Image-Error-Based Level of Detail for Landscape Visualization

M. Clasen[1] and S. Prohaska[1]

[1]Zuse Institute Berlin, Germany

Abstract

We present a quasi-continuous level of detail method that is based on an image error metric to minimize the visual error. The method is designed for objects of high geometric complexity such as trees. By successive simplifications, it constructs a level of detail hierarchy of unconnected primitives (ellipsoids, lines) to approximate the input models at increasingly coarser levels. The hierarchy is constructed automatically without manual intervention. When rendering roughly 100k model instances at a low visual error compared to rendering the full resolution model, our method is two times faster than billboard clouds.

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

1. Introduction

Landscape visualization requires rendering a large number of 3d objects such as plants.These are often modeled at high resolution, while coarser representations would suffice for objects covering only a small portion of the screen. Level of Detail (LoD) methods are a way to create such coarser representations and reduce rendering times.

Landscape planners often model their scenes in geographic information systems (GIS), where only an iconographic representation is used for plants if they are displayed at all. Based on this data, the visualization tool is used to display the landscape in 3D, where an interactive view enables the landscape planner to specify camera paths or single views. These are used to create images or videos. The main requirement is the absence of LoD artifacts while minimizing rendering times to enable a quick iterative workflow. Landscape planners usually obtain their plant models from existing collections, such as the Greenworks shop (see http://www.xfrog.com/). These models come at a quite high level of detail which imposes a significant load on interactive systems. Fully automated LoD systems solve this with minimal time and knowledge requirements for the user. To evaluate our method, we chose a scene based on the typical usage scenario of our application, where landscape planners use as many plant instances as possible while balancing on the edge of interactivity. It does not have to be photorealistic, it is sufficient if the LoD is indistinguishable from the source mesh when rendered using OpenGL. When they have to choose between 50 fps with few instances and 5 fps with many instances, they usually accept the lower framerate. The same rules applies to memory usage, where up to a few hundred different models are used per scene.

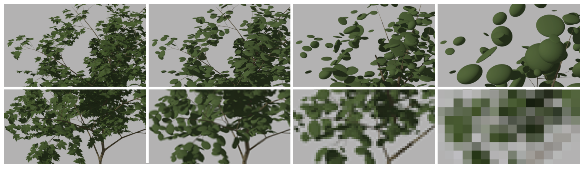

Figure 1: *Original model and three levels of detail. Top: High resolution to show the LoD primitives; bottom: magnification of image at target resolution to show that the LoD primitives are hardly noticeable.*

In this paper, we present a novel LoD method that addresses the following goals: First, it should allow rendering images that closely match images rendered using the full resolution model without imposing restrictions on the input model. Some applications of landscape visualization, such as visibility analysis, require the opacity and coverage to match those of arbitrary full resolution models. Second, image quality at coarse LoD should degrade gracefully and provide a good visual cue of the original model. This can be useful to increase interactivity, for example during navigation, at the cost of image quality. Third, the LoD method should automatically handle a wide range of situations. It should allow a user of landscape visualization to render models without explicitly modeling or controlling the level of details. It should also avoid expensive level or image-based morphings during rendering, because they typically require careful modeling (see Fig. 7 in [SW08]). Fourth, the LoD method should efficiently use graphics hardware. It should, for example, minimize the number of 3D API state changes during rendering to reduce the fixed overhead per model instance.

Our LoD method (Fig. 1) relies on three main contributions. First, we show that an image-based error metric is feasible for both model simplification and LoD selection, even without the connectivity information used in [Lin00] for pre-simplification and local image updates. Second, we show how to accelerate the method by a local error estimate which reduces the number of necessary image comparisons. Third, we show how to leverage unconnected primitive types (ellipsoids, lines) to approximate plant models. In contrast to spheres, ellipsoids can closely match the shape of a tree crown, reducing the required number of primitives for a given image quality. We thoroughly evaluate how to choose the parameters of the algorithm.

After discussing related work in section 2, we describe the LoD construction in section 3. In section 4, we describe how the LoD model is rendered and how to use the image error metric to select an appropriate level of detail. Results are presented in section 5 and discussed in section 6, before we conclude in section 7.

