Effective Multi-resolution Rendering and Texture Compression for Captured Volumetric Trees

Christian Linz, Alex Reche, George Drettakis, Marcus Magnor

Eurographics Workshop on Natural Phenomena (2006)
E. Galin, N. Chiba (Editors)

Effective Multi-resolution Rendering and Texture Compression for Captured Volumetric Trees

Christian Linz[1], Alex Reche-Martinez[2], George Drettakis[2] and Marcus Magnor[1]

[1]Institut für Computergraphik, TU Braunschweig, Germany
[2]REVES/INRIA, Sophia-Antipolis, France

Abstract
Trees can be realistically rendered in synthetic environments by creating volumetric representations from photographs. However, volumetric tree representations created with previous methods are expensive to render due to the high number of primitives, and have very high texture memory requirements. We address both shortcomings by presenting an efficient multi-resolution rendering method and an effective texture compression solution. Our method uses an octree with appropriate textures at intermediate hierarchy levels and applies an effective pruning strategy. For texture compression, we adapt a vector quantization approach in a perceptually accurate color space, and modify the codebook generation of the Generalized Lloyd Algorithm to further improve texture quality. In combination with several hardware acceleration techniques, our approach achieves a reduction in texture memory requirements by one order of magnitude; in addition, it is now possible to render tens or even hundreds of captured trees at interactive rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Modeling and rendering trees has been a goal of computer graphics research since the early days of the field [Blo85, dREF*88, PL90]. While most of the effort has been in solutions to generate entirely synthetic trees (e.g., [PL90, dREF*88, DHL*98]), an alternative is the approach to capture and render real trees [SRDT01, RMMD04]. For both synthetic and captured trees, however, polygonal representations (mainly of the leaves) result in objects which are very complex and thus expensive to render. In addition, generating geometric levels-of-detail (LOD) for disconnected triangle meshes, such as the leaves of a tree, is an unsolved problem; the few solutions proposed to date require mixing various different representations(e.g., [Ney98,MN98,BCF*05]). However, trees are a good candidate for volumetric representations [RMMD04]; one big advantage of such an approach are appropriate multi-resolution LOD structures resulting naturally from the hierarchical data structure representing the volume.

Although Reche et al. [RMMD04] did use a volumetric representation, no multi-resolution solution was presented, and the texture memory requirements were prohibitively high. Despite the realistic renderings provided by the ap-

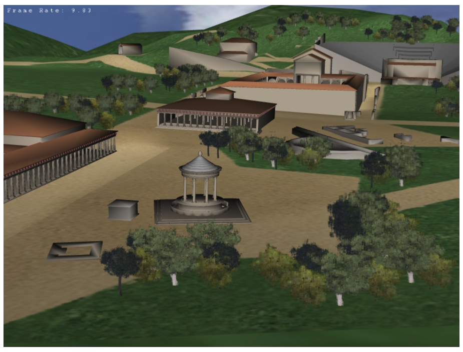

Figure 1: *A scene with 290 trees running at 12 fps, and requiring 2.9 MB of texture memory for 3 different types of trees. Using the previous approach [RMMD04], several seconds are required per frame and 641 MB texture memory are needed.*

proach, the method remains unusable for all practical purposes (60,000-140,000 polygons and 60-140MB of texture memory per tree).

In this paper we present solutions to both the rendering speed and the texture memory problems. We present an efficient multi-resolution rendering approach, in which we choose the appropriate data-structure by creating textures for each level, Sect. 3. In addition, we employ an efficient pruning strategy based on the properties of the generated textures. We then present a modified texture compression approach, choosing an appropriate color space during compression, Sect. 4. To improve the results, we introduce a modification to the Generalized Lloyds Algorithm used during codebook generation for vector quantization. Finally, we use several graphics hardware acceleration techniques which allow us to achieve better performance and texture compression rates, Sect. 5.

Overall, our technique allows us to render complex scenes containing tens or hundreds of trees at interactive frame rates. Texture memory consumption is reduced by two orders of magnitude. For example, the scene in Fig. 1 shows a scene with three types of trees using a total of 2.9 MB of memory, running at 10 fps. Using the previously existing approach, each frame would take tens of seconds to render, and 641 MB of texture memory would have be required. We believe that with these improvements, captured volumetric trees will become an interesting solution for games and other interactive 3D applications.

