7. International Immersive Projection Technologies Workshop
9. Eurographics Workshop on Virtual Environments (2003)
J. Deisinger, A. Kunz (Editors)

Fast Approximate Visible Set Determination for Point Sample Clouds

Stephan Mantler and Anton L. Fuhrmann

VRVis Center for Virtual Reality and Visualization
(mantler|fuhrmann)@vrvis.at

Abstract

We present a fast, efficient method to determine approximate visible sets for vegetation rendered as point sample clouds. A hardware accelerated preprocessing step is used to determine exact visibility for a selected set of views; at runtime the current view is rendered using an approximate visible set constructed from the three closest precalculated views. We will further demonstrate how this method leads to a significant per-frame reduction of the original data size.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Visibility Determination

1. Introduction

When rendering large numbers of trees for terrain flyover or realtime GIS visualizations, a tree's leaf is typically smaller than a single pixel. Therefore, rendering these leaves as individual quadrilaterals (maybe even textured) is wasteful. In this case, clustering techniques (groups of leaves are approximated by a single polygon) or image based techniques with a similar strategy are usually employed[2, 4, 3]. Point based and hybrid techniques have been also explored[8, 1]. We will follow the point based approach, as it allows for an almost seamless transitions to higher levels of detail. The representation of foliage as a cloud of point samples implies that the proposed method can also be applied to other data of similar nature (such as laser range data, for example).

Since point sample clouds are a very large number of independent and disjoint samples, traditional occlusion based acceleration methods such as backface culling or spatial schemes can not be applied easily. Still, the large number of samples necessitates some means of simplification in order to achieve realtime rendering performance.

Our proposed algorithm uses a preprocessing step to determine the exact visible set for a number of views; at runtime the visible sets of the three closest matches to the actual view are combined for rendering. Point budgets can be assigned, so that distant objects are rendered with less detail.

1.1. Related Work

The rendering of plants has been quite thoroughly investigated in recent years, and a number of innovative algorithms have been published for rendering realistic plants in realtime. IDV, Inc. has developed a commercial product that renders groups of leaves as a single polygon for efficiency [2]. Meyer et al. render trees with hierarchical image based rendering and bidirectional texture functions. Their method also supports shading and shadowing[4]. Deussen et al. present a hybrid polygonal and point based technique for rendering various levels of detail of plants in realtime[1]. Their system supports importance reduction, which renders visually important parts of a scene at a higher quality than the rest.

In their landmark paper, Weber and Penn propose a hybrid solution for rendering their highly detailed trees. Their approach to levels of detail is not the generation of multiple distinct models, but reinterpretation of a particular model, by rendering leaves either as textured polygons or point samples. Also, branches can be rendered either polygonally or as simple lines[8].

Apart from the system proposed by Meyer et al.[4], none of these systems use view direction based data reduction methods, and rely on orientation independent methods for LOD generation. Meyer et al. sample view directions to generate a hierarchy of bidirection texture functions, and also include shading and shadowing in their image based rendering system.

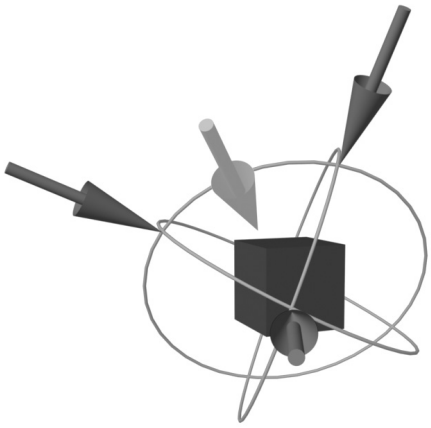

Figure 1: *Identifying closest available view directions. For an arbitrary viewpoint (light grey arrow), the three closest precalculated directions (dark arrows) are identified and rendered.*

Jakulin's Slicing and Blending algorithm[3] uses a view direction based method to blend between different views of a tree model. However, his approach uses a small number of parallel "slices" (textured quadrilaterals) to represent foliage.

The current view direction is also used in point based rendering systems for solid models, such as Qsplat[5], for backface culling and splat size estimation. When approximating leaves through individual point samples, there is no continuous surface or solid, making backface culling inadequate.

