Interactive Vegetation Rendering with Slicing and Blending

Aleks Jakulin

Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia

Abstract

Detailed and interactive 3D rendering of vegetation is one of the challenges of traditional polygon-oriented computer graphics, due to large geometric complexity even of simple plants. In this paper we introduce a simplified image-based rendering approach based solely on alpha-blended textured polygons. The simplification is based on the limitations of human perception of complex geometry. Our approach renders dozens of detailed trees in real-time with off-the-shelf hardware, while providing significantly improved image quality over existing real-time techniques. The method is based on using ordinary mesh-based rendering for the solid parts of a tree, its trunk and limbs. The sparse parts of a tree, its twigs and leaves, are instead represented with a set of slices, an image-based representation. A slice *is a planar layer, represented with an ordinary alpha or color-keyed texture; a set of parallel slices is* a slicing. *Rendering from an arbitrary viewpoint in a 360 degree circle around the center of a tree is achieved by blending between the nearest two slicings. In our implementation, only 6 slicings with 5 slices each are sufficient to visualize a tree for a moving or stationary observer with the perceptually similar quality as the original model.*

1. Introduction

It is important to draw a distinction between vegetation modeling and rendering. Vegetation modeling is successfully done with various flavors of L-systems [6]. There is abundant literature in this field, for example [11, 1], along with several commercial modeling tools. However, the generated polygonal models have extremely high complexity, making them unsuitable for real-time rendering. The current line of research seems to lie in improving and animating the models, while real-time rendering of vegetation has not yet been explored extensively.

Due to variety of vegetation, it is necessary to focus on particular types. The three possible benchmarks are grass, small plants and trees. Grass resembles hair and fur, which are already an active field of research, with some interactive approaches, such as [4, 5]. Most small plants are not geometrically complex, and their leaves are relatively large in comparison with the size of the whole plant, which opens few possibilities for shortcuts. Therefore, our focus will be on trees, although the main ideas of our technique fully apply to bushes, and all those plants whose leaves are small in comparison with their size.

Our objective was to develop a practical method that would run interactively for several dozen or even hundreds of trees with perceptually full detail using off-the-shelf hardware, priced at 200 EUR or less. The method must allow unrestricted movement on a horizontal plane around the trees at distances of up to 10 to 15 meters between the tree and the observer. At this distance, the projection of the tree crown is small enough that the projection covers only a part of the viewport, and that individual leaves can no longer be discerned. This restriction is not stringent and the method would be directly useful in important applications in driving simulations and architectural visualizations. It was not our objective to allow viewing the tree from viewpoints above and below the tree.

Our method fulfills and exceeds these requirements, taking advantage of the following empiric properties of human visual system:

- Stereo vision based on binocular disparity has limited precision: a distant tree appears to be a 3D object primarily due to motion parallax.
- The perception of parallax has limited precision; velocities are clustered, and zones which move with approximately the same velocity are recognized as individual solid objects. A branch is an example of such an object, if the tree is viewed from a distance.

- Full geometric complexity of a tree is neither recognized nor remembered.
- Humans can only focus on individual segments of the image at once.

We have combined the traditional mesh-based geometry rendering for the trunk and limbs of the tree with image-based rendering for the crown of a tree. The boundaries between the two different geometry representations are normally obscured by the foliage, facilitating this simple approach. The meshes of the tree trunk and limbs can be simplified with a large variety of mesh simplification algorithms [7].

For the tree crown we use an image-based multi-layered representation. Our layers are individually textured oriented quadrilaterals, we name them *slices*, using the same terminology as [4]. A set of parallel slices is *a slicing.* Rendering a slicing is an approximation to precise pixel warping. For a single crown, we use several slicings to facilitate rendering from arbitrary directions.

The second contribution of this paper is our approach to rendering, targeted at the existing hardware graphics accelerators. We interleave the rendering of two most appropriate slicings, and adjust the opacity of each slicing depending on its orientation towards the observer. Thus the transitions between slicings are kept smooth. Another benefit of this is a significantly reduced number of slicings required.

After a brief review of related work, we present our methodology, first revising the slicing representation and the procedure for texture generation. We then proceed to introduce the blending approach for slicing rendering. Our experience with the implementation and the performance measurements can be found in section 3. A discussion of the method and ideas for further work are presented in section 4. We conclude with a short summary of our work.

