Vision, Modeling, and Visualization (2015)
D. Bommes, T. Ritschel, and T. Schultz (Eds.)

Level-of-Detail for Production-Scale Path Tracing

M.Prus[1] and C.Eisenacher[2] and M.Stamminger[