Another method for rendering point based data is to use a randomized approach[6, 7]. In this case, a (possibly view dependent) heuristic selects arbitrary samples from the data set. Special care must be taken to make sure that holes and artifacts do not occur, for example by using a poisson disc distribution function.

2. Preprocessing

Visible set determination can be performed very efficiently in hardware. First, a number of views is generated by selecting points on the unit sphere. Since trees cannot usually be viewed from directly underneath, this can also be constrained to a hemisphere. Our implementation employs a simple tetrahedron subdivision scheme to generate view directions.

The entire data set is then rendered from each view, but instead of using the original color information each leaf is rendered with a unique color. Individual leaves are rendered as single point primitives. To avoid unwanted artifacts, the data set is rendered without antialiasing, attenuation, or lighting. The frame buffer is then read back and each pixel's value is used to identify the visible sample at this point.

For graphics hardware that supports occlusion querying, reading back the frame buffer can be avoided: The data set is first rendered in full as before. It is then re-rendered in chunks of n samples with occlusion query enabled, and the depth test set to 'less or equal'. If no pixels have been updated, the entire chunk was invisible and can be discarded. Otherwise it is subdivided and processed recursively to identify all visible samples.

2.1. Sample Identification

To be correctly identified after the frame buffer has been read back in, each sample needs to be assigned an unique color. For simplicity, we use the sample's array index in the data set. Of course, 8-bit RGB colors effectively limit the maximum number of samples that can be identified in a single pass to $2^{24} - 1$.

However, this limit can be easily avoided by using multiple partitions of $2^{24} - 2$, reserving the value $2^{24} - 1$ for the background (no sample) and 0 for any samples not within the current partition. The preprocessing step then needs to render the entire set more than once, until all samples have been covered.

2.2. LOD Estimation

By rendering samples with more than one pixel and by re-rendering the same view from varying distances, a level of detail estimation can be obtained as follows: For each pixel that has been covered by a certain id, the contribution of the corresponding sample is increased by one. After all distances have been covered, samples with zero contribution are discarded, and the remaining samples are sorted by descending contribution. This way, leaves which are visible from all distances end up with the highest contribution count and will be rendered even if the the total number of points to be rendered exceeds the allocated budget.

2.3. Alternative LOD Generation

Even though it is tempting to assume that samples which are visible at larger distances are also visible at a closer range, this is not always correct for perspective projection. Such samples may be obscured if perspective views are rendered from varying distances, rather than at different scales. Figure 2 illustrates this difference. However, a similar error is incurred from approximating the view direction; there may be samples that should be visible from the actual view point but were not detected in any of the three views.

2.4. Stems and Branches

If desired, stems and branches can be included in this system. This is particularly useful if the tree has been generated

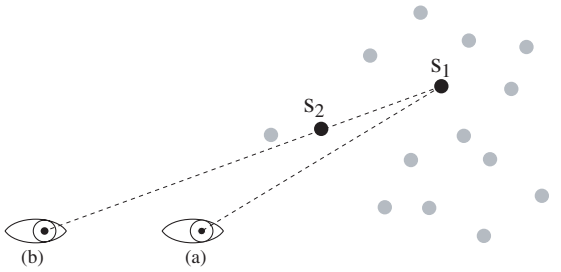

Figure 2: *Perspective Occlusion. Splat S1, although visible from viewpoint (a), is occluded by splat S2 when viewed from viewpoint (b) as it is seen at a slightly different angle.*

automatically and includes a great number of small twigs. We have implemented this by assigning a separate range of identifier values to polygons; the sample identification process only requires minor changes to accommodate this. Although the resulting polygonal data does not lend itself very well to triangle striping or similar acceleration schemes, it greatly reduces the data to be rendered while still maintaining good visual quality.

2.5. Alternative LOD Method

As an alternative to sorting samples by contribution alone, they can be arranged by view distance first. The advantage of sorting samples by decreasing distance is precise control over how many points need to be rendered: an upper bound is the splat count for the next (closer) view (see figure 3). Interpolation could be used for smooth transition between view distance steps, although ideally this should not be necessary, as the closer view does already provide coverage for all visible samples. We have therefore chosen not to implement smooth transitions as they are already quite unobtrusive.