1.1. Related work

The currently predominant approach to vegetation rendering seems to be billboarding [3], where the trees are reduced to planar quadrilaterals that maintain their relative orientation towards the observer. With improvement of terrain and object rendering, the quality offered by billboards is no longer acceptable due to their planarity, invariance to rotation, and lack of parallax.

Another frequent approach is manual design of simplified geometry models. For example, a tree can be represented with a green lollipop or can be formed from two intersecting quadrilaterals textured with the same billboard texture. In recent years the tree models have become significantly more intricate, for example, individual branches are represented with foliage-textured curved sheets. However, we are not aware of any automatic tools for generation of such models, and manual modeling and segmentation are long and laborious processes. Automatic mesh simplification is currently not applicable to sparse tree models.

Geometric complexity, measured by e.g. the number of leaves, is a parameter to most automatic model generation tools. This way, it is possible to generate simple but unrealistic 500-leaf models of trees, often used in architectural visualization and high-end simulations. Similar low-detail models are sold by several vendors.

Recently, a number of more advanced image-based rendering methods have been proposed. Layered depth images [10] are one of the dominant approaches. Instead of specifying color and optionally transparency for each 2D location in an image, a layered depth image (LDI) contains an arbitrary number of depth pixels, where each depth pixel is described by color, 20-bit depth, and a splat index. Most of the examples presented by the researchers were LDI's of very detailed vegetation. The authors provide an implementation based on software rasterization which only runs on Intel Pentium III computers, but the frame rates in a small window remain relatively low (8-10 frames per second for 300 by 300 resolution on 300 MHz PC, as reported in [10]). The data files are very large, an LDI of a detailed model of a single chestnut tree is over 17 MB. A similar approach to LDI has been investigated by [2], specifically for tree rendering. Finally, it must be noted that texture mapping and billboarding are older forms of image-based rendering.

Another paradigm in image-based rendering is image caching. A layered impostor [8] is conceptually similar to a single slicing, but the impostor layers are generated dynamically. The implementations of layered impostors seem to be focused on representing solid objects, resulting in significantly larger numbers of layers. Graphics acceleration hardware is supported by the implementations. A single RGBA texture is used for each impostor, where the alpha channel of the texture is used to represent depth. Consequently, a single impostor can only be viewed from a relatively narrow region of space. Using a number of layered impostors to represent an object from arbitrary directions has been studied in [9]. The described method maps layered impostors evenly to the sides of a platonic solid encapsulating the object. It is noted that the number of impostors can be reduced by exploiting the object symmetry. However, trees are not very appropriate for image caching due to their prohibitive geometric complexity.

Perhaps the most promising real-time approach is based on volumetric textures, which are represented quite similarly as layered impostors, usually with 64 layers. In [4], the need for multiple slicings with different orientations is noted and the formulae for determining the closest slicing are provided.

2. Our approach

2.1. Slicings

Most trees can be decomposed into two structurally different entities: the trunk and limbs as "solid" entities on one hand, and the foliage and twigs in the crown of the tree as "sparse" entities on the other. The solid entities are perceived similarly as other solid objects, are of sufficiently low geometric complexity to be rendered quickly. In addition to that, generic mesh simplification methods [7] can be used to automatically reduce the complexity depending on the required level of detail.

Sparse entities contain the majority of the geometric complexity of a model. For the model in Figure 1, the trunk and large branches can be represented with 200 triangles, whereas the twigs and leaves are formed from 8093 and 11564 triangles, respectively. The nature of twigs and foliage offers few possibilities for mesh simplification. A leaf or a twig cannot be geometrically simplified to less than a triangle or a quadrilateral with the existing methods. Although the exact shape of leaves is usually not perceived, usage of round points for leaves or lines for twigs would be immediately noticed as an artifact.

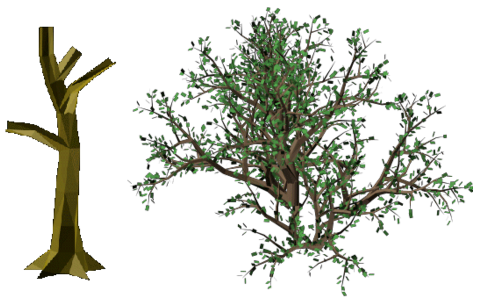
Figure 1: *The primitives of a tree model can be segmented into the solid set (200 triangles) and into the sparse set, where 8093 triangles are used for branches and 11564 for foliage.*

Fortunately, the geometric complexity of sparse entities is excessive also for accurate perception, and this offers possibilities for appropriate simplification. The effect of parallax is the necessary means for the illusion of depth, but as velocities in complex sparse objects are perceptually clustered, only a discrete numbers of parallax layers are sufficient to provide full sensation of depth, if viewing the tree from the approximately same direction. The sufficient number of layers is significantly lower than for solid objects.