2. Related Work

3D landscape visualization has been a topic for a long time in 3D graphics. The introduction of GPUs on the desktop and the appearance of advanced plant modeling systems, such as [LD98], in the late nineties, accelerated the development of many rendering strategies and LoD schemes for plants. This includes the usage of plant scenes to evaluate generic approaches such as [SD01]. Boudon et al. [BMG06] provide an excellent overview over the different data structures used for plant rendering. They classify representations into detailed (for near-field), global (for large scenes) and multiscale (LoD transition between detailed and global). Multiscale approaches are subdivided into structural and spatial variants, where structural methods group by physical connectivity and spatial methods by distance. Our method uses both: Leaves are treated as independent, whereas branches are simplified based on their connectivity. This hybrid approach is inspired by [DCSD02] and [WP95]. Quadrics surfaces for natural scenes were introduced by [Gar84]. In terms of [MTF03], our method is vegetation specific and focussed on real-time rendering.

The LoD scheme most commonly used for single plant models in interactive rendering is the billboard cloud approach. [FUM05], [BCF*05], and [LEST06] generate billboard clouds from geometric models. They use 1–5 MB texture memory per model. Behrendt et al. [BCF*05] propose a layered surface texture approach for lower levels of detail. They repeat the textures using Wang-tiling to circumvent the texture memory limit. Therefore, this approach is unsuitable for explicit scenes in which the user has control over the position of each instance. At an even higher memory cost (38 MB for a single model at 256^3), the volumetric billboard method by [DN09] provides alias-free rendering of any kind of 3D data, including plants and non-manifold buildings.

Point rendering approaches have similar memory requirements as billboard clouds. Gilet et al. [GMN05] mention 26 MB for a 300k point model, resulting in approximately 100 bytes per primitive. This roughly equals the memory requirements of billboard clouds, where 100 byte can represent a 5×5 pixel sized rgba leaf. Gilet et al. [GMN05], like [DCSD02], also leverage the advantage of point representations by merging multiple plants in regular grid cells. Like [GMN05], we base our point renderer on the continuous level of detail method by Dachsbacher et al. [DVS03]. They upload a whole point hierarchy as a “sequential point tree” (SPT) to the GPU. The main advantage is that a single draw call can render any LoD, and the GPU decides which points to render.

Cook et al. [CHPR07] describe a stochastic approach, similar to [DCSD02], to simplify large sets of similar looking aggregate items such as snow, swarms of insects, or uniform plants. Their approach works fine in this domain, and we use a similar area preservation method. But we strive for a more widely applicable technique that degrades gracefully in performance while ensuring that a chosen image quality is met when applied to unwieldy data.

While most LoD methods, including [GMN05] and [DVS03], use local properties to derive the image error and select the LoD, we base our error metric on the actual image quality. [Lin00] uses a similar approach to simplify meshes, but his approach, in contrast to ours, relies on mesh connectivity to accelerate the error metric updates. Qu et al. [QYN*06] simplify point clouds according to the expected impact to the applied texture, not measuring the actual image errors. Drettakis et al. [DBD*07] choose the LoD based on the resulting image on-the-fly, which provides more flexibility at the cost of performance compared to producing a LoD hierarchy in a preprocessing step.

3. Simplification

To create the quasi-continuous LoD structure, we first convert the source model primitives to the LoD primitives. This is a preprocessing step, only required once for each model. Since this step is primitive type dependent, we introduce it together with the respective primitive (see below). Given the resulting highest LoD, we successively merge two primitives until only a single primitive is left (Alg. 1). The simplification hierarchy is stored in a tree similar to [DVS03]. To choose the next two primitives to be merged, we first randomly gather $N_{new} \cdot N_{local}$ mergeable primitive pairs, called candidates. From these we select the N_{new} with the lowest local error estimate. We measure the image error resulting from the application of this merge step (relative to the source model), and insert candidate into a candidate heap based on [BK02]. Then we choose the candidate with the lowest measured error from the heap. If the measurement is from an earlier simplification step, we measure it again, since the surrounding changes can affect the image error. Updating only the top of the heap can result in a suboptimal choice, but this is negligible compared to the cost of updating the full heap. If the best candidate is found, the two primitives are merged and the candidate is removed from the heap. In a last step, we limit the heap to the best N_{cache} candidates to avoid storing bad and outdated choices while retaining those that might be better than those gathered in the next iteration. We don't use a pre-simplification step as proposed in [Lin00] due to the negative effect on the accuracy of lower levels of detail in this hierarchical scheme.