A straightforward implementation of this method would require storing additional information for each sample, or searching through the previously determined (greater) distances to determine if this sample has been seen already (obviously, each sample should be rendered only once). However, this time consuming step can be avoided:

Each view direction is rendered from various distances, beginning with the greatest distance. Samples that have already been visible at an earlier iteration are not assigned their original id, but simply rendered with the reserved background id. They will therefore still correctly obscure other samples, but remain invisible to the following sample identification process.

3. Rendering

During rendering, the three closest view directions are determined for each object. This can be done naively by finding the dot product of the actual view vector and the vector stored for each view direction. A more efficient method would be to precompute the three closest views in a cube map or similar lookup table.

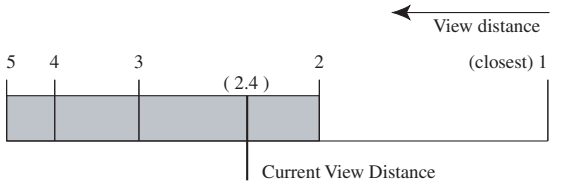

Figure 3: *Sorting point samples by distance. For the current view distance, all point samples up to and including view distance '2' would be rendered.*

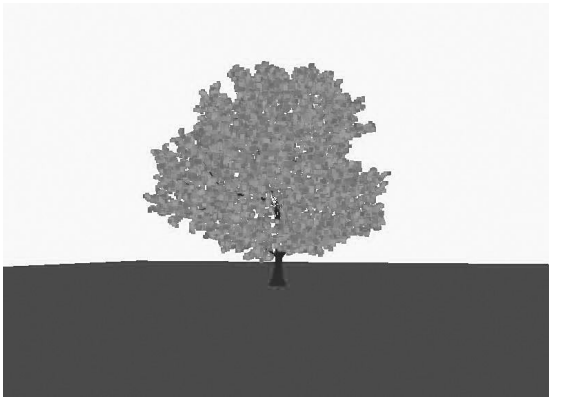

Figure 4: *Rendering of the reduced "Oak" model from an arbitrary view point.*

Next, the number of samples to be rendered is found as a function of object distance and the total number of samples for this view direction. The samples can then be rendered efficiently as a single vertex array. For the original algorithm, since samples are ordered by contribution more important (ie. highly visible) data is guaranteed to be rendered even if only few samples are displayed. The alternative method always renders a sufficient number of point samples through the use of a distance based lookup table for the appropriate sample count.

3.1. Rendering Multiple Instances

If a model is reused several times, for example to produce a group of trees, a slight variation in color and texture will greatly enhance the visual quality. However, this is not easily possible since sample data - including color information - is stored in vertex array for efficient rendering. For color variation, this array would either have to be copied and modified for each instance, or walked through "by hand" and each sample rendered individually. In both approaches, the advantage of using a vertex array is lost.

We circumvent this problem by reusing the same vertex array, including color information. The variation of appearance is achieved by adjusting light source color and intensity instead, which can be regarded as adding an individual per-instance bias (see Figure 12).

4. Results

We have implemented preprocessing and rendering of the leaves of various models of trees; the tree models were generated through an algorithm similar to the one proposed by Weber and Penn[8]. Lighting was precalculated by ray casting through a regular volume grid; the density of each cell was estimated through a heuristic function based on the number of enclosed leaves. The stems, although certainly necessary for a realistic impression, have been omitted for the prototype implementation described in this abstract, but should be available for the final paper.

Figure 5 displays the preprocessing results for one particular tree model. 26 views were generated through a tetraeder subdivision scheme, and the total and relative count of visible samples found with the original algorithm. This model only contains a dense set of leaves and no polygonal stems or branches. Loading the original data set (10 MB) and preprocessing took about 35 seconds on an 1.4GHz Intel Pentium 4 with GeForce4 graphics. The total number of samples for all views is 86265, each consisting of a vector and color information, for 15 bytes/sample and 1.3MB total per tree model. Data structure overhead is about 20 bytes per view direction. If polygonal data is stored as well, it can be estimated with an additional 45 bytes/triangle.

For comparison, Meyer et al. report "a few tens of Megabytes" for their method, and a preprocessing time of about 75 minutes using an Onyx2 Infinite Reality.

