Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

MALTE CLASEN PHILIP PAAR[1] STEFFEN PROHASKA

Level of Detail for Trees using Clustered Ellipsoids

[1]Laubwerk GmbH

ZIB-Report ZR-11-41 (November 2011)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Level of Detail for Trees using Clustered Ellipsoids

M. Clasen[1] , P. Paar[2] and S. Prohaska[1]

[1]Zuse Institute Berlin (ZIB), Germany
[2]Laubwerk GmbH, Germany

Abstract

We present a level of detail method for trees based on ellipsoids and lines. We leverage the Expectation Maximization algorithm with a Gaussian Mixture Model to create a hierarchy of high-quality leaf clusterings, while the branches are simplified using agglomerative bottom-up clustering to preserve the connectivity. The simplification runs in a preprocessing step and requires no human interaction. For a fly by over and through a scene of 10 k trees, our method renders on average at 40 ms/frame, up to 6 times faster than billboard clouds with comparable artifacts.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image Generation - Display algorithms—Computer Graphics [I.3.6]: Methodology and Techniques - Graphics data structures and data types—

1. Introduction

Interactive plant visualization has many applications such as games, flight simulators, real-time preview for architectural modeling tools, and geovirtual visualization systems. Interactivity is, for example, considered important by landscape planners and landscape architects, who expressed their preoccupation with time-consuming rendering in a survey on applications and requirements of 3d visualization software [Paa06]. As important as interactivity is a sufficient degree of detail. Several studies have demonstrated that the degree of detail, particularly for foreground features like vegetation, soil surface, or water, is a key factor in how people relate to computer-generated visual simulations of landscape scenery [Lan01]. Achieving interactivity and a sufficient degree of detail at the same time for plants of realistic densities of vegetation cover is still a problem, in particular in densely vegetated areas such as fields and forests. Systems that are designed for interactive use, such as mainstream geographic information systems (GIS) for landscape planning and geo browsers such as Google Earth, represent vegetation cover to date only in a quite rudimentary way. Specialized landscape renderers that excel at high quality on the other hand, such as E-on Software Vue or Planetside Software Terragen, require non-interative rendering times.

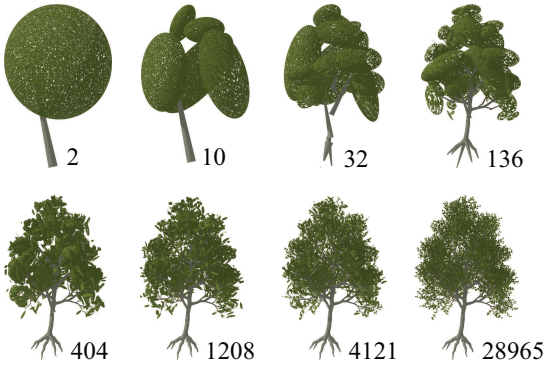

Figure 1: *Number of primitives for the levels of detail for a mesh with 123 k triangles.*

Level of Detail (LoD) can be used to balance interactivity and degree of detail. LoD methods reduce the complexity of 3d models and thus can reduce both the resource usage (memory, time) and aliasing artifacts, enabling the rendering of visually rich scenes at interactive frame rates. Many LoD methods focus on a reduction of surface complexity. Trees, however, with thousands of only loosely connected leaves, have a different geometric complexity than surfaces. So while the simplification of buildings from sophisticated

façades to flat rectangles is solved in numerous ways, rendering the vegetation surrounding the man-made structures is still a challenge. Specifically, rendering realistic (non-tiled) forests at interactive frame rates on commodity PCs is still not solved in a completely satisfactory way, as we discuss in more detail in the section on related work below.

In this paper, we present an LoD method for trees. The method is based on fuzzy clustering of unconnected leaf primitives, where each cluster is represented by a noise-textured ellipsoid. Our main contributions are, first, that an effective generic clustering method is used instead of the common ad-hoc techniques, which improves the accuracy of the lower LoD steps and reduces the number of primitives (Fig. 1) and therefore the GPU time. Second, we apply a noise texture to the ellipsoids for surface normals and alpha-test opacity, which yields a more natural look than the primitives used in previous work. Third, we propose an LoD selection based on primitive size in order to reduce aliasing.

We now discuss related work (section 2). Then we present the clustering schemes for ellipsoids and lines (section 3), followed by the image error metric used in the line clustering step (section 4), the primitive rendering (section 5), and the selection of the appropriate level of detail (section 6). This is followed by a comparison of performance and image quality to previous methods (section 7) and a discussion (section 8).