Algorithm 1: Successive merging

```
repeat
    gather $N_{new} \cdot N_{local}$ new merge candidates;
    select $N_{new}$ with lowest local error estimate;
    foreach new candidate do
        measure the resulting error $e$;
        add to candidate heap;
    end
    repeat
        select candidate with the lowest error;
        if candidate error is outdated then
            measure error $e$ again;
            add to candidate heap;
        else
            apply candidate;
        end
    until candidate is applied;
    prune heap to $N_{cache}$ candidates;
until no candidates are left;
```

3.1. Line

We use lines to approximate plant branches. The branch structure is either given by the modelling system as in [DL97] or can be reconstructed from the source model. We interpret the branches as linear segments with 3D coordinates, radius and surface materials for both start and end

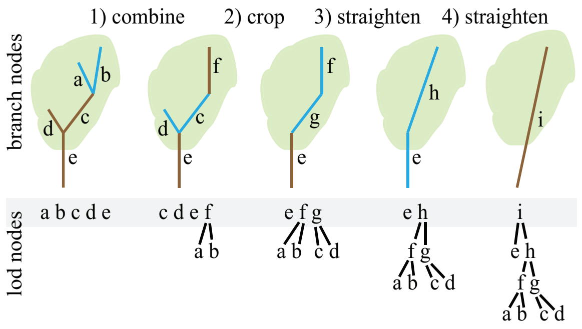

Figure 2: *The source line skeleton is given as a hierarchy of branch nodes which represents the geometric connectivity. Two adjacent nodes can be replaced by a parent LoD node, until only a single branch node is left. This is the root of the LoD hierarchy.*

points. We also retain the branch connectivity. Based on this information, we can define merge steps and successively convert the geometric branch hierarchy to a LoD hierarchy (Fig. 2).

Two consecutive branches can only be straightened if the second branch has no other siblings to avoid losing visual connectivity. The resulting line is build from the start point of the first line and the end point of the second line. Two sibling branches can be combined if both don't have any following branches. Here all properties of the lines are interpolated, weighted with the size of the combines lines. A single branch with no following branches can be cropped. We estimate the local error of these operations based on their basic properties : Straightening and combining have the least visual impact if the angle between the branches is small, while cropping depends on the size of the cropped branch.

3.2. Ellipsoid

We approximate the non-branch elements of plants with ellipsoids. Ellipsoids enable a good approximation of both flat structures (leaves) and voluminous structures (fruits). For coarse LoD, ellipsoids can approximate whole treetops quite accurately. To import the elements, we sample the textured triangles of the source model that belong to the respective element by uniformly distributed points (Fig. 3). For each point, we store the 3D position and material properties. If the alpha value of the texture is below 0.5, the point is discarded. In a second step, we run a principal component analysis (PCA) on the point positions and interpret the eigenvectors as coordinate frame for the ellipsoid. The radii r_i along the coordinate axes is given by the square root of the eigenvalues. In contrast to the lines, there is no connectivity between ellipsoids.

To find a pair of mergeable ellipsoids, we build a Delaunay tetrahedralization based on the ellipsoid centers (Fig. 4).

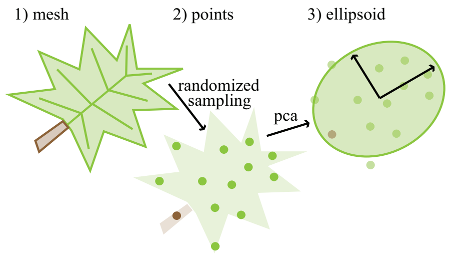

Figure 3: *We import ellipsoids by point-sampling the source triangles of a sub-object (leaf, fruit) and fitting the samples by a PCA.*

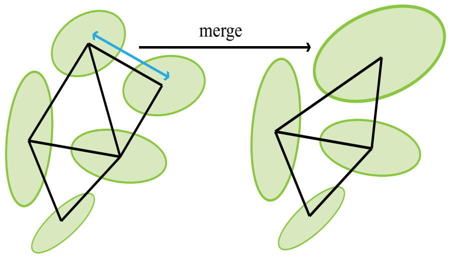

Figure 4: *We organize the source ellipsoids in a Delaunay tetrahedralization. Two adjacent ellipsoids can be replaced by a parent LoD node, until only a single node is left. The spatial neighborhood is not an algorithmic constraint, but a good estimate for the resulting image error.*