In Figure 6, a complete tree model has been preprocessed with the alternative, distance-sorting algorithm. Each column represents a particular view direction, and the individual view distances are stacked from distant (bottom) to close-up (top). The upper graph represents triangular data (ie. stem and branches), and the lower graph displays point samples (ie. leaves). In this case, the total time for I/O and preprocessing was about three minutes. The total data file size is about 12MB.

The model, an oak-like tree, has a much less dense leaf cover than the "Balsam" model, and therefore also a higher percentage of visible leaves in comparison. A number of things can be seen in these graphs: For some view directions, there are distance steps that do not reveal any new samples. This may be in part due to the use of the OpenGL Point Parameter extension to adjust the point size according to distance.

Figure 4 shows the oak model rendered from an arbitrary viewpoint. Three out of 70 view directions have been rendered, resulting in a total of 22897 points (25.15%) and 763 polygons (1.47%). Assuming that triangles are three times as expensive as points to send to the graphics pipeline, the relative bandwidth usage can therefore be estimated at 10.19%. Figure 7 tracks this value over time as the viewpoint is moved about the model. As can be seen, it drops significantly - to less than 0.3% - as the view distance increases.

Due to the nature of our algorithm, overdraw can be almost completely avoided while still providing guaranteed bounds on visual fidelity. This mostly depending on how many view samples have been generated, as well as any other parameters such as point size. Consequentially, these cannot be changed dynamically and need to be chosen carefully at the preprocessing step.

Also, since our implementation only uses the view direction and position for visibility determination, the camera's field of view is one of these fixed parameters. While this may be an acceptable solution for most cases, a more flexible solution may be desirable to achieve better visual quality if these parameters do need to be changed dynamically.

view	visible	pct.	view	visible	pct.
1	2617	1.02%	2	1650	0.64%
3	4495	1.76%	4	2428	0.95%
5	4360	1.70%	6	3809	1.49%
7	2465	0.96%	8	1238	0.48%
9	3983	1.56%	10	2321	0.91%
11	2751	1.07%	12	3763	1.47%
13	2751	1.07%	14	2429	0.95%
15	4113	1.61%	16	3838	1.50%
17	3980	1.55%	18	2135	0.83%
19	3546	1.39%	20	3687	1.44%
21	3974	1.55%	22	4295	1.68%
23	3783	1.48%	24	4114	1.61%
25	4315	1.69%	26	3425	1.34%
		total	26	86265	33.69%

Figure 5: *Preprocessing results for tree "Balsam" (total points 256024, splat size 4 pixels, viewport* $600 * 600$*).*

5. Conclusion and Future Work

We have presented a new algorithm for view direction based data reduction of disjoint point sample clouds. Our method is fast (using commonly available hardware acceleration), memory efficient, and easily adaptable to other data with similar properties. Preliminary results have been presented, illustrating a large savings in memory, preprocessing and rendering load.

The benefit of this method is that it can handle objects and static object groups where traditional occlusion based algorithms fail due to high complecity.

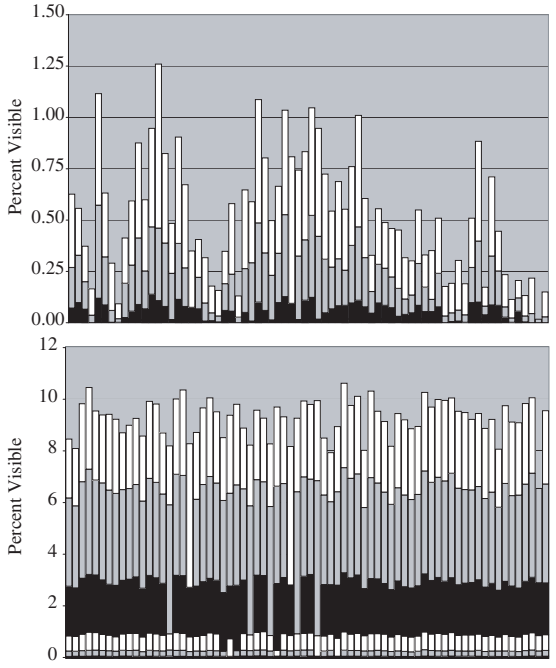

Figure 6: *Visible branches (ie. triangles, top) and leaves (point samples, bottom) for tree "Oak" (total points 91054, triangles 52076).*