2. Previous Work

In the interest of brevity, we will restrict our discussion to a selection of the most relevant previous work. Most previous methods concentrated on entirely synthetic trees based on procedural methods such as grammars (L-systems) (e.g., [PL90, DHL*98]) or rule-based plant growing systems which use codified botanical knowledge such as the AMAP system [dREF*88]. Such approaches have been used to create highly realistic images of forests and trees, albeit with high polygon counts.

Other than the method of Reche et al. [RMMD04] (described in Sect. 2.1 in more detail), methods for capturing real trees include [TKN*92], based on two photographs with emphasis on shading, and Shlyakhter et al. [SRDT01] who use a visual hull created from photographs of the tree. They then fit an L-system to generate a polygonal model, while leaves are textured by re-projecting the photographs onto the polygons. As was the case for the synthetic trees mentioned above, the resulting models have high polygon counts; in addition, level-of-detail mechanisms are hard to develop for such representations.

Several image or volume-based rendering methods have been proposed for trees. The multi-layer z-buffer method uses precomputed synthetic images of trees [MO95, Max96]. In volumetric texture approaches, the complex tree geometry is represented as an approximation of the reflectance at a distance [Ney98]. An adaptation of this approach to hardware was developed later using textured slices for interactive rendering [MN98]. Meyer et al. [MNP01] presented a hierarchical bidirectional texture solution for trees at different levels of detail, resulting in efficient level-of-detail rendering for trees. Another approach has been developed in [QNTN03], in which a volumetric approach effects an implicit level-of-detail mechanism, for lighting (both sun and sky) and shadows, using shadow maps. Efficient rendering of trees can also be achieved using point-based methods [DCSD02]. Mantler and Fuhrmann [MF03] propose a view direction based method to reduce the amount of points to be rendered and achieve impressive savings in memory and rendering load. More recently billboard clouds [BCF*05, FUM05] have been used for rendering trees. All of the above techniques are applied to polygon-based synthetic trees. As such they could be applied to the captured trees of Shlyakhter et al. [SRDT01], but it is unclear how these could be applied to volumetric trees.

2.1. Volumetric trees

Our rendering and texture compression approach builds on the method of Reche et al. [RMMD04]. For clarity, we summarize the method here in more detail.

Tree capture proceeds in three steps. Initially, a set of photographs is taken from around the tree, and the cameras of these images are calibrated. Then, alpha-mattes are extracted from the images, giving an opacity value to each pixel in each view. In a second step, the opacity values are used to perform an opacity estimation on a hierarchical grid, similar to tomography, resulting in the assignment of a density value for each grid cell. The grid used in [RMMD04] was a tri-grid, i.e., each cell is subdivided into 27 children. The degree of refinement of the grid directly influences the quality of the reconstructed volume, where higher refinement allows the reconstruction of finer details. In the final step, textures are generated using a heuristic based on the input images, the depth of the cell in the tree and the alpha/opacity values. The generated textures are then assigned to a billboard in each cell. There is one texture per billboard per input camera position. To render a novel view, the cells are traversed in back-to-front order. The billboards generated from the two closest input cameras are weighted and blended together in the sense of the **over** operator. The two closest cameras are computed once per frame in software.

As mentioned above, despite high-quality tree renderings, this method suffers from high texture memory requirements and the lack of multi-resolution rendering. We address both shortcomings with our new approach.