2. Related Work

Many plants have a high geometric complexity, with leaves loosely connected to a large number of twigs and branches. Boudon et al. [BMG06] classify a wide range of LoD methods developed for this special kind of 3d models. In their terms, our method is a multiscale approach (suitable for both near-field views and large scenes) using structural information with line primitives for the trunk and spatial information with ellipsoidal primitives for the leaves, similar to the work of Deussen et al. [DCSD02], Gilet et al. [GMN05], and Clasen et al. [CP10].

The most popular techniques to render plants are based on the generic Billboard Cloud (BBC) method introduced by Décoret et al. in [DDSD03], such as the work of Fuhrmann et al. [FUM05] and Behrendt et al. [BCF*05]. They use a set of textured impostors to transform the geometric complexity into planar images. Leveraging the GPU texture filtering capabilities results in low amounts of spatial and temporal aliasing, but at the cost of spatial deviations from the reference geometry. [CP10], on the other hand, is optimized for a low measured image error compared to a reference image using the HDR-VDP metric introduced by Mantiuk et al. in [MDMS05]. Since this metric does not discriminate different noise patterns with similar properties, the drawback is visible temporal aliasing (see Fig. 13). While there are effective techniques to reduce temporal noise in videos, for example by Kim et al. [KW97], they usually introduce a lag of a few frames, which limits the applicability to interactive applications. Given this trade-off, we tuned our method towards the behavior of billboard clouds, because measured image errors are usually less important in interactive applications than perceived artifacts. This can be done enforcing a lower limit on the point size in screen space, as proposed by Deussen et al. [DCSD02] and Gilet et al. [GMN05].

Clustering has been used by various LoD methods in different ways. Gilet et al. [GMN05] and Clasen et al. [CP10] rely on hierarchical bottom-up clustering of point primitives. [GMN05] is driven solely by a spatial data structure, whereas [CP10] uses an image error metric to merge the clusters. While the original billboard clouds in [DDSD03] and its optimization for trees in [FUM05] use ad-hoc clustering heuristics, [BCF*05] additionally maps the billboard cloud generation to the partitioning (non-hierarchical) k-means algorithm. We carry the idea of using a generic clustering scheme over to point-based LoD and use the Expectation-Maximization (EM) algorithm first presented in [DLR77] (of which k-means can be written as a special case) with a Gaussian Mixture Model (GMM) for clustering.

3. Building the LoD Hierarchy

To build the LoD hierarchy, we employ two different methods for leaves and branches. Leaves are represented by sets of clusters of ellipsoids, while a successively simplified line hierarchy is used for branches.

3.1. Ellipsoids

In the first step, leaves from the source mesh are converted to single ellipsoids by uniform sampling followed by a principal component analysis (PCA). To calibrate the visible area of the ellipsoids to the mesh, we compute the image error (Sec. 4) for multiple scaling factors and choose the minimum (See [CP10] for details).

Second, we build a LoD hierarchy on top of the single leaf primitives. We use a fuzzy partitioning clustering algorithm for each level, subsequently reducing the target number of clusters until only a single cluster is generated for the coarsest level. This yields more accurate representations for each level than the agglomerative bottom-up clustering used in previous methods such as [DVS03] and [CP10] (Fig. 2).

To improve the approximation, we use the Expectation-Maximization (EM) algorithm with a Gaussian mixture model (GMM) for clustering, which yields ellipsoidal clusters. We chose EM/GMM over the widely used k-means for two reasons: K-means uses a spherical cluster model and yields equi-sized clusters. Both properties do not match the typical structures of trees. Since the local optimum of EM/GMM depends on the initialization values, we run the clustering and the subsequent primitive creation (as follows)

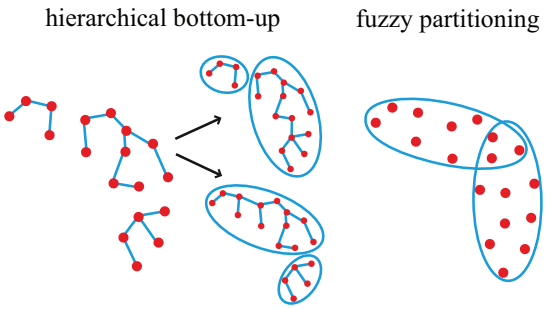

Figure 2: *Hierarchical bottom-up clustering tends to result in unbalanced clusters that depend on small variations in the input data, whereas partitioning can find robust global optima. Fuzzy cluster assignments further improve the approximation.*

