LoD. The small holes are due to slight changes in the leaves alignment.*

Fortunately, modern GPUs incorporate primitives (ddx, ddy) that help computing the number of needed evaluations. We have used the formula described by [LB06] (using the Hash Texture size stored in the billboard, as described in Section 4.2). Once we know the relationship between the Hash texel and its projection onto the screen, a number of needed entries to evaluate from the list is computed. This number is a factor times the length of the diagonal of the projected Hash texel. This expression is also used to choose the mip-map level of the Leaves Texture.

5.2. Photorealistic lighting

The shading model used for the trees can be split into two techniques. On the one hand, trunk and branches simplified models are rendered with bump mapping, ambient occlusion and shadow mapping. On the other, the foliage lighting is obtained as the combination of the proposed multi-layer technique with ambient occlusion and shadow mapping.

Billboard clouds approximately preserve the original geometry, thus they can also replace the original tree when shadows or global illumination effects are computed. We also store a normal map together with the RGB map in the leaf texture, to allow for accurate real-time relighting. If needed, a correction for the billboard plane normal could also be stored in the entries of the Lists Texture, and used in conjunction with the local normal retrieved from the normal map to obtain a final normal to compute the shading.

To compute shadows, i.e., the visibility from point and directional lights (like the sun), we use Light Space Perspective Shadow Maps [WSP04]. To generate the shadow map, we adapted its usage in combination with the multi-layer method: for close-up views, where shadows need to be highly detailed, we use a full evaluation of the layers as described before. But, as soon as the observer gets farther enough, as described above, we switch to the low-resolution evaluation shader which provides good results at a fraction of the evaluation cost.

In order to complete the lighting model, we incorporate a pre-computed ambient occlusion [GFS07] term to take into account global illumination effects, computed for the original polygonal tree. We encode the ambient occlusion term as one new field in each entry in the Lists Texture. Because of the limitations of current graphics hardware for packing into single precision RGBA channels, we decided in our implementation to store the ambient term in a separate texture, indexed the same way as the Lists Texture.

To be compatible with our high-resolution representation of leaves into the Leaves Texture, we extrapolate the ambient term computed for the polygonal leaves to the four vertices of the 2D leaf bounding box. This is computed by setting a simple over-determined system of linear equations that relate the ambient term values at each projection of a polygonal vertex onto the billboard plane, with the corners that define the 2D bounding box of the projected leaf. See Figure 11. This system can be solved with any numerical method, or it can be approximated by taking only the four vertices of the original leaf that include the maximum value, the minimum value, and two extra values of the ambient term, and by solving only a system of 4 equations with 4 unknowns.

In the visualization, this ambient term is linearly interpolated in the fragment shader to provide the final ambient occlusion factor for the fragment on screen. One optimization would be, when the ambient term is stored in a separate texture, to use two consecutive rows to store the information for a single list, so ambient values would be interpolated by the GPU when queried. For this to provide good results, the Local Position vector mentioned in Section 5, should be used in combination with the texture coordinates used retrieved from the Hash Texture. For the low-resolution texture, we use an ambient occlusion term encoded in the alpha channel of the low-resolution leaves texture.

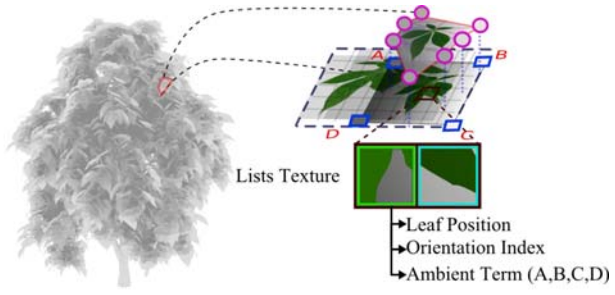


Figure 11: *Ambient term generation for the multi-layer leaves representation is encoded as a new field of the List Texture at the leaf level. The small squares represent the actual ambient occlusion values at the corresponding vertices.*

6. Implementation Details

The method presented so far requires one Hash Texture for each billboard, and in normal cases, there could be 256 and more billboards, largely exceeding current hardware capability of indexing textures at once. To handle this, several context switches would be needed, severely impinging on the overall performance. To solve this problem, all Hash Textures are condensed in a single large Hash Texture Atlas. In the same way, the different lists of the Lists Textures for each billboard are stored together in a single Lists Texture Atlas. In our experiments, the largest textures needed to store the Hash Textures and Lists Textures is about 1024^2 texels.