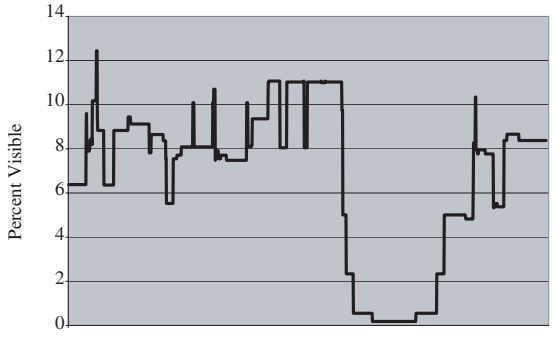

Figure 7: *Percentage of total bandwidth used over time for a number of views.*

5.1. Future Work

Obviously, this algorithm is not inherently restricted to point sample clouds and should be easily generalized to other data such as disjoint polygons or entire objects (which can be approximated as bounding spheres or bounding boxes for speed).

As an immediate improvement, the polygonal stems could also be identified by the same process, leading to a fully usable tree rendering model. To improve near field rendering quality, the preprocessing step could be made with polygonal leaves. By counting the pixels covered by each leaf, there would be an immediate metric for selecting the appropriate level of detail for each leaf (more than a few pixels: polygonal; only few pixels: point; zero: discard). Each view would then consist of several arrays: a stem list, a polygonal leaves list, and the original point sample list.

5.2. Acknowledgements

Most of this work has been done at the VRVis research center, Vienna, Austria (`http://www.vrvis.at/`), which is partly funded by the Austrian government research program Kplus. The authors would also like to thank Andreas Reichinger for his tree model generation software.

References

1. Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. In *IEEE Visualization '02*, October 2002.

2. Interactive Data Visualization, Inc. Speedtree product homepage. web page, 2002. http://www.idvinc.com/speedtree/.

3. Aleks Jakulin. Interactive vegetation rendering with slicing and blending. Eurographics Conference Short Paper, 2000. http://zeus.fri.uni-lj.si/ aleks/slicing-and-blending/.

4. Alexandre Meyer, Fabrice Neyret, and Pierre Poulin. Interactive rendering of trees with shading and shadowing. In *Workshop on Rendering*, Eurographics. Springer-Verlag Wien New York, July 2001.

5. Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

6. Gevorg Grigoryan University. Probabilistic surfaces: Point based primitives to show surface uncertainty.

7. M. Wand, M. Fischer, and F. Meyer. Randomized point sampling for output-sensitive rendering of complex dynamic scenes, 2000.

8. Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 119–128. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

Figure 8: *Reduced "balsam" model rendered from an arbitrary viewpoint. Rendered with 9017 point samples (3.5% of the original size).*

Figure 9: *Full "balsam" model, rendered from the same viewpoint as Figure 8 for direct comparison (256024 point samples).*

Figure 10: *Difference image between original and reduced "balsam" models. Red pixels identify places where a different splat would have been rendered in front by the full model.*

Figure 11: *Per-pixel difference image between original and reduced "balsam" models. Red pixels indicate the difference between pixel values of the reduced and full models.*

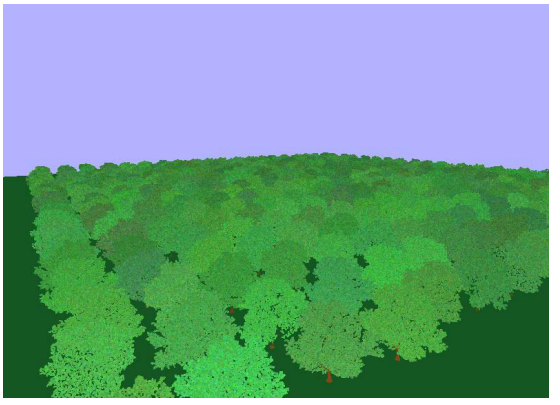

Figure 12: *400 instances of the "Oak" tree model, with individual orientation and color bias, rendered at 15-20Hz on a GeForce4.*