Adjacent ellipsoids can either be merged to an average ellipsoid, or one of them can be discarded while enlarging the other one. For the enlarge operation, we first determine the ellipsoid with the larger surface area $a_L > a_S$ based on Knud Thomsen's formula. To compensate the reduced visual surface due to ellipsoid intersection, we approximate it by the surface a_I of intersection of their axis-aligned bounding boxes (a_{bbL}, a_{bbS}) and scale the radii accordingly:

$$r_{i,new} = r_i \sqrt{\frac{a_L + a_S}{a_L} \frac{a_{bbL} + a_{bbS} - a_I}{a_{bbL} + a_{bbS}}}$$

To average two ellipsoids, we align their coordinate frames to minimize the rotation between them (Fig. 5). Then we interpolate the properties of both ellipsoids, using SLERP for the orientation and linear interpolation for all other values, with the interpolation weight

$$t = \frac{a_S}{a_S + a_L}.$$

We also enlarge the resulting ellipsoid as before. We estimate the error of an ellipsoid merge step by the approximated ambient occlusion ao as in [LBD07], the distance d between the center points and the difference s between the surface areas $a_S \cdot s = a_L$: $e = (1 - ao) \cdot d \cdot s$ (Fig. 6).

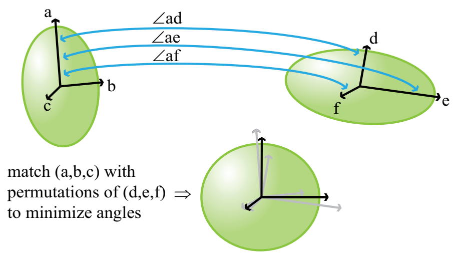
a
b
c
d
e
f
∠ad
∠ae
∠af
match (a,b,c) with permutations of (d,e,f) ⇒ to minimize angles

Figure 5: *To merge two ellipsoids, first we pair their axes to minimize the rotation between the orientations. Then we interpolate orientation and extent.*

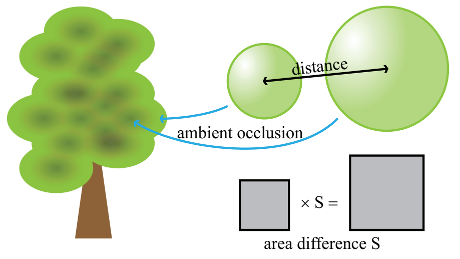

Figure 6: *Higher ambient occlusion, lower distance, and lower area difference result in a lower error estimate for ellipsoids.*

3.3. Calibration

Because the primitive import introduces approximation errors, we scale all primitives of the same type by a constant factor. This factor is determined automatically by comparing the coverage of the imported scaled primitives with the source model and selecting the factor with the lowest image error (Fig. 7).

3.4. Image-based Error Metric

To measure the image error caused by a simplification step, we render multiple views from both the source model and

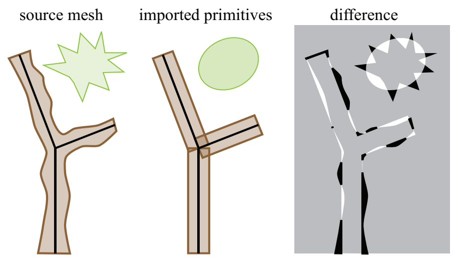

Figure 7: *Since the primitive import is not exact, we compare the coverage of the primitives to the source mesh to adjust the primitive size accordingly.*

the simplification as proposed in [Lin00]. We include the alpha channel to avoid artifacts due to the choice of a background color. The images are then compared using a GPU implementation of the Multiscale Structural Similarity Index (MS-SSIM) by [WSB03].

4. Rendering

For rendering, the LoD tree is transformed to a sequential generalized primitive tree (from coarse to fine) and stored on the GPU as a whole. For each instance a prefix of the tree is processed using the vertex shader to determine which primitives don't match the error criteria and should be omitted. Transformation and rendering follow directly from [DVS03], with the only differences being r_{min} replaced by our node error e, and r replaced by an error threshold e_{max} (see below). For incremental updates during the simplification step, we omit the sorting step and add or replace only the changed nodes (parent and two children) for each merge candidate.