Each depth layer lies on a fixed plane in the object space. In this paper it will be assumed that all layers, or *slices*, are parallel and equidistant. To distinguish a set of such slices from 3D textures, layered depth images, and layered impostors, we will refer to it as *a slicing*.

A slicing is displayed by simply rendering all the slices it is composed of. All the slices are located at fixed positions in the world space. The orientation of a slicing is fixed and is invariant to the observer location and orientation. Appropriate perspective warping uses the same pipeline as ordinary texture mapping.

Due to the assumptions of slice plane equidistance and parallelism, all the slices in a slicing have the same normal vector, which will be referred to as the slicing orientation vector $\mathbf{v}_s$, where $|\mathbf{v}_s|$ is the distance between slices. A slicing is bounded by a slicing box, two sides of which are the closest and the furthest slice. The slicing box is defined with the centroid $\mathbf{c}_s$, the height h_s and the width w_s. To be able to render each slice as a textured quadrilateral, its four vertices must be unambiguously determined. They lie at the intersections of the slice plane and the edges of the corresponding slicing box. It will be assumed that all slicings of an object share the same value of $\mathbf{c}_s$. By definition, each slice has only one side, but two slices with collinear orientation vectors may (and usually do) share the same slicing plane.

A slicing is displayed by back-to-front rendering of the textured quadrilaterals for each slice. As every texel in the textures is either full opaque or fully transparent, the rendering order does not matter. The use of MIP-mapped textures [3] improves the performance significantly.

2.2. Slice textures

Each small primitive p in the sparse part of the tree model is assigned to a single slice S_s for every slicing s on the basis of Euclidean distance of the centroid of the primitive $\mathbf{c}_p$ to the slice plane, $d(S_s, \mathbf{c}_p)$. The assignment occurs if $2d(S_s, \mathbf{c}_p) \leq |\mathbf{v}_s|$. A top-down view of the process is shown in Figure 2.

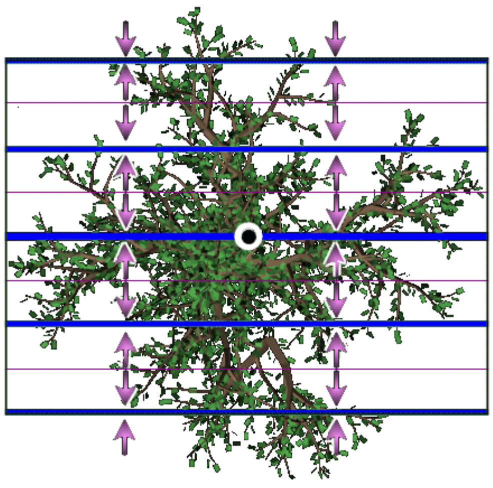
Figure 2: *Each primitive is assigned to the closest slice in the direction of the arrows. The slicing box encapsulates all the slices. The bull's-eye marks the slicing center.*

Although this simple mapping works well for small primitives, a slightly different procedure should be used for larger primitives, to avoid cracking and ordering artifacts. In such cases, the primitive should either be assigned either to several slices, or to the polygonal mesh.

The texture rendering pipeline performs the warping, so parallel projection is used for texture creation. The left, right, top and bottom sides of the slicing box are also the frustum planes. These sides of the box have to enclose all the primitives. The front and back slices do not have to enclose all the primitives, because the primitives are mapped inwards onto them.

The choice of the renderer is arbitrary, as long as the image can be exported as an RGBA image. However, the colors and lighting should match at the visible points of contact between the sliced representation of the tree crown and the mesh-based representation of tree trunk and limbs. If the textures are prerendered in a different environment, the colors must be aligned. This is often not trivial, as interactive rendering frameworks, such as OpenGL, cannot properly approximate the complex lighting model used by ray tracing software. The best method for computing the textures would probably be achieved by precomputing the vertex lighting in a radiosity package for the whole tree. Afterwards, this precomputed vertex lighting data is used to render the textures with the same rasterizer used for the interactive implementation. This approach ensures both quality and color alignment.