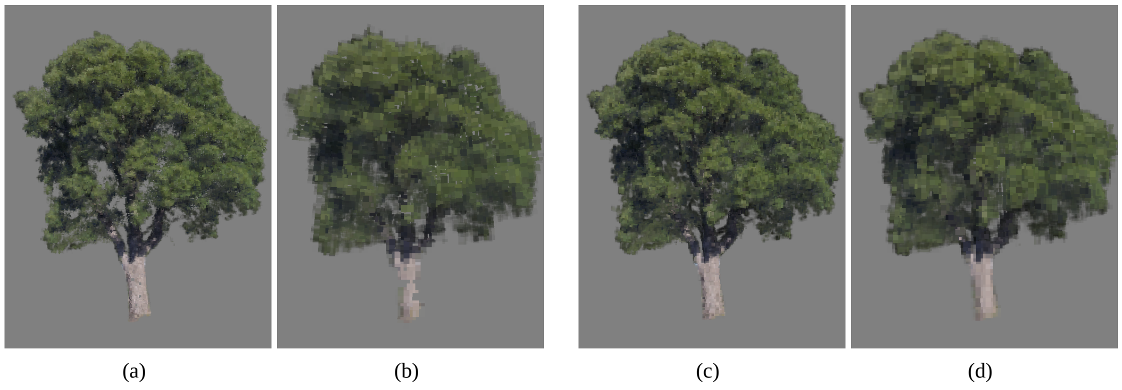

Figure 2: *Switching from (a) level 4 (51,000 polygons) to (b) level 3 (3,800 polygons) using a tri-grid (average RMS=31.6228). Octree representation with a switch from (c) level 6 (34,000 polygons, pruned 16,000 polygons) to (d) level 5 (6,300 polygons, pruned 3,800 polygons). Note that the transition is less abrupt (average RMS=24.5589).*

3. Multiresolution Rendering

We discuss here two main elements needed to achieve efficient multi-resolution rendering. The first is the choice of the appropriate hierarchical data structure and the generation of the corresponding textures, as well as how to choose the appropriate level of detail. The second is an efficient pruning strategy, based on the properties of the generated textures.

3.1. Using an Octree for Multiresolution Rendering

Our goal is to provide a smoothly varying level-of-detail (LOD) mechanism for tree rendering. The volumetric representation is based on a hierarchical data structure. Thus LOD can be achieved naturally by choosing and rendering the appropriate levels of the data structure.

The tri-grid structure used in [RMMD04] is inappropriate for multiresolution rendering since switching from one level to the next involves replacing a single cell (and the corresponding billboards) by 27 sub-cells. This leads to large jumps in the number of primitives, resulting in irregular frame rates. It also produces very visible transition artifacts for the textures which also cannot be avoided by a dissolve in the sense of [Max96]. We choose to use an octree; as a result the jumps in number of primitives are not as large as with the tri-grid structure, and the transitions between different levels of detail are less visible, especially for the lower LODs. Fig. 2(a,b) compares two neighboring levels of the tri-grid hierarchy with neighboring levels of the octree hierarchy (c,d). The artifacts are more clearly visible in the the accompanying video. In the original approach, no provision was made to create billboards and assign textures at *intermediate* nodes of the hierarchy. We use the same texture generation process as in [RMMD04], but at each level of the hierarchy. An alternative would be to average the textures from the lower levels: however, the overhead of texture computation of the intermediate levels corresponds to 37.5% of the total texture generation time. We considered that the tradeoff was worthwhile, since the resulting intermediate level textures are of higher quality.

The selection of the level of detail to be used is based on the distance of each cell to the current camera viewpoint. We set up a fixed number of planes, orthogonal to the camera viewing direction before each rendering pass. During rendering, for each cell of the octree structure, we check whether its center point lies in front of or behind the current LOD selection plane. If it lies in front of the plane, the cell is rendered at the currently active LOD. Else the tree descent stops one level above the currently set LOD, replacing eight cells by their parent cell.

3.2. Efficient Pruning Structure

In the original method [RMMD04], rendering speed was hindered by the large number of billboards to be rendered. In addition to the multi-resolution, Sect. 3.1, a basic optimization can be performed by better understanding the properties of the textures associated with the billboards attached to each cell. We prune the billboards that do not contribute to the rendered result. For a given cell and a given viewpoint, there is no need to render a billboard if the texture contains no color information. Thus, it can be pruned. Our method is comparable to the view direction based data reduction proposed in [MF03]. After careful study of the generated textures, we realize that this occurs quite frequently using the texture generation heuristic of [RMMD04]. During the texture generation process, for each cell we check whether it is visible from a given point of view. We trace a ray through the volume and accumulate the alpha values until we hit the cell. If the accumulated alpha values of the cells hit by the ray exceed a threshold, the cell is essentially invisible from the given viewpoint. The heuristic for texture generation uses the alpha value in its determination of color;