The sequential tree can be prepared in a preprocessing step, so the run-time work is limited to uploading and rendering it on the GPU. We use the ellipsoid renderer presented by [SWBG06], which raycasts an ellipsoid in the fragment shader. Our line renderer is derived from [MSE*06]. Compared to the OpenGL built-in point and line primitives, GPU raycasting results in correct shading. Image space screen-door blending as in [MGvW98] hides the difference between the highest LoD and the original mesh without depth sorting of the individual primitives.

4.1. LoD Selection

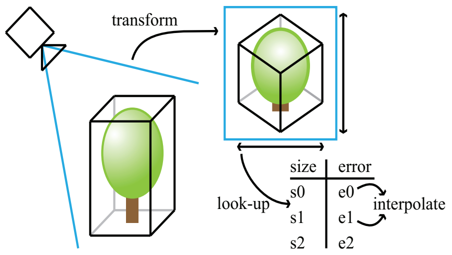

Figure 8: *To estimate the maximal allowed error of an instance, we transform its bounding box to screen space and look up the size in the error table.*

To select the LoD at run-time, we prepare a look-up table in the preprocessing step. First we determine the maximal tolerated image error by computing the error of the highest LoD for an image resolution where the smallest features cover only a few pixels. Since decreasing the resolution hides smaller errors, a lower LoD is sufficient. For each $2^{-i}; i > 0$ scale of the initially chosen resolution, we determine the lowest LoD that still meets the error threshold.

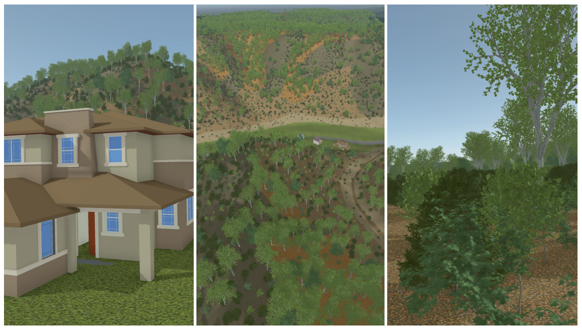

Figure 9: *Three views from the scene that we used to evaluate the LoD methods (≈ 90,000 instances of 24 models).*

At run-time, we compute the screen-space resolution of the bounding box of an instance and use this size to interpolate the necessary LoD from the table (Fig. 8). If the target resolution is higher than defined in the look-up table, we switch to the source model for high-quality close-ups.

5. Results

Based on the requirements of landscape planning (see Introduction), we designed a scene which is navigatable at slightly reduced quality and renders at about 1 fps at full quality, matching the frame rates of Fig. 8 in [SD01]. An area of of $\approx 3km^2$ is covered with $\approx 90,000$ instances of 10 plants (various Xfrog-modelled Abies, Acer, Taxus, and Tilia) (Fig. 9). The plant models' complexity varied between 999 and 27,390 lines and ellipsoids. The usage of only 10 models has no effects on performance since the model switching overhead for > 100 instances per model is negligible.

We defined a camera path through this scene including overviews and close-ups. Along this path (see supplementary material), we rendered frames at 1280×720 using the source model, our LoD, billboard clouds (BBC) with the implementation from [Coc08], and sequential point trees (SPT). Note that even though Fig. 8 in [DVS03] shows plant models, SPT was originally designed for 2-manifold point-based surfaces. All three implementations use a hybrid approach and render the source model primitives where necessary and are, therefore, able to meet any image quality limit.

5.1. Performance

Following our goal of exact visual results, we set all methods to a hardly noticeable difference for the full quality comparison. We measured the difference using the HDR-VDP implementation of [MDMS05] and calibrated each LoD method to produce approximately the same image error over the camera path. Then we measured the time to render the models without terrain and deferred shading on a Intel Core 2 Duo at

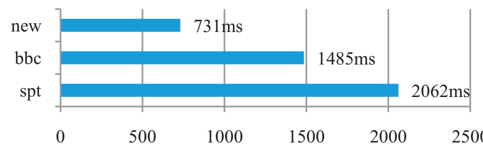

Figure 10: *Average time to render the camera view of the objects in the scene depicted in Fig. 9 at* 1280×720*: Our new method, Billboard Clouds (bbc) and Sequential Point Trees (spt).*