The required number of slices depends on the required interactive rendering quality, but 3 to 7 slices are usually optimal for a tree. Note that this is significantly lower than the theoretical estimates in [4], which do not consider the perceptual velocity clustering and sparseness. Also, multiple levels of detail could be created by using more slices if the tree is close, and perhaps a single slice if the tree is far and the parallax is no longer discernable. If a large number of slices is required for a very high level of detail, a different rendering method such as LDI's or completely polygonal representation should be considered. Slicing works best when the texture sizes are 256×256 or less. These are merely rules of thumb, dependent on the architecture of the graphics hardware and the nature of a specific application.

2.3. View Blending

A single slicing is sufficient for rendering a tree if the line connecting the slicing center $\mathbf{c}_s$ and the observer $\mathbf{o}$ is well-aligned with the slicing orientation $\mathbf{v}_s$. If this is not the case, for example, if the observer is facing the slicing in a direction perpendicular to the slicing orientation, the slices will appear as thin and glaring artifacts.

A single slicing does not contain enough information to facilitate 360 degree viewing around the vertical axis. For this aim, multiple slicings have to be computed, each with a different slicing orientation. For robust performance, it is desired that the orientations are evenly distributed, to allow displaying the tree from all applicable orientations. In majority of applications, such as road vehicle simulations and architectural visualization, most trees are always viewed from the approximately same relative height, while the movement on the horizontal plane is less obstructed. In such cases, the slicing orientations need to be evenly distributed points on a unit circle around the model, whereas in a general case, the orientations should be distributed on a unit sphere.

In certain applications, a particular tree is only viewed from a limited set of viewpoints, and the slicing orientation distribution can take advantage of this. However, it is important to note that the same slicing model can be instantiated in the world with different scale and orientation. Trees with evenly distributed slicing orientations are more appropriate for instantiation.

It is important to resolve which slicing is the most appropriate for a given observer location $\mathbf{o}$. For this, the relevant information is the *discrepancy angle* γ_s between the slicing orientation vector $\mathbf{v}_s$ and the direction vector from the slicing box centroid to the observer which is $\mathbf{o}-\mathbf{c}_s$. It is important to distinguish the camera orientation vector from the $\mathbf{o}-\mathbf{c}_s$ vector. Angle comparisons can be clearly done using the dot product.

An obvious approach might be rendering the single most appropriate slicing on the basis of the lowest angle of discrepancy, but this would result in abrupt switches between slicings. The obvious way of remedying this is creating a larger number of slicings, but this unacceptably increases the required amount of memory and reduces the performance. A more efficient approach is to again take advantage of the characteristics of the human visual perception, and find a way of smoothly blending between slicings.

Each slicing is positioned at a different place in the environment because of perspective warping of the slice textures, as Figure 3 shows. Two slicings thus cannot be superimposed on a single plane. We simply display the closest, primary, slicing and the second closest, secondary, slicing simultaneously, varying their individual levels of transparency. Two slicings are sufficient to provide smooth fading.

The transparency of the slicings is proportional to the difference between the discrepancy angles of the primary and the secondary slicing. As it was mentioned earlier, when the discrepancy angle of the primary slicing is 0, the secondary slicing should be fully transparent. When the discrepancy angles of the primary and the secondary slicing are the same, the blending coefficients should be equal for both slicings in order to smoothly perform the transition.

Of the commonly available blending functions, the most appropriate is $\mathbf{c}_t\alpha+\mathbf{c}_f(1-\alpha)$, where $\mathbf{c}_t$ is the superimposed texture color vector, $\mathbf{c}_f$ is the background color vector, and α is the opacity of the superimposed texture. This function is supported by the majority of 3D acceleration hardware