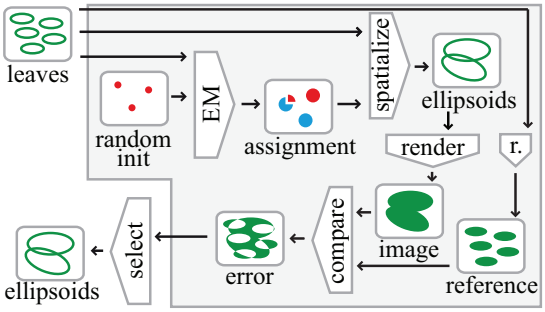

Figure 3: *To create a LoD, we first run the EM algorithm with a random initialization, convert the cluster weights to ellipsoids and measure the difference between the rendered ellipsoids and the source leaves. We repeat this several times and select the ellipsoids with the lowest image error.*

Algorithm 1: Building the ellipsoid cluster hierarchy H.

input : A set L of leaf ellipsoids; a number of tries t
output: A list H of sets of ellipsoids
$I_{ref} \leftarrow$ `RenderImage` (L);
$n \leftarrow \|L\| \cdot \frac{1}{4}$;
$H \leftarrow \emptyset$;
while $n \geq 1$ **do**
  $E \leftarrow \emptyset$;
  for $i \leftarrow 1$ **to** t **do**
    $P \leftarrow$ `CreateRandomPoints` (n);
    $W \leftarrow$ `FindEMClusterWeights` (P, L);
    $E_i \leftarrow$ `CreateEllipsoids` (W, L);
    $I \leftarrow$ `RenderImage` (E_i);
    $e \leftarrow$ `ComputeImageError` (I, I_{ref});
    $E \leftarrow E \cup (E_i, e)$;
  end
  $H \leftarrow H \cup$ `SelectOptimalEllipsoids` (E);
  $n \leftarrow n \cdot \frac{1}{4}$;
end

and error evaluation multiple times (Fig. 3, Alg. 1), where more than four times did not further improve the result. To select the best clustering based on the multiple random initializations, we then render the generated ellipsoids and compute the image error (Sec. 4) relative to the rendered source leaves. While multiple runs of the EM algorithm are usually evaluated based on the log-likelihood value, we prefer the image-error-based comparison because it is a better estimate of the image quality at run-time. Since most of the time is spent on the EM algorithm itself, the additional image rendering and comparison steps are negligible.

For each iteration, first we generate a set of random points in the bounding box of the leaves. The EM algorithm takes these as initial cluster center values. It then iteratively approximates the center points of the leaf ellipsoids with 3D

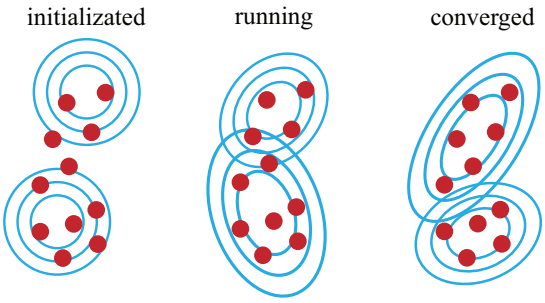

Figure 4: *The EM/GMM algorithm is initialized with random cluster positions and iteratively approximates the samples with normal distributions of arbitrary orientation.*

normal distributions (Fig. 4). Once the EM loop has converged, we use the resulting fuzzy cluster assignments $w_{i,j}$ (probability that point i is represented by cluster j; $\sum_j w_{i,j} = 1$) as weights for a PCA of the center points of the leaves (Fig. 5). Although the EM algorithm internally uses Gaussians, which could directly be interpreted as ellipsoids, we use PCA in a separate step to create a cleaner software architecture. This allows for varying implementations of the clustering method. Once the ellipsoids are generated, we discard all those whose surface area is less than 1% of the largest. This reduces both rendering time and aliasing due to tiny primitives that hardly affect the resulting shape. The size of the remaining ellipsoids is calibrated using the same method as for the initial import.

To avoid the artificial look of perfect ellipsoids, we add a noise texture. We create two mip-mapped cube map textures filled with white noise: One grey-scale texture as alpha channel and one RGB as normal map. A single pair of noise textures is sufficient for all models. The alpha channel texture is used to create alpha-test holes in the sur-

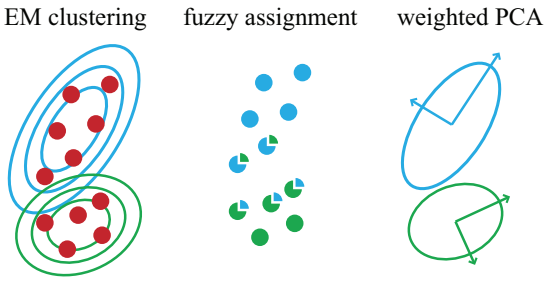

Figure 5: *The EM algorithm results in assignment probabilities for each sample. We use these as weights for a PCA to compute the cluster ellipsoids.*