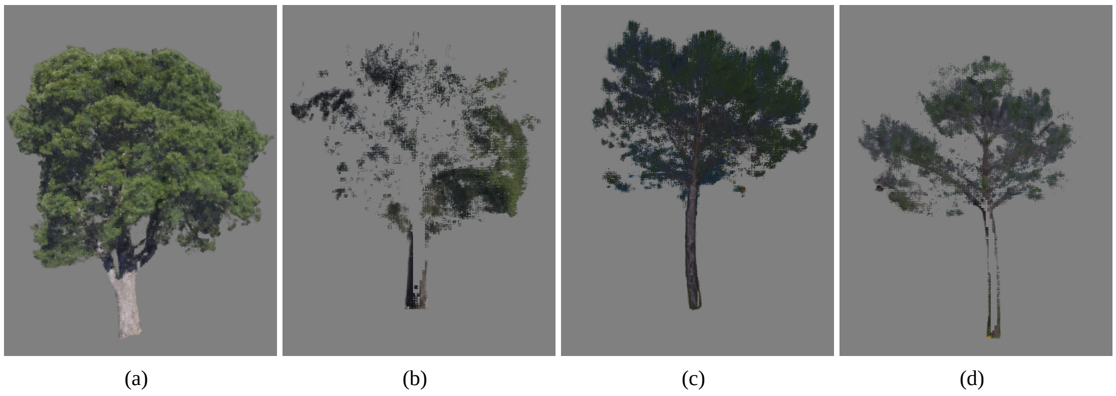

Figure 3: *(a) A tree with a dense crown, (b) seen from the side with the foremost half of the volume cut away (camera is to the right). (c) A tree with a sparse crown, (d) again seen from the side with the foremost half of the cells of the volume cut away.*

as a result, for trees with a dense crown, no colors are assigned to the textures of many of the interior cells or of those cells on the opposite side of the viewpoint. This is clearly illustrated in Fig. 3(a) and (b). In (b) we render a view where we "slice away" the front half of the tree. We can clearly see that most interior cells contain empty textures, and have been pruned. For trees with a sparse crown, Fig. 3(c),(d), this strategy also works, although in a less aggressive manner. A graphical illustration is given in Fig. 4, showing significant improvement for trees with a dense crown; the gain is lower for trees with very sparse crowns. The pruning strategy is applied recursively to the entire octree structure.

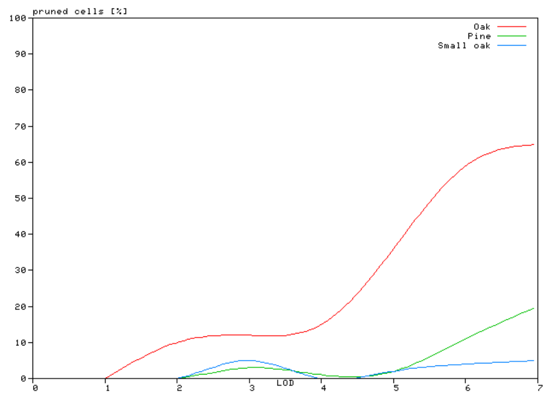

Figure 4: *Statistics on the percentage of pruned cells per level of detail and tree. For trees with a dense crown (oak), our strategy prunes up to 65% of the cells. For the small oak, the percentage of pruned cells is lower since the crown of the tree is very sparse and the alpha estimation already cuts away most of the interior of the tree canopy.*

4. Texture Compression

The textures generated by the volumetric approach [RMMD04] are of the order of 100-150MB of texture. The packing method reported there did not actually reduce the texture memory required at run time. The reduction reported was for offline storage purposes only.

Since our multiresolution approach makes it necessary to create texture information for the lower levels of detail, more memory may be required, making it even more important to reduce texture memory.

The change from the tri-grid to the octree means that there are typically more levels in the hierarchy, which may result in an increase in the number of billboards. Fortunately, this is partially compensated by the decrease in billboard texture size, from 8x8 (typically used in [RMMD04]) to 4x4. The resulting texture memory is often actually reduced (see Section 6). Nevertheless, the memory requirements are still too high for the use in computer games or other applications.