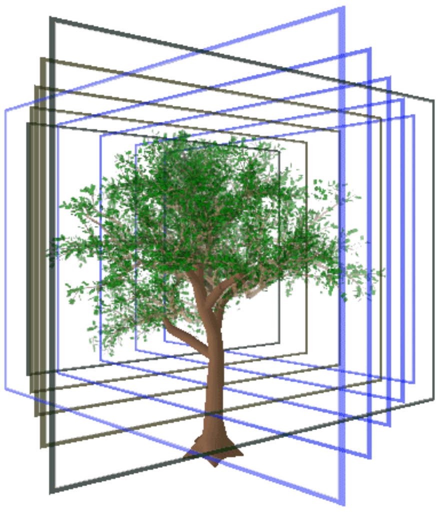

Figure 3: *Primary and secondary slicings are blended together to create a solid-looking rendering of a tree. The screenshot was taken after rectangular frames were added to the slice textures. This is a worst-case example, with both discrepancy angles close approximately 30 degrees.*

and recommended for similar purposes by [3]. A simple linear interpolation depending on the discrepancy angle works very well. As the discrepancy angle varies from 0 to being equal to the secondary discrepancy angle, the primary slicing opacity, α_p, varies from 1 to 0.5.

The equality of distortion angles occurs when both angles are equal to one half the angle between the primary and secondary slicing orientations, but if this assumption is used, the direction vector $\mathbf{o}-\mathbf{c}_s$ should be projected to the plane that both slicing orientation vectors are perpendicular to before computing the angle. The secondary slicing opacity α_s is then $1-\alpha_p$.

After determining the transparency levels for both slicings, they can be rendered. Most rendering pipelines require correct back-to-front ordering of the alpha blended primitives in the view. After rendering all the opaque primitives, all the semi-transparent primitives have to be rendered in the correct back to front order, but only if they overlap. In the case of trees, the mesh is opaque and the slicings are transparent. Note that although individual color-keyed textures of the slices themselves are opaque, the blending between two slicings results in semi-transparency. If we assume that trees do not intersect one another, the following rendering sequence is used: after all the trunks are rendered in an arbitrary order, the tree crowns are individually rendered from the furthest to the closest.

For each crown, the slicings should be also rendered in the correct order. The primary and secondary slicings usually intersect each other, but it is not necessary to add complexity by splitting them so that they could be correctly sorted. In fact, no sorting is necessary if the trees do not intersect: the rendering order should start with the furthest secondary slice, continued with the furthest primary slice, second furthest secondary slice, and so on, until the closest primary slice enters the rendering pipeline. However, when the primary and secondary slicings are swapped at the point of equal distortion angle, there is a very slight visible switch. It was observed that this artifact is not striking enough to justify additional computational cost of rendering a significantly larger number of primitives.

The blended slicings are slightly blurred, while individual slicings, rendered when the discrepancy angle is 0, are not. This may cause problems if the textures are sparse, and tree crown slicings are indeed sparse. An example of a consequence is that instead of displaying one opaque leaf in one slicing, it might happen that two half-transparent leaves are displayed next to each other. In addition to that, with the blending function used, the background behind the tree is still slightly visible, depending on the number of slices. This is not particularly problematic as in reality the tree crowns are naturally semi-transparent. The rotation of the tree is usually slow enough that blurring and sharpening are not bothersome.

The blurring and semi-transparency are thus not important artifacts. But the variation of perceived brightness, which is a consequence of the tree appearing semi-transparent as a whole in front of the background of a different color, needs to be corrected. To balance the brightness, the blending ratio was multiplied by an empirically determined linear correction factor $f_c = 1 + m_s\alpha_s$, where m_s is a correction multiplier which primarily depends on the sparseness of the tree crown and on the number of slices, and to a lower extent on the display gamma exponent and the background color. A good starting approximation for m_s is 0.5. Consequently the corrected slicing transparencies are $\bar{\alpha}_p = f_c\alpha_p$ and $\bar{\alpha}_s = f_c\alpha_s$. Of course, more complex correction factor formulae could yield better results.

Sometimes the slices in the back are almost completely occluded by the slices in front. If performance is at stake, and a small decrease in quality is acceptable, the last few slices do not have to be rendered.

3. Implementation

The method described above was implemented using the OpenGL API. We used an automatically generated tree model. The model was manually segmented into the solid and sparse parts. The mesh of the solid part was manually simplified to reduce the number of polygons, whereas the primitives in the sparse part were used to create the textures. Automatic segmentation would be relatively simple to implement within a software package for tree modeling.