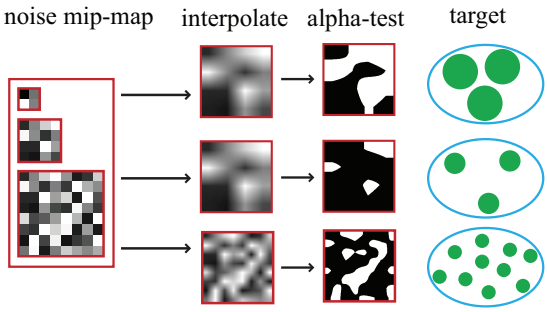

Figure 6: *We compute frequency and threshold for the ellipsoid noise texture based on the number and area of clustered leaves.*

face based on a threshold value t shared among all clusters of a single level of detail, and a cluster specific mip-map LoD level l_j (Fig. 6). The mip-map texture for level l has $2^l \times 2^l$ texels. For $t = 0.5$, half of the texels are discarded, so l_j represents about 2^{2l_j-1} features. The target number of features is given by the sum of leaf weights $\sum_i w_{i,j}$, so $l_j = 0.5(1 + log_2 \sum_i w_{i,j})$. The threshold t is determined using the calibration routine, where the image error (Sec. 4) is computed for multiple values of t and corresponding scaling coefficients for the ellipsoid area to compensate the area loss due to the alpha test. Mip-map lod level and blending coefficient of the normal map noise are computed in the same way. The blending coefficient is used to blend between the surface normal of the ellipsoid and a random unit vector from the normal map noise texture.

To get a LoD hierarchy, we run this loop for several steps, starting with a number of clusters of $\frac{1}{4}$ the number of leaves and further reducing this number by $\frac{1}{4}$ until only a single cluster is generated.

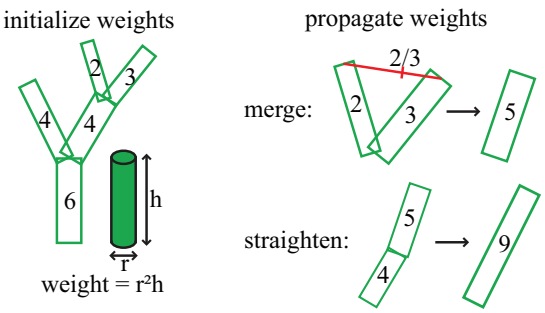

Figure 7: *We initialize line weights by primitive volume. We use the weights to interpolate in merge steps.*

3.2. Lines

For lines, we rely on agglomerative bottom-up clustering to preserve the connectivity. Starting from the imported skeleton, we successively merge two primitives in each step until only a single line is left. To determine the best merge candidate pair, we randomly select mergeable pairs and evaluate the image error of the resulting simplified model. Then we apply the pair with the lowest error. We use two merge operations, straighten and combine. The straighten operation replaces two consecutive lines by a single line, whereas the combine operation replaces two lines with the same parent line by an averaged line. We initialize the weights for the averaging operation on the imported skeleton by line volumes (Fig. 7). On each merge step, we assign the sum of the weights to the resulting line. While this follows the general idea presented in [CP10] (see for further details), using propagated weights instead of ad-hoc weights improves the quality of lower LoD steps. We do not use their crop operation because we observed no further improvements over merge/straighten only.

4. Image Error Metric

To measure the difference between two images, we use a root mean square error (RMSE) metric. RMSE is only sensitive to differences in single pixels. Since the perceived quality is also affected by large scale artifacts, we run the RMSE metric for multiple image resolutions. While perception-based error metrics seem to be a natural choice for comparing image quality, we found RMSE better suited in our case. Perception-based metrics such as HDR-VDP by [MDMS05] and MS-SSIM by [WSB03] are designed to ignore global differences in contrast and brightness, and emphasize the difference between correlated and uncorrelated noise. These kinds of errors do not appear in our controlled environment. On the contrary, it is a significant difference in the calibration step whether the primitives are invisibly small or cover the entire frame buffer. But MS-SSIM would detect no structural difference between these cases. Therefore we rely on RMSE,

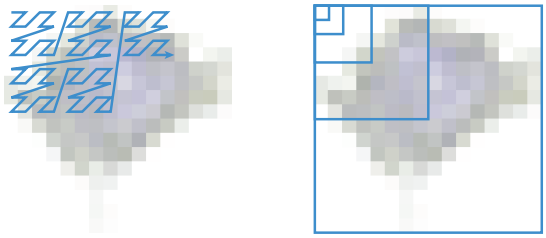

Figure 8: *To compute the RMS in multiple resolutions in a single pass, we use a space filling z-curve.*

which is also an order of magnitude faster (996 comparisons per second for 512^2 images on a Radeon HD4850, 88 cps for MS-SSIM in [CP10]). Our GPU implementation computes multi-scale RMSE in a single pass on the GPU by scanning the difference image (pixel-wise $img_{sample} - img_{reference}$) along a space-filling z-curve (Fig. 8). This way we can accumulate the errors for each resolution band with a single value per band, taking $log_2(n)$ values, where n is the width of the image.