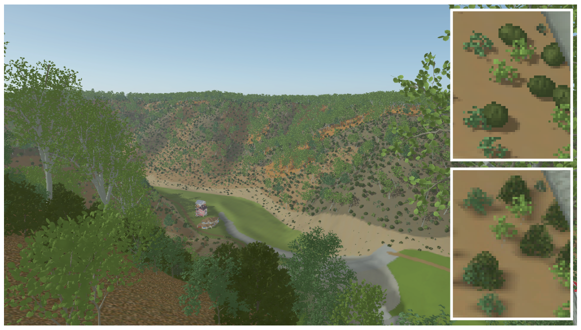

Figure 11: *Interactive view at reduced LoD. Inset compares reduced LoD (top) with standard LoD (bottom).*

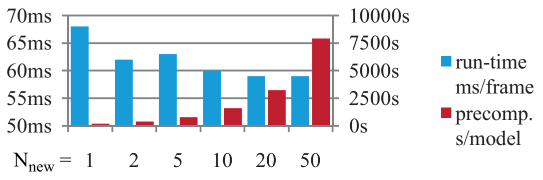

Figure 12: *Increasing* N_{new} *proportionally increased the precomputation time, while the run-time performance improved only marginally for* $N_{new} > 2$*.*

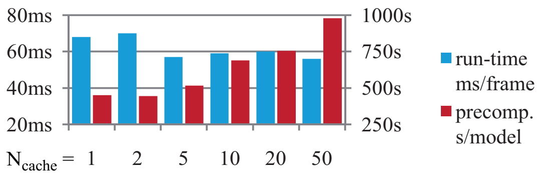

Figure 13: *Increasing* N_{cache} *increased the precomputation time, with no run-time benefit for* $N_{cache} > 5$*.*

2.13 Ghz using a Radeon HD4850 under OpenGL (Fig. 10). Rendering one frame with our proposed LoD took 731ms on average, which was faster than BBC by a factor of two (1485ms) and SPT by almost three (2062ms) (source mesh without LoD: 11s). Rendering the plants at a reduced level of detail (Fig. 11) took 194ms and matches our target of 5 fps. As shown in the inset, the small Taxus models were rendered as single ellipsoids.

5.2. Memory Usage

Our implementation uses 140 bytes per ellipsoid and 48 bytes per line.With the native 1:2 branching in the LoD hierarchy, we have $2n$ LoD primitives for a model with n primitives on the finest level. Skipping every other level leads to 1:4 branching as in [DVS03], reducing the number to $\frac{4}{3}n$. However, we observed no effect on the rendering performance.

5.3. Precomputation Performance

The precomputation parameters affect both the precomputation and the run-time performance. To find a good compromise between these, we measured both for an Acer model with 4240 primitives, varying each parameter separately and choosing the other parameters such that they do not hide the effect of the varied parameter. Increasing N_{new} (at $N_{local} = 1$, $N_{cache} = 1$, view size $v = 256$, number of views $\#v = 8$) results in proportionally higher precomputation times (Fig. 12). However, we observed only small effects on run-time performance for $N_{new} > 2$. The cost of an increased N_{cache} (at $N_{local} = 1$, $N_{new} = 2$ to fill the cache, $v = 256$, $\#v = 8$) is smaller, since not all cached candidates have to be evaluated in every step (Fig. 13). We observed a higher run-time performance up to $N_{cache} = 5$. The precomputation cost of local error estimates is relatively small (Fig. 14). We observed no effect on run-time performance for $N_{local} > 2$ (at $N_{new} = 1$, $N_{cache} = 1$, $v = 256$, $\#v = 8$).

The view size for the image comparison has the largest effect on run-time performance (Fig. 15). We measured a constant performance increment up to 256^2 pixels (at $N_{new} = 5$, $N_{cache} = 5$, $N_{local} = 5$, $\#v = 8$). Our SSIM implementation is limited to this value due to high GPU memory usage (in addition to the buffers required for rendering, 512^2 standalone) . Increasing the view size had no effect on the precomputation times. We further analyzed the effect of the view size and found that the run-time performance is connected to the size of the smallest primitives. When they fall below pixel resolution, the image error metric cannot evaluate them properly, resulting in sub-optimal LoD. The number of views affects the view-independence and, therefore, run-time performance. We observed improvements up to 4 views (at $N_{new} = 5$, $N_{cache} = 5$, $N_{local} = 5$, $v = 256$) (Fig. 16). There was no effect on precomputation cost. Given these results, we chose $N_{new} = 2$, $N_{local} = 2$, $N_{cache} = 5$, a view size of 256^2 and 4 views and measured the total precomputation times (Tab. 1).