Therefore, we additionally employ an approach for texture compression proposed by Beers et al. [BAC96]. It is based on vector quantization (VQ) and offers high compression ratios with little loss in quality. We modify their approach to use a perceptually oriented color space such as CIELab for the computation of the texture codebook. We will elaborate on the details in the following subsections.

4.1. Texture quantization

The most crucial part of compressing a texture using VQ is designing the codebook. As in [BAC96], we employ the Generalized Lloyd Algorithm (GLA), an iterative clustering algorithm which yields a locally optimal codebook for a given set of texture blocks, the training vector set. Our training vector set consists of all 4x4-billboards, encoded as vectors. The algorithm starts with a set of potential codewords from the training set and iterates on the following steps. Each texture block is grouped with the nearest codeword according to a given distance measure. The centroids of the clusters are taken as the new codewords for the next iteration. This process repeats until the set of codewords converges.

The choice of the color space is important at this point. While the use of RGB space with the L_2-norm as a distance measure is simple and fast, GLA often groups textures that are close according to the distance measure but are of different perceptual colours. To counter this problem, we transform the billboard textures to the more perceptually oriented color space CIELab and use the L_2-norm as a distance measure.

The resulting compression has higher overall quality. This can be seen in the comparison in Fig. 5 for the oak tree. In (a) we see the tree with the original uncompressed texture. In (b) we see the compressed texture using RGB space. Clearly, RGB compression results in loss of contrast and overall lower visual quality. In (c) we use the CIELab space. The quality is higher, and contrast is better preserved.

4.1.1. Alpha channel quantization

The alpha channel quantization is straightforward. We encode each billboard alpha texture into a vector of grey level values and run the generalized Lloyd algorithm (GLA) on this data. Since we quantize the alpha channel independently of the RGB channel, we have to ensure that after quantization, every non-transparent pixel maps to a non-black pixel in the associated color texture. We address this problem by a heuristic that replaces every black pixel in the quantized texture with the color of the brightest pixel, computed from every non-black pixel of the billboard texture. Using the brightest pixel in this heuristic avoids black pixel artifacts in the rendition of the compressed tree.

4.1.2. Color channel quantization

We encode the color channels of each billboard texture into an appropriately sized vector. Afterwards, GLA is run on this vector data. The number of clusters is given by the user and trades compression ratio against quality of the compressed textures. We illustrate this tradeoff in Fig 6.

A drawback of the quantization method of [BAC96] is the implicit averaging of colors introduced by the GLA cluster computation, leading to overall darker textures, as well as loss of contrast in the textures. Our solution is to modify the GLA to replace the cluster centroid by the closest original input vector. If we compare Fig. 5(c) to Fig. 5(d), we can see that in the center of the tree certain regions have preserved their bright areas.

To avoid extensive texture context switching, the quantized alpha textures as well as the quantized RGB textures are organized in a texture atlas, typically of size 512x512. Decompression consists of simply computing appropriate texture coordinates in the codebook atlas for each billboard.

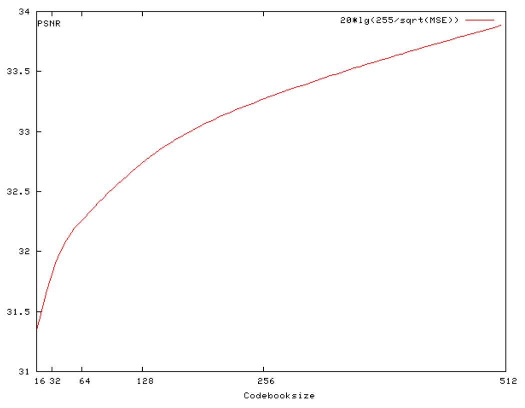

Figure 6: *Average peak signal noise ration (PSNR) of the compressed textures as a function of the codebooksize. Higher PSNR values indicate a lower error in the quantization.*

4.2. Other compression methods

Principal component analysis (PCA) [Jol80] is another possibility for texture compression. Given a billboard size of 4x4, we are able to represent the entire set of billboards with 64 eigentextures and one mean texture for the entire RGBA texture. In order to reconstruct the original billboard, we additionally store 64 coefficients per billboard. While this approach offers higher compression rates than VQ without loss in quality, we do not recommend this strategy since the reconstruction of every billboard requires multiple rendering passes per billboard. Given the complexity of the trees with respect to the number of billboards, this is prohibitive for interactive frame rates.