5. Rendering

Like [CP10], we use a sequential point tree (SPT) similar to [DVS03] to store the primitives on the GPU. Since we do not have an inherent parent-child relationship between the ellipsoids, we compute the error thresholds for whole LoD steps (Sec. 6), not for single primitives. Therefore we lose the continuous LoD property. However, the steps are sufficiently close to enable image-space blending like [MGvW98].

We render the ellipsoids using the raycaster presented in [SWBG06]. It computes the ray-ellipsoid intersection in the local coordinate system where the ellipsoid is a sphere. This way we can use the coordinates of the intersection as index to the noise cube map, and simply add the normal noise to the normal vector before transforming it to eye space. To avoid aliasing due to the noise texture, we limit the noise frequency based on the screen-space derivatives of the texture coordinates, just like common mip-mapping. The OpenGL function `fwidth()` provides a ready-to-use upper limit on the mip-map level.

The line renderer is based on [MSE*06], unchanged from the [CP10] implementation.

To improve the visual quality, we apply shadow mapping and precomputed ambient occlusion (AO) (Fig. 9). For the mesh, we compute the AO term per vertex by path tracing random directions in the surface normal hemisphere until the ray leaves the model. For the LoD primitives, we choose random points on the primitive surfaces, rejecting those inside other primitives.

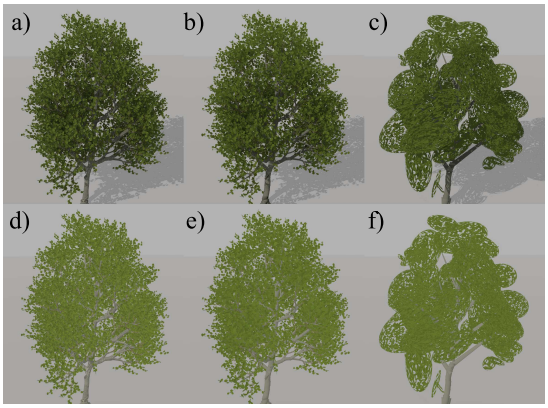

Figure 9: *We use shadow mapping and precomputed ambient occlusion (top) to improve the realism over local illumination only (bottom). Left to right: Mesh, highest LoD, coarser LoD.*

6. Level of Detail Selection

The appropriate level of detail for a given model size in screen space is constrained by two soft limits: When the feature size falls below the limit of the Nyquist frequency, aliasing occurs. In our method, the feature size is determined by the size of the lines and ellipsoids, because the noise texture is already frequency capped. The hard edge of the primitives has a theoretically unlimited frequency, so aliasing cannot be avoided. Without resorting to supersampling, we can only reduce it by reducing the number of edges, which means lowering the level of detail to increase the primitive size. This leads to the second soft limit, the artifacts introduced by large primitives. Fig. 1 shows that a certain number of primitives is required to faithfully represent a model for a given resolution. We use a default average ellipsoid size of 5 pixels in diameter and a default average line width of 1 pixel. This sounds small, but given the uneven distribution of line widths between trunk and twigs, the most visible lines are a few pixels wide.

Based on these sizes, we set the error value of each primitive to the inverse of the target resolution pixels per meter in screen space. This allows us to merge primitives of different sources (lines and ellipsoids of a single model, or multiple models to a group) without additional error normalization steps, and it is compatible to the run-time error selection scheme proposed in [CP10].

7. Results

We implemented out method using parts of the solution presented in [CP10] and the respective free source code available at `http://biosphere3d.org`. To evaluate our new LoD method, we compared it to [CP10] and the billboard

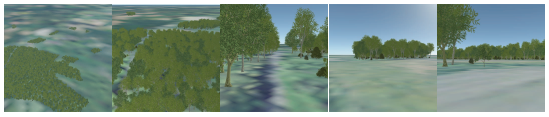

Figure 10: *We measured the performance along a camera path through a scene of 10,000 trees.*