Root mean square is fast and simple, but not robust in terms of perceived visual quality. When comparing the

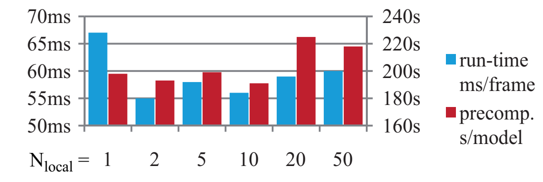

Figure 14: *Increasing the number of local estimates has only a small effect on the precomputation times. There is no run-time benefit for $N_{local} > 2$.*

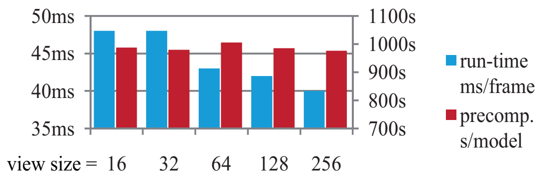

Figure 15: *Increasing the image error view size improves the run-time performance with no overhead during precomputation.*

HDR-VDP implementation of [MDMS05] to MS-SSIM by [WSB03], we found the latter to be robust enough while performing better. With 232 comparisons per second at 512^2 on a Geforce 8800GT, it is even faster than [SS07].

6. Discussion

Our results show that the proposed method performs well compared to both the billboard cloud method and to the sequential point tree. The precomputation costs are higher. But this step is fully automated and only required once per model, so our method is suitable for landscape visualization with static models. It is not suitable for dynamic models and cannot exploit the connectivity information of 2-manifolds. The widely used Speed-Tree library cannot be directly compared to our approach. Speed-Tree uses optimized level of details created by human modelers and, thus, cannot be readily applied to third party models that are only available as high-resolution models. Also, in most games, the high visual quality in the midground is counterbalanced by sparse forests in the background, which landscape planners are unlikely to accept.

The limited view size due to the SSIM memory usage could be circumvented by a tiled SSIM implementation. Alternatively, an optimized usage of temporary result buffers and an increase in available GPU memory might alleviate this issue at lower cost.

It is advisable to use incremental updates for the model during simplification. An unsorted SPT prevents prefix rendering, but the overhead of the additional rendered primitives is negligible compared to the update of the whole model. Our method does not require a separate contrast

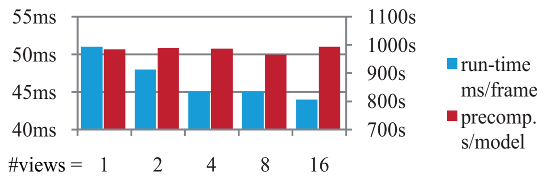

Figure 16: *Computing more than four image error views shows no benefit on run-time (no effect on precomp. times).*

primitives	1k	2.8k	5.3k	11k	27k
sorted	78 s	271 s	644 s	2268 s	10317 s
unsorted	45 s	111 s	245 s	496 s	1686 s

Table 1: *Precomputation times by number of primitives for the standard sorted SPT and our unsorted implementation*

preservation as [CHPR07]. We support both averaging and discard steps, and the image error metric automatically guarantees that the constrast error introduced by discard steps is compensated by averaging steps. Due to the faithful approximation of the source leaves by the ellipsoids, there is no inherent contrast error between source mesh and lod. However, aliasing can emphasize differences between textured triangles and raycasted primitives (see bonus material).

Using volumetric primitives such as ellipsoids enables a LoD hierarchy where even the coarsest level with a single primitive is useful, if the simplification scheme respects the overall shape of the model. We ensure this using a multi-scale error metric. This allows building a next hierarchy on top using similar methods.