In addition to the shader level-of-detail technique described in Section 5.1, we used different billboard clouds for the foliage with a decreasing number of billboards, and switched among them as the distance to the observer increased. We stored three different billboard clouds that were stored in a single Vertex Buffer Object and indexed as necessary. The simplest billboard cloud is used in conjunction only with the low-resolution texture as described above, and the changes in billboard levels are ensured to be consistent with the corresponding shader LoD at the time of the switch. With respect to the tree trunk and branches, we used a progressive geometry-based Level-of-Detail technique, as described in [Hop96].

7. Results

The proposed technique has been implemented in OpenGL/Cg integrated into the Ogre3D engine, and run on a NV8800GTS graphics card. Figures for the usage of different tree species can be found in Table 1. The whole tree pre-processing stage takes between 2 to 5 minutes for all the examples presented here. All scenes were rendered at

Tree Species	Leaves	Polys/ leave	Billboards[3]	Memory[3]
Chestnut[1]	11291	10	278	3486762
Oak[2]	23660	2	194	3119734
Cork[1]	20144	4	121	1462870
Horse Chestnut[1]	9366	18	202	3376653
Shrub[2]	2388	2	198	1349091
Maple[1]	3662	2	175	1381600
Alder[1]	1049	2	228	1332837

Table 1: *Figures for different species showing the number of leaves, polygons per leaf, billboards and memory usage (in bytes) with our method. Labels mean:* [1]*Generated by Xfrog [Xfr06],* [2]*Generated by Forester,* [3]*Using texture compression DTX1-5, which provides a 1:6 compression ratio.*

Trees	Polygonal	Billboard
1	180	80
50	47	46
100	15	39
1000	5	28
10000	n/a	20

Table 2: *FPS for varying numbers of trees, for the original tree and the new method. Distances range from very close views to very far trees (covering only about* 32^2 *pixels).*

1280x1024 pixels, resulting in the frame rates of the system for varying number of trees presented in Table 2.

In Figure 12 we can see the results when compared to the original polygonal tree. As we can see in the figure, with only 278 billboard planes with a Hash Texture of 1024×1024 cells and up to 10 layers, the final results look very similar to the original. Figure 13 show complex scenes with tens of thousands of trees under very different lighting conditions and viewing distances.

8. Conclusions and future work

We have presented a method that allows interactive visualization of large forests, with tens of thousands of trees at interactive frame rates, even with realistic lighting effects like shadows and an ambient occlusion term. The new method consists of an indirect texturing technique, in combination with a texture used as a hash for fast indexing and retrieval, and a layered representation of the overlapping leaf fragments. Previous methods require extremely large texture memory space to preserve small details in the leaves, while the presented method preserves them with a low memory footprint and a few extra texture accesses. As an example, a chestnut would require about 246MB of texture space to achieve the same visual quality as our method, which only needs 3.7MB. For comparison, the original polygonal model for the leaves needs 23Mb.

Level-of-detail techniques have specifically been developed, not only at the geometric level (reducing the number of billboards and simplifying the geometry of the trunk and branches), but also at the shader level, drastically reducing the computational costs associated with rendering trees far from the observer. It is important to note that detail in the leaves is preserved even at very short distances, something that was not done before with billboard clouds. These results hold even for trees without dense foliage, making our technique highly suitable for instancing.