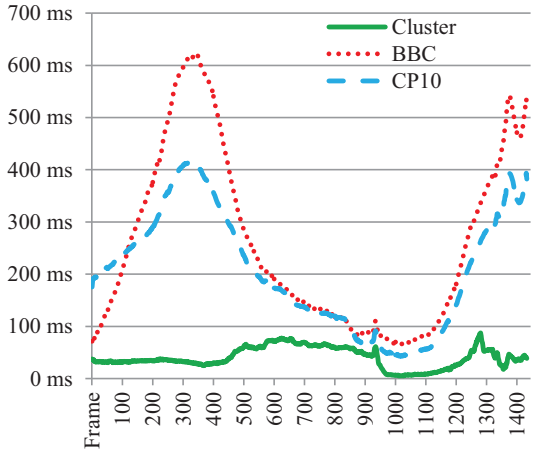

Figure 11: *Time per frame to render the plants at the default resolution for all three methods.*

cloud (BBC) implementation used in [Coc08]. We calibrated the level of detail selection for [CP10] and BBC similar to the method described in section 6, so that the feature size is in the order of a few pixels. Shading was restricted to local illumination to avoid image differences due to the varying lighting terms in the implementations. We created a scene of 10 000 plants in forest groups that could be rendered at interactive frame-rates with all three methods and measured the performance along a camera path of 1440 frames of 1536×864 pixels (Fig. 10) on an Intel Core 2 Duo at 3 GHz and an ATI Radeon HD5870. The path starts with an overview, followed by a descent down to human perspective inside the main forest. Then it heads towards a smaller group of trees and turns back to the main forest.

On each frame, we measured the time for the plant rendering only by taking the difference to a run without trees. On average, our new method took 40 ms per frame, BBC 260 ms, [CP10] 207 ms, and the mesh without LoD 925 ms. The exact frame times are shown in Fig. 11 (we omitted the mesh for improved clarity). Both BBC and [CP10] show peaks when many small trees are rendered—first in the view from above, second in the view from the side. Our new method renders many small trees several times faster.

Since most applications allow reducing the image quality in favor of performance, we measured the same path with

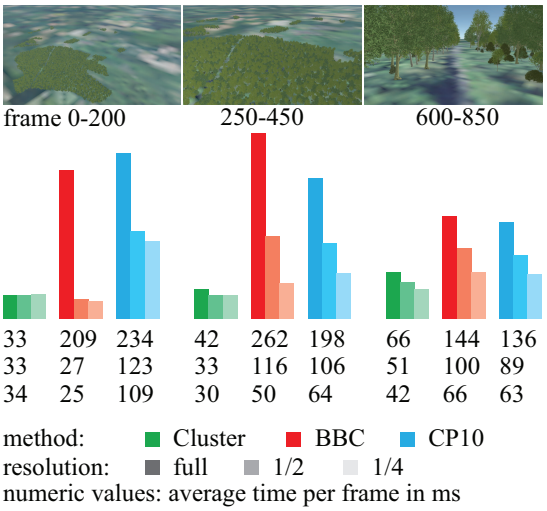

Figure 12: *Performance depends on the camera view and the target resolution for the LoD selection. The rendered image resolution remained constant at* 1536×864 *pixels.*

the LoD selection set to $\frac{1}{2}$ and $\frac{1}{4}$ of the actual image resolution. We chose three distinct frame groups to illustrate the performance behavior (see Fig. 12): In the first 200 frames, all trees cover only a few pixels of the screen. From frame 250 to 450, the trees in the foreground are a few dozen pixels tall. From frame 600 to 850, the camera is inside the forest and trees cover the whole LoD range from mesh in the foreground to a few pixels in the background. Our new method shows the best overall performance, five to six times faster than BBC and [CP10]. At lower resolutions and for distant trees, the billboard clouds marginally outperform it.

We initially configured the LoD selection based on the feature sizes. Based on the captured frames, we also measured the actual RMS image errors compared to a supersampled reference rendering of the source mesh and, additionally, to the next frame in the sequence to evaluate temporal noise (Fig. 13). Temporal noise increased from BBC over our new clustering method to [CP10], while the difference to the reference image decreased in the same order.

We also measured the difference between a single tree model at the highest LoD and the respective source mesh (Fig. 14). This difference causes popping artifacts in the near field. Our new method uses the same high-level representation as [CP10]. Both show only small deviations in the order of single pixels, while BBC shows larger artifacts due to the plane alignments.