7. Conclusion

We presented a LoD method based an image-error metric driven simplification of unconnected primitives. It has several advantages compared to previous methods. It works on non-manifold input data.The image quality is guaranteed to match a specified limit. The appearance scales well to lower LoD, up to the coarsest level where a single ellipsoids represents the general shape of a tree. By leveraging the quasi-continuous SPT data structure, we can render many instances at different LoD with minimal CPU overhead. Using advanced primitives such as 3D lines and ellipsoids, less primitives are required for a given image quality when compared with simple splats. The net result is a method which outperforms existing fully automated solutions when high image quality is required.

8. Acknowledgements

We would like to thank Lenné3D GmbH for the plant models, Timm Dapper for the plant skeleton extractor and Liviu Coconu for the billboard cloud implementation.

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

[BK02] BISCHOFF S., KOBBELT L.: Ellipsoid decomposition of 3d-models. In *3D Data Processing Visualization and Transmission* (2002), pp. 480– 488. 3

[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. 2, 7

[Coc08] COCONU L.: *Enhanced Visualization of Landscapes and Environmental Data with Three-Dimensional Sketches*. PhD thesis, Univ. of Konstanz, July 2008. 5

[DBD*07] DRETTAKIS G., BONNEEL N., DACHSBACHER C., LEFEBVRE S., SCHWARZ M., VIAUD-DELMON I.: An interactive perceptual rendering pipeline using contrast and spatial masking. In *Proc. Eurographics Symp. Rendering* (2007). 2

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

[DL97] DEUSSEN O., LINTERMANN B.: A modelling method and user interface for creating plants. In *In Proceedings of Graphics Interface 97* (1997), Morgan Kaufmann Publishers, pp. 189–197. 3

[DN09] DECAUDIN P., NEYRET F.: Volumetric billboards. *Comp. Graph. Forum 28* (2009). 2

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. *ACM Trans. Graph. 22*, 3 (2003), 657–662. 2, 3, 5, 6

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

[Gar84] GARDNER G. Y.: Simulation of natural scenes using textured quadric surfaces. In *Siggraph Comput. Graph. 18, 3* (1984), pp. 11–20. 2

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

[LBD07] LUFT T., BALZER M., DEUSSEN O.: Expressive illumination of foliage based on implicit surfaces. In *EG Workshop on Natural Phenomena* (2007). 4

[LD98] LINTERMANN B., DEUSSEN O.: A modelling method and user interface for creating plants. *Comp. Graph. Forum 17*, 1 (1998), 73–82. 2

[LEST06] LACEWELL J. D., EDWARDS D., SHIRLEY P., THOMPSON W. B.: Stochastic billboard clouds for interactive foliage rendering. *Journal of Graph. Tools 11*, 1 (2006), 1–12. 2

[Lin00] LINDSTROM P.: *Model simplification using image and geometry-based metrics*. PhD thesis, Atlanta, GA, USA, 2000. Adviser-Turk,, Greg. 2, 3, 5

[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. 5, 7

[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

[MTF03] MANTLER S., TAYLOR R. F., FUHRMANN A. L.: The state of the art in realtime rendering of vegetation. VRVis Center for Virtual Reality and Visualization, July 2003. 2

[QYN*06] QU L., YUAN X., NGUYEN M. X., MEYER G. W., CHEN B., WINDSHEIMER J.: Perceptually guided rendering of textured point-based models. In *Eurographics Symposium on Point-Based Graphics* (2006). 2

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (London, UK, 2001), Springer-Verlag, pp. 151–162. 2, 5

[SS07] SCHWARZ M., STAMMINGER M.: Fast perception-based color image difference estimation. In *ACM Siggraph Symposium on Interactive 3D Graphics and Games Poster* (2007). 7

[SW08] SCHERZER D., WIMMER M.: Frame sequential interpolation for discrete level-of-detail rendering. *Comp. Graph. Forum (Proceedings EGSR 2008) 27*, 4 (June 2008), 1175–1181. 2

[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

[WP95] WEBER J., PENN J.: Creation and rendering of realistic trees. In *Proc. SIGGRAPH '95* (New York, NY, USA, 1995), ACM, pp. 119–128. 2

[WSB03] WANG Z., SIMONCELLI E. P., BOVIK A. C.: Multi-scale structural similarity for image quality assessment. In *IEEE Asilomar Conference on Signals, Systems and Computers* (2003). 5, 7