Another popular approach to image compression is the discrete wavelet transform (DWT) [Add02]. While this method also offers good compression ratios for large 2D images, it is not advantageous here due to the large number of small billboards.

5. Hardware Optimizations

Almost all information needed to render the tree is static. We thus assemble this information once in a preprocessing step and setup the graphics hardware accordingly. We transfer all static per-vertex information once to the fast graphics hardware memory using vertex buffer objects. Moreover, we push computations such as the correct orientation of each billboard onto the graphics hardware using Cg. During rendering, we traverse the underlying structure in a back-to-front manner, collect indices into the vertex buffer objects and set up texture coordinates appropriately.

Using OpenGL, texture compression is available via the extension `GL_EXT_texture_compression_s3tc`.

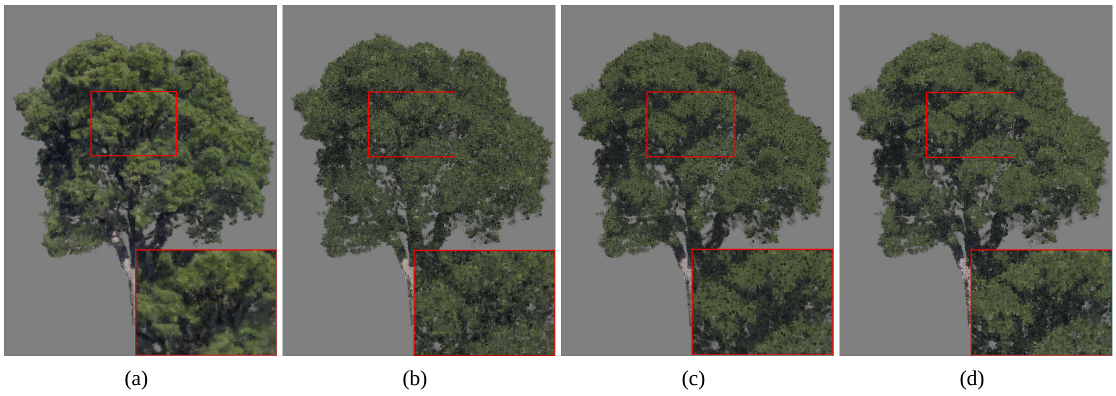

Figure 5: *(a) Tree with uncompressed texture. (b) Quantization using RGB space with visible errors (average RMS=30.1971). (c) Quantization with CIELAB space, with original GLA algorithm (average RMS=28.1523). (d) Modified GLA algorithm using CIELAB space (average RMS=28.1017). The RMS is computed with respect to (a) and averaged over the colour channels.*

The technical details of this compression scheme can be found in [Ope]. We use DXT5 texture compression which gives an additional compression ratio of 4:1.

6. Results

All results described here are run on a Linux Fedora PC with an NVIDIA 6800 graphics card with 128MB of texture memory, and a 3.06Ghz Xeon CPU. We have tested our approach using three tree models: the oak and pine which were also used in [RMMD04], and the additional small oak model presented in Reche's thesis [RM05]. We will always refer to these trees in this order unless explicitly stated otherwise. We compare the various improvements to the basic, non-hierarchical tri-grid algorithm of [RMMD04].

Before pruning, the corresponding number of billboards is 154,000, 54,000 and 35,000 for a 7-level octree subdivision. The corresponding numbers for a 5-level tri-grid are 361,000, 152,000 and 114,000. After pruning (Sec. 3.2), the *average* number of billboards is 53,000, 43,000 and 33,000, computed by rotating around the tree. In terms of rendering speeds, the average frame rate for the three trees are: 1.5, 2.6 and 2.5 frames per second (fps) for the tri-grid, 4, 6 and 6 fps for the complete octree and 11, 11 and 12 fps for the pruned octree. As we can see, we have an average 78% reduction in the number of polygons and a 86% average speedup in rendering speed, for equivalent quality trees.