It must be mentioned that this algorithm is resolution dependant: depending on the screen-size of the trees and the screen resolution, different frame rates would be obtained. Nevertheless, as mentioned in Section 7, we are able to present about ten thousand trees in a complex scene in resolutions of 1280×1024 with an acceptable frame rate.

One line of promising immediate research is to change the progressive level-of-detail technique described in section 5.1 to a continuous LoD technique. This technique would perform an interpolation between shader levels, resulting in a smoother switch between LoD levels. Another interesting line is to extend the presented technique to include small branches, which would reduce the complexity of the geometry of the trunk, which then would be more easily handled with more traditional techniques. Finally, the incorporation of branch movements seems an interesting challenge to be modelled with the presented algorithm, as the Hash Texture would be slow to regenerate.

9. Acknowledgments

This work has been supported by GameTools FP6 (IST-2-004363) project, TIN2004-07451-C03-01 and TIN2004-07672-C03-01 projects from the Spanish Government, and by the Spanish-Hungarian Fund (E-26/04).

References

[BCF*05] BEHRENDT S., COLDITZ C., FRANZKE O., KOPF J., DEUSSEN O.: Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum 24*, 3 (2005), 507–516.

[CBL99] CHANG C.-F., BISHOP G., LASTRA A.: LDI tree: A hierarchical representation for image-based rendering. *ACM Computer Graphics*, Annual Conference Series (1999), 291–298.

[DDSD03] DÉCORET X., DURAND F., SILLION F. X., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Trans. Graph. 22*, 3 (2003), 689–696.

[Eve99] EVERITT C.: Interactive order-independent transparency, 1999. White paper, NVIDIA Corporation.

[FUM05] FUHRMANN A. L., UMLAUF E., MANTLER S.: Extreme model simplification for forest rendering. In *Eurographics Workshop on Natural Phenomena* (2005).

[GFS07] GONZÁLEZ F., FEIXAS M., SBERT M.: An information-theoretic ambient occlusion. In *International Symposium on Computational Aesthetics* (2007).

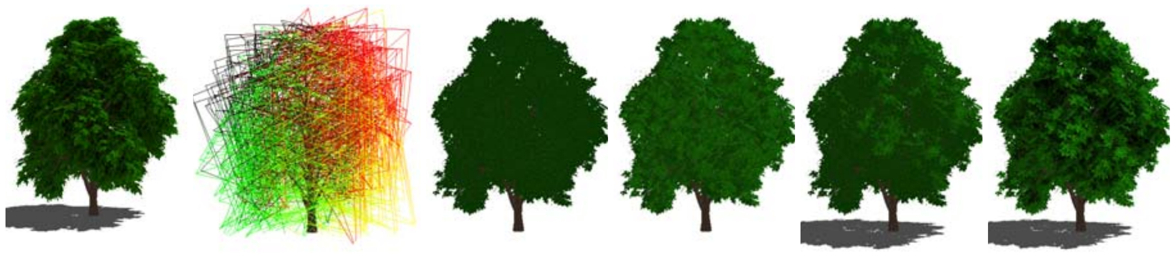


Figure 12: *Comparison between the polygonal tree and the multi-layer technique. From left to right: Original polygonal tree fully lit, the wire frame billboards, billboards with diffuse lighting, direct lighting, shadows and, finally, the full illumination model including the ambient occlusion term.*