Each slicing contained 5 slices. The 6 slicings were spaced 60 degrees apart. Symmetry was not exploited. Thus, the tree was rendered using only 40 vertices and 10 quadrilaterals. No lighting or shading was required. The trunk and the limbs were rendered as a triangular mesh with approximately 200 primitives. We have tried the method with 12 slicing orientations evenly distributed around the unit circle, but the expected quality improvement did not, while the expected performance drop did occur.

We used the unmodified POVRay ray tracing software and orthogonal projection to render the textures. The camera was oriented directly towards the slicing box centroid. The frustum was composed of the top, bottom, left and right sides of the slicing box. This guaranteed that the images exactly correspond to slice textures. The advantage of using ray tracing for texture creation was that detailed shading significantly improved the realism of the textures, which could not be as easily achieved by creating the textures using, e.g., the OpenGL renderer. However, the lighting parameters had to be adjusted to correspond to those used for rendering the textures with POVRay.

We have used MIP-mapping to optimize the performance. The color depth of the textures was reduced to 15 RGB bits and 1 transparency bit provide sufficient quality for half the memory requirements of 32 bit RGBA textures. Additional diversity of the landscape was achieved by instantiating the same tree model with different scale and orientation.

3.1. Performance

The implementation of our rendering method has been timed in comparison with the original polygonal model on a 350 MHz Pentium II computer with a NVIDIA GeForce 256 DDR graphics accelerator in 640×480 resolution. The crown of the polygonal model was rendered flat-shaded without lighting, whereas the trunks and limbs of both the polygonal and the hybrid model were rendered smooth-shaded with one positional light.

Meshes were represented with compiled display lists containing triangle vertex data. Triangles and not triangle stripes or fans were used in the display list, but the benefit would be limited with leaves. On the other hand, the crown mesh was plain and unshaded, which would not be acceptable in most applications. Only a single level of detail was used for both techniques.

The frame rendering time was sampled for a pre-recorded camera path consisting of 2600 frames through a grove with a specific number of trees. Not all trees were visible in each frame. After averaging the rendering time for subsequences of 20 frames, the rendering times for sequences were sorted in descending order. The first 5% of the largest rendering times after averaging were excluded, to ignore the artificial delays caused by the multitasking operating system. The largest remaining frame rendering time was taken as a realistic measure of the worst case performance.

The results in Figure 4 show that even with transform and lighting hardware, the hybrid representation with slicings is approximately 20 times faster than the mesh representation considering the worst-case performance. Average-case performance is not valid for comparison in interactive applications, but even with this measure, the mesh-based representation is consistenly and significantly worse than the hybrid representation as illustrated in Figure 4. The rendering quality can be compared in Figure 5, where the first tree is rendered as a shaded mesh with a point light source, the second and third trees as the slicing-mesh hybrid when the discrepancy angle is low or maximal (30 degrees), respectively.

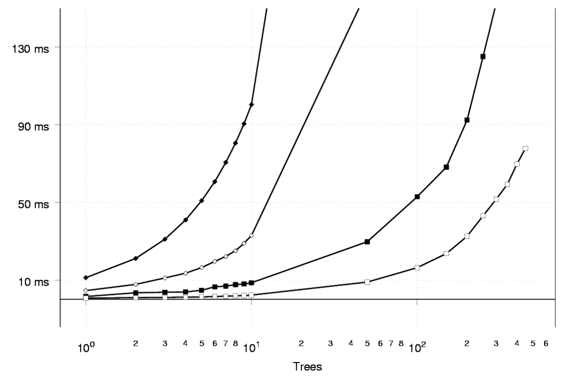

Figure 4: *Comparison of the average (empty shapes) and the worst case (filled shapes) frame rendering time for mesh (diamonds) and slicing (squares) representation of a given number of trees. The number of trees is displayed on a logarithmic scale.*

4. Further work

Computing a single discrepancy angle per slicing using its the slicing box centroid as the reference point is an approximation. In fact, the discrepancy angle could be computed for each individual point of a slicing, but the errors remain low if the slicing does not cover a large portion of the viewport. It is possible to imagine a variety of subdivision techniques to remedy this problem by bounding the error. A possible example might be subdivision of each slice into several subslices when close. All the subslices lie on the same plane and use the same texture, but the primary and secondary subslices are determined independently, as are their transparencies. It must be noted that zooming on slicings introduces other problems, mostly due to significantly larger texture memory requirements.