The texture memory consumption for a multi-resolution tri-grid version of the above trees is 266, 147 and 228 MB, before any compression (5-level). The corresponding octree texture memory consumption, before any compression is 72, 60 and 86 MB (7-level). Using our compression approach described in Sec. 4, the memory requirement is 3.1, 2.1 and 6.3 MB. We use a codebook of size 512 for the oak and pine and 1024 for the small pine, which has less contrast overall in the textures. Compared to the uncompressed tri-grid we thus achieve a 64:1 improvement in texture memory consumption (19:1 for the octree). Using the hardware compression, the actual texture memory consumption on the graphics cards is again reduced by a factor of 4, resulting in a 256:1 overall compression compared to the tri-grid (76:1 for the octree).

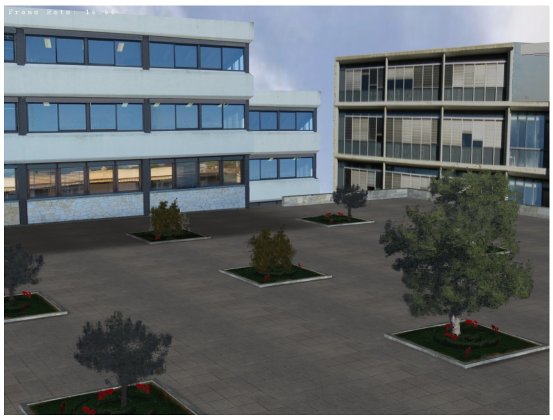

Figure 7: *Scene with 12 trees in a square; this scene runs at 20 fps. All three trees are present.*

The performance of the multi-resolution rendering algorithm is harder to compare, since there is no equivalent in the original algorithm. We show in Figs. 1,7 typical interactive “game-like" environments. The “square” scene renders at an average frame rate of 20 fps (max. 24 fps and min. 17 fps), while the scene of the Ancient Greek city of Argos renders at 10 fps (max. 25 fps and min. 7 fps) for the walkthroughs shown on the accompanying video. On average, 6 trees for the “square” scene and 180 trees for the city of Argos are inside the viewing frustum. All three trees are used. Using the original approach, this would require 641 MB of texture memory, rendering it unusable for common graphics cards. The total texture memory consumption using our approach is 2.9 MB, or a total compression rate of 240:1 in this case.

Although the improvements in rendering speed and texture memory consumption now allow for rendering hundreds of trees, it is still limited by the traversal of the underlying data structure, needed for correct back-to-front sorting of the billboards. This is done entirely on the CPU and the traversal time hence gives a lower bound for the speed of our method.

7. Conclusion and Discussion

We have presented a multiresolution rendering approach for captured volumetric trees, together with an efficient texture compression approach. In particular, we use an octree data structure which allows smoother multiresolution level-of-detail control. We employ generated textures at every level. We are able to significantly reduce the number of billboards required to represent the volume by pruning those which do not contribute to the image. For texture compression, we use a vector quantization scheme [BAC96] which is modified in two ways: we use a perceptually uniform color space (CIELab), and we modify the GLA algorithm for codebook generation, improving texture quality. The compression does result in a some quality loss (intensity and contrast levels) as evidenced in Fig. 5, but the overall quality is high.. It may be possible to further improve the quality by using alternative heuristics.

The approaches we have introduced, together with the use of a set of graphics hardware optimizations, reduce the texture memory required for the display of the captured volumetric trees by one order of magnitude (e.g., from 72 MB for the oak tree down to 3.1 MB). The multi-resolution algorithm allows the display of large numbers of trees in realistic settings for games or other interactive applications. In the previous approach [RMMD04] a single tree ran at 2 fps; in the examples we have shown, we are able to render environments with 290 trees at 10 fps on average.

The methods we have presented are a significant improvement over previous state of the art for captured volumetric trees. Without these, use of this representation was impractical for all realistic usage scenarios such as games etc. We believe that with the solutions presented here, captured volumetric trees will now be a viable and interesting option for real-world applications such as games etc., since the resulting trees are realistic and convincing, and can be displayed rapidly compared to other approaches.