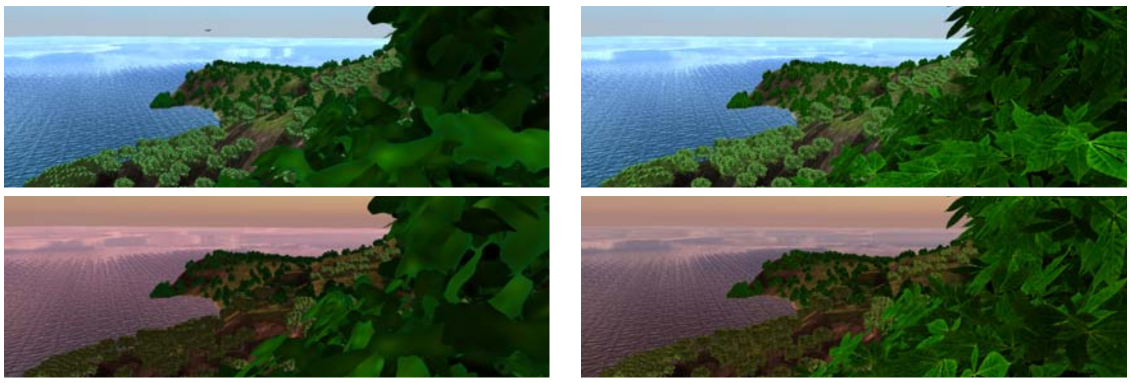


Figure 13: *A complex sample scene under different lighting conditions. Left column shows the scene with the classic method [BCF*05, FUM05] using the same memory footprint as the images on the right, using the new method.*

[Gla05] GLANVILLE R. S.: Texture bombing. In *GPU Gems 1*, Fernando R., (Ed.). Addison-Wesley, Oct. 2005, pp. 323–338.

[GSSK05] GARCÍA I., SBERT M., SZIRMAY-KALOS L.: Leaf cluster impostors for tree rendering with parallax. In *In Proceedings of EG Short Presentations 2005* (2005), pp. 69–72.

[HG97] HECKBERT P. S., GARLAND M.: Survey of polygonal surface simplification algorithms. In *ACM SIGGRAPH 1997 Course Notes* (1997).

[Hop96] HOPPE H.: Progressive meshes. *ACM Computer Graphics 30*, Annual Conference Series (1996), 99–108.

[LB06] LOVISCACH J., BREMEN H.: *Game Programming Gems 6*. Charles River Media, Inc., 2006, ch. Rendering Road Signs Sharply.

[LRMDM06] LINZ C., RECHE-MARTINEZ A., DRETTAKIS G., MAGNOR M.: Effective multi-resolution rendering and texture compression for captured volumetric trees. In *Game-On 2006* (2006), pp. 16–21.

[Max96] MAX N.: Hierarchical rendering of trees from precomputed multi-layer z-buffers. In *Eurographics workshop on Rendering techniques '96* (1996), pp. 165–174.

[MDK99] MAX N., DEUSSEN O., KEATING B.: Hierarchical image-based rendering using texture mapping hardware. In *Rendering Techniques (Eurographics Symposium on Rendering)* (1999), pp. 57–62.

[MN98] MEYER A., NEYRET F.: Interactive volumetric textures. In *Rendering Techniques (Eurographics Workshop on Rendering)* (1998), Drettakis G., Max N., (Eds.), Springer Wein, pp. 157–168.

[OL98] OLANO M., LASTRA A.: A shading language on graphics hardware: the pixelflow shading system. *ACM Computer Graphics 32*, Annual Conference Series (1998), 159–168.

[RCB*02] REMOLAR I., CHOVER M., BELMONTE O., RIBELLES J., REBOLLO C.: Geometric simplification of foliage. In *Eurographics'02 Short Presentations* (2002), pp. 397–404.

[RMMD04] RECHE-MARTINEZ A., MARTIN I., DRETTAKIS G.: Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Trans. Graph. 23*, 3 (2004), 720–727.

[SGHS98] SHADE J. W., GORTLER S. J., HE L.-W., SZELISKI R.: Layered depth images. *ACM Computer Graphics 32*, Annual Conference Series (1998), 231–242.

[SPE05] Speedtree, interactive data visualization inc., 2005. http://www.idvinc.com/speedtree.

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering 2004)* (2004), pp. 143–151.

[Xfr06] Greenworks: Organic software, 2006. http://www.greenworks.de/.