To analyze the behavior at low resolutions, we first captured images of all methods at 19 kb memory usage and second at about 130 primitives (Fig. 15). 19 kb corresponds to the lowest BBC resolution with three textured quads using

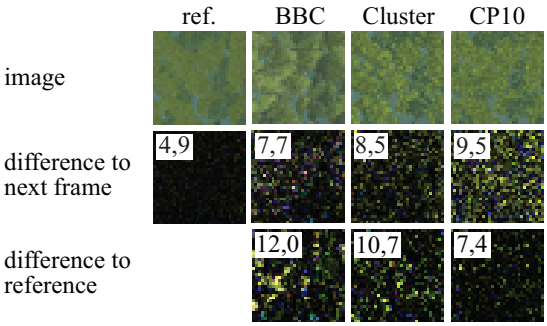

Figure 13: *Our new clustering method has an artifact intensity between BBC and [CP10], where BBC is less accurate compared to the reference image and [CP10] exhibits more temporal noise.*

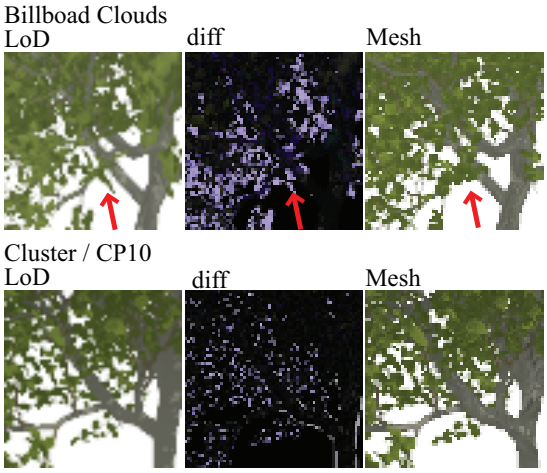

Figure 14: *The difference between the highest LoD and the source mesh is larger for BBC, where the texture planes change the leaf positions. For this LoD, BBC uses the source mesh for the trunk.*

GPU texture compression. Ellipsoids take 140 bytes each, lines 48 bytes, so we took a conservative assumption of 130 primitives for the LoD steps of the new clustering method and [CP10]. To have comparable vertex shader load, we also added a BBC LoD step at 125 primitives, which uses 240 KB. The resulting image quality of the 19 KB clustering is comparable to the 240 KB BBC, while the 19 KB BBC shows considerable artifacts. The 19 KB [CP10] has hardly visible resemblance at the rendered resolution and is only usable at the target resolution of a few pixels.

The precomputation times depend on the number of primitives. On average, our implementation took 0.9 s per source leaf and 0.3 s per source branch to generate the complete

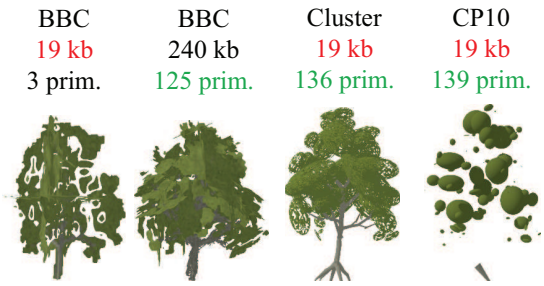

Figure 15: *Comparison of the methods at 19 KB (equal memory usage) and about 130 primitives (equal vertex shader load).*

#primitives	1 k	2.7 k	5 k	10 k	27 k
time[min]	7	12	23	91	396

Table 1: *Precomputation times in minutes by number of source primitives*

LoD hierarchy, resulting in the absolute times shown in table 1. For our models, ellipsoid clustering accounted for 83% of the time.

8. Discussion

In most cases, our method is faster than the previous methods BBC and [CP10]. The single exception is rendering of distant trees at reduced LoD resolution. In this case, BBC has the lower draw call overhead. It uses only a single primitive type, textured quads, and requires only a single draw call for a model, while our methods requires two for ellipsoids and lines. The peak in the BBC results for the highest resolution in the first frame group is caused by the relatively large gap between the lowest LoD used by the two reduced resolutions (essentially a simple cross-billboard) and the next step. So while the lowest BBC LoD is more efficient than the lowest LoD of our method, the latter has a smoother transition, avoiding sudden peaks. This yields a more predictable performance behavior.

The noise analysis shows that there is a trade-off between the difference to the reference frame and the difference to the next frame. BBC has larger deviations from the reference, but less temporal noise, while [CP10] shows the opposite behavior. Our method seems to be well balanced in between. While contrast preservation as described in [CHPR07] happens automatically by averaging the properties of merged primitives, the increased contrast in Fig. 13 visible for both our method and BBC is the result of the lower limit on the primitive size to avoid aliasing. The only practical way around this trade-off is supersampling, because per primitive filters would result in translucent pixels which require a costly depth-sort for proper blending.

Our method is up to several times faster than the state-of-the-art billboard clouds at a comparable image quality. However, as the accompanying video shows, 10 000 trees are still not enough for larger forests. Distant views, where each instance takes only a few pixels on screen, could profit from LoD schemes for groups. Taking a random subset of primitives as proposed in [DCSD02] is fast and easy to implement but prone to aliasing. Our method could be extended by interpreting coarse LoDs of multiple models as a new model and running the simplification process as described. Due to our self-contained primitives, this could be applied to groups of different models. However, in many applications users want to modify scenes. Although the precomputation times are acceptable for static models, this step would have to be accelerated at least by an order of magnitude.