In our implementation, we focused on rendering trees for an observer located approximately on the ground plane. Even if the angle between the ground plane (where the tree is positioned) and $\mathbf{o} - \mathbf{c}_S$ was less than 45 degrees, the tree was rendered with good quality, as slicings similarly reproduce the effect of parallax for vertical movement. But blend-

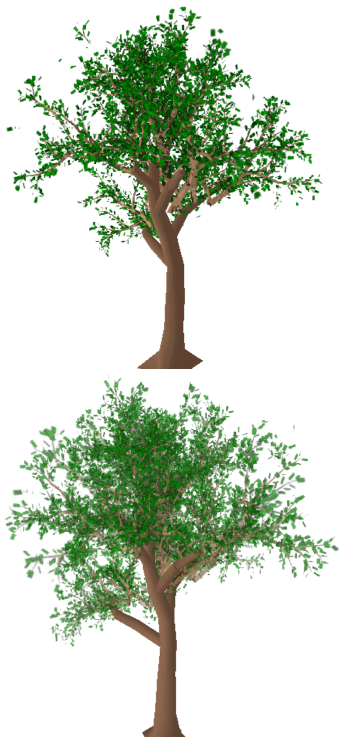
Figure 5: *Comparison of rendering quality for slicing-based representation with discrepancy angle of 0 degrees (top) and two blended slicings with discrepancy angle of 30 degrees (bottom). The pictures were taken at a very short distance, and the tree was filling the viewport completely.*

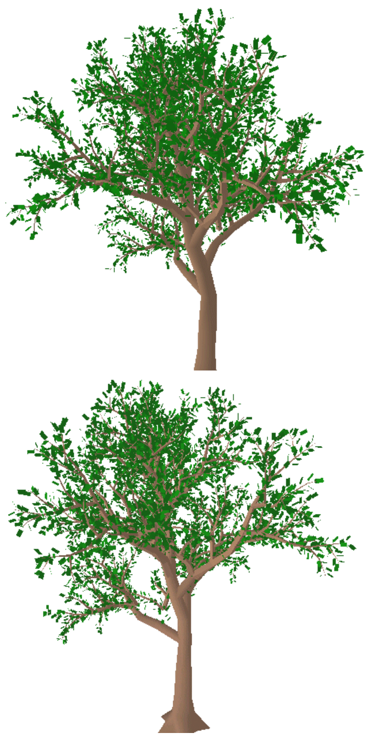
Figure 6: *The images of the mesh-based representation from the viewpoints corresponding to ones in the Figure 5.*

ing between two closest slicings might no longer be sufficient for an observer allowed to fly freely around the tree. When viewing the trees from the top, blending is not required, and a single vertically oriented slicing works well, as [4] has shown. A possible solution, other than distributing the slicing orientation vectors on a unit sphere, might be blending three slicings, two of them being horizontally and one of them vertically oriented.

An important parameter influencing the choice of the method is the distance from the observer to the tree and the time the observer spends focusing on a tree. From sufficiently far away or if moving very quickly, billboards are still the optimal choice. If very close up or if focusing on the geometric details of the branches, slicings may not offer sufficient quality, and one should resort to polygonal or LDI representations. An interesting challenge are methods for smooth blending between these diverse representations.

The most important computational bottleneck of our method is the consumption of texture memory, which limits the resolution of slice textures and the variety of models. For satisfactory performance, it is important to keep most of the textures in the texture memory of graphics board. If this condition is not fulfilled, the performance falls rapidly. For our model with 256×256 textures, approximately 4 megabytes of texture memory are required if no compression is used. Better memory bus architectures as well as better MIP-map memory management should lessen the problem.

Memory consumption could be reduced by using texture compression, supported by modern graphics hardware. Slice textures are also relatively sparse, with large transparent regions. Several software-based compression algorithms could be applied, for example quad-tree image compression algorithms. The fully transparent leaves of the quad-tree have to be neither rendered, nor stored as textures. Lower fill rate requirements and lower consumption of texture memory are gained by increasing the number of primitives, so the optimal balance depends on the hardware platform and the texture size.

It is possible to exploit symmetry to further reduce the amount of texturing memory required. A model can contain a number of planes of symmetry, and textures are symmetric across them. In addition to this, two slicings could be made from one by creating a mirrored instance of each slice. This is relevant if lighting is even from all the sides and if the depth order of primitives for each slice is not significantly influencing the look of the texture, for example, if there is little occlusion between primitives.