In future work, we plan to address the two limitations of captured volumetric trees, that is fixed lighting, which is currently embedded in the input photographs used to generate the textures, and the fact that the trees cannot currently be modified, since they are an exact reconstruction of an existing tree. For both of these issues, it will be necessary to create a semantic representation of the tree by identifying the trunk, branches and leaves, allowing their manipulation both photometrically and geometrically. In addition, it would be beneficial to push the back-to-front sorting of the billboards onto the graphics hardware.

References

[Add02] ADDISON P. S.: *The Illustrated Wavelet Transform Handbook.* Institute of Physics, 2002.

[BAC96] BEERS A. C., AGRAWALA M., CHADDHA N.: Rendering from compressed textures. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM Press, pp. 373–378.

[BCF*05] BEHRENDT S., COLDITZ C., FRANZKE O., KOPF J., DEUSSEN O.: Realistic real-time rendering of landscapes using billboard clouds. In *Eurographics 05* (2005).

[Blo85] BLOOMENTHAL J.: Modeling the mighty maple. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1985), ACM Press, pp. 305–311.

[DCSD02] DEUSSEN O., COLDITZ C., STAMMINGER M., DRETTAKIS G.: Interactive visualization of complex plant ecosystems. In *Proc. IEEE Visualization 2002* (October 2002), pp. 219–226.

[DHL*98] DEUSSEN O., HANRAHAN P., LINTERMANN B., MĚCH R., PHARR M., PRUSINKIEWICZ P.: Realistic modeling and rendering of plant ecosystems. *Proc. SIGGRAPH'98* (1998), 275–286.

[dREF*88] DE REFFYE P., EDELIN C., FRANSON J., JAEGER M., PUECH C.: Plant models faithful to botanical structure and development. In *Proc. SIGGRAPH 88* (1988), pp. 151–158.

[FUM05] FUHRMANN A. L., UMLAUF E., MANTLER S.: Extreme model simplification for forest rendering. In *EG Workshop on Natural Phenomena 05* (2005), pp. 57–66.

[Jol80] JOLIFFE I.: *Principal Component Analysis*. Springer, 1980.

[Max96] MAX N.: Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In *Proc. 7th EG Rendering Workshop* (1996), pp. 165–174.

[MF03] MANTLER S., FUHRMANN A. L.: Fast approximate visible set determination for point sample clouds. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003* (New York, NY, USA, 2003), ACM Press, pp. 163–167.

[MN98] MEYER A., NEYRET F.: Interactive volumetric textures. In *Proc. 9th EG Rendering Workshop 1998* (July 1998), pp. 157–168.

[MNP01] MEYER A., NEYRET F., POULIN P.: Interactive rendering of trees with shading and shadows. In *Proc. 12th EG Workshop on Rendering, 2001* (Jul 2001), pp. 183–196.

[MO95] MAX N., OHSAKI K.: Rendering trees from precomputed Z-buffer views. In *Proc. 6th EG Workshop on Rendering* (1995), pp. 74–81.

[Ney98] NEYRET F.: Modeling animating and rendering complex scenes using volumetric textures. *IEEE Trans. on Visualization and Computer Graphics 4*, 1 (Jan.–Mar. 1998), 55–70.

[Ope] OpenGL Extension Registry. http://oss.sgi.com/projects/ogl-sample/registry/EXT/texture_compression_s3tc.txt.

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: The algorithmic beauty of plants. *Springer, New York* (1990).

[QNTN03] QIN X., NAKAMAE E., TADAMURA K., NAGAI Y.: Fast photo-realistic rendering of trees in daylight. In *Proc. of Eurographics 03* (Sept. 1–6 2003), pp. 243–252.

[RM05] RECHE-MARTINEZ A.: *Image-based Capture and Rendering with Applications to Urban Planning*. PhD thesis, Université de Nice, Sophia-Antipolis, 2005.

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

[SRDT01] SHLYAKHTER I., ROZENOER M., DORSEY J., TELLER S.: Reconstructing 3D tree models from instrumented photographs. *IEEE CG & A 21*, 3 (May/June 2001), 53–61.

[TKN*92] TADAMURA K., KANEDA K., NAKAMAE E., KATO F., NOGUCHI T.: A Display Method of Trees by Using Photo Images. *Journal of Information Processing 15*, 4 (1992), 526–534.