9. Conclusion

We presented a novel LoD method for rendering trees. It combines the rendering efficiency of the ellipsoid and line primitives and the sequential point tree data structure with the accuracy of approximation of fuzzy partitional clustering. As a result, it outperforms previous methods for interactive rendering of forests (40 ms on average for a scene 10 000 trees).

Given the achieved efficiency of rendering instances at low LoD, transforming, culling, and LoD selection are probably becoming the next bottlenecks in scenes with a larger amount of tree instances. Furthermore, we consider extending the LoD hierarchy beyond single instances the most important next step towards realistic landscapes.

References

[BCF*05] BEHRENDT S., COLDITZ C., FRANZKE O., KOPF J., DEUSSEN O.: Realistic real-time rendering of landscapes using billboard clouds. *Comp. Graph. Forum 24*, 3 (2005), 507–516. 2

[BMG06] BOUDON F., MEYER A., GODIN C.: *Survey on Computer Representations of Trees for Realistic and Efficient Rendering*. Tech. Rep. RR-LIRIS-2006-003, LIRIS Lab Lyon, 2006. 2

[CHPR07] COOK R. L., HALSTEAD J., PLANCK M., RYU D.: Stochastic simplification of aggregate detail. *ACM Trans. Graph. 26*, 3 (2007), 79. 7

[Coc08] COCONU L.: *Enhanced Visualization of Landscapes and Environmental Data with Three-Dimensional Sketches*. PhD thesis, Univ. of Konstanz, July 2008. 6

[CP10] CLASEN M., PROHASKA S.: Image-error-based level of detail for landscape visualization. In *Proceedings of VMV - Vision, Modeling & Visualization* (2010). 2, 4, 5, 6, 7

[DCSD02] DEUSSEN O., COLDITZ C., STAMMINGER M., DRETTAKIS G.: Interactive visualization of complex plant ecosystems. In *IEEE Visualization* (2002). 2, 8

[DDSD03] DÉCORET X., DURAND F., SILLION F. X., DORSEY J.: Billboard clouds for extreme model simplification. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)* (2003). 2

[DLR77] DEMPSTER A., LAIRD N., RUBIN D.: Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Metholodigical) 39(1)* (1977), 1–38. 2

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. *ACM Trans. Graph. 22*, 3 (2003), 657–662. 2, 5

[FUM05] FUHRMANN A. L., UMLAUF E., MANTLER S.: Extreme model simplification for forest rendering. In *EG Workshop on Natural Phenomena* (2005). 2

[GMN05] GILET G., MEYER A., NEYRET F.: Point-based rendering of trees. In *EG Workshop on Natural Phenomena* (2005). 2

[KW97] KIM J. ., WOODS J.: Spatio-temporal adaptive 3-d kalman filter for video. *IEEE Transactions on Image Processing Volume 6 Issue 3* (1997), 414–424. 2

[Lan01] LANGE E.: The limits of realism: perceptions of virtual landscapes. *Landscape and Urban Planning 54* (2001), 163–182. 1

[MDMS05] MANTIUK R., DALY S., MYSZKOWSKI K., SEIDEL H.-P.: Predicting visible differences in high dynamic range images - model and its calibration. In *IS&T/SPIE's 17th Annual Symp. Electronic Imaging* (2005), vol. 5666, pp. 204–214. 2, 4

[MGvW98] MULDER J. D., GROEN F. C. A., VAN WIJK J. J.: Pixel masks for screen-door transparency. In *Proc. of VIS '98* (1998), IEEE CS Press, pp. 351–358. 5

[MSE*06] MERHOF D., SONNTAG M., ENDERS F., NIMSKY C., HASTREITER P., GREINER G.: Hybrid visualization for white matter tracts using triangle strips and point sprites. *IEEE TVCG 12*, 5 (2006), 1181–1188. 5

[Paa06] PAAR P.: Landscape visualizations: Applications and requirements of 3d visualization software for environmental planning. *Computers, Environment and Urban Systems 30* (2006), 815–839. 1

[SWBG06] SIGG C., WEYRICH T., BOTSCH M., GROSS M.: Gpu-based ray casting of quadratic surfaces. In *Proc. of EG Symposium on Point-Based Graphics* (2006). 5

[WSB03] WANG Z., SIMONCELLI E. P., BOVIK A. C.: Multiscale structural similarity for image quality assessment. In *IEEE Asilomar Conference on Signals, Systems and Computers* (2003). 4