Plant model generation tools that support slicings would be useful. These tools have complete information about the segmentation of the model, and a single branch could be represented with a single appropriately oriented slicing, which would yield better quality than creating a slicing from the whole tree, where a single branch might appear in multiple slicings.

Our implementation does not yet support lighting and shadows. Precomputed shadow maps and bump mapping would be relatively simple and effective solutions, without causing a significant slow-down. An interesting direction for further work is also animation of the effect of wind on vegetation, for example by moving or warping individual slices.

Slicing representations offer a new perspective for automatic acquisition of natural environments for specific applications. Namely, it is sufficient to assign a pixel to the nearest slice in each slicing to achieve realistic interactive rendering, rather than determining the depth exactly.

5. Conclusion

Vegetation has often been pinpointed as something that cannot be properly rendered in interactive 3D applications. Although there have been solutions for interactive vegetation rendering when the landscape is viewed from bird's-eye view, there have been no solutions for applications that require greater freedom of movement.

In this paper we presented an approach based on segmenting a complex model of the tree into two heterogeneous representations. The tree trunk and limbs are represented as an usual triangular mesh, whereas an image-based representation based on slicings is used for amorphous and sparse foliage and twigs. A slicing is a set of equidistant, parallel and individually textured quadrilaterals, placed into the scene as fixed geometry. A slicing can be efficiently rendered with graphics hardware.

Multiple slicings can be used for complete depiction of objects. We described a method for smooth blending between two closest slicings, which allows the sliced object to be interactively rendered from a variety of viewpoints. This operation is also fully supported by graphics hardware.

Slicings were designed to work optimally for trees at distances from 10 to 50 meters away, when the tree is sufficiently far away that its projection covers only a part of the viewport, which ensures low discrepancy angle error, low texture memory, and low fill rate requirements. Closer, other representations may offer better quality, and further away the quality of billboards might be satisfactory.

This range of distance is most frequent in applications involving roadside trees and urban parks. It is also appropriate for smaller plants in indoors architectural visualizations. The method is sufficiently quick to allow groves of 100 trees to be rendered at 20 frames per second with current off-the-shelf hardware. The rendering quality is subjectively better than that with the original mesh. Rendering 100 mesh-based trees would take 1 second with the same hardware.

Acknowledgements

The author would first of all like to thank Aleš Leonardis for many valuable conversations, much support and advice. Iztok Bajec has provided access to hardware and offered much help with the performance measurements. The tree model was created with Tree Druid from Zenstar Software.

References

1. B. Lintermann and O. Deussen. Interactive modeling of plants. *IEEE Computer Graphics and Applications*, **19**(1), January/February 1999. 1
2. N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. *Eurographics Workshop on Rendering 1996*, 165–174, June 1996. 2
3. T. McReynolds and D. Blythe. Advanced Graphics Programming Techniques Using OpenGL. *SIGGRAPH 99 Course*, August 1999. 2, 3, 5
4. A. Meyer and F. Neyret. Interactive volumetric textures. *Eurographics Rendering Workshop 1998*, 157–168, June 1998. 1, 2, 4, 7
5. F. Neyret. Synthesizing verdant landscapes using volumetric textures. *Eurographics Workshop on Rendering 1996*, 215–224, June 1996. 1
6. P. Prusinkiewicz and A. Lindenmayer. The Algorithmic Beauty of Plants. Springer-Verlag, New York, 1990. 1
7. E. Puppo and R. Scopigno. Simplification, LOD, and Multiresolution - Principles and Applications. *Eurographics'97 Tutorial Notes*, 1997. 2, 3
8. G. Schaufler. Per-object image warping with layered impostors. *Eurographics Rendering Workshop 1998*, 145–156, June 1998. 2
9. G. Schaufler. Image-based object representation by layered impostors. *ACM Symposium on Virtual Reality Software and Technology '98*, 99–104, November 1998. 2
10. J.W. Shade, S.J. Gortler, L. He and R. Szeliski. Layered depth images. *Computer Graphics (SIGGRAPH 98 Conference Proceedings)*, 231–242, July 1998. 2
11. J. Weber and J. Penn. Creation and rendering of realistic trees. *Computer Graphics (SIGGRAPH 95 Conference Proceedings)*, 119–128, August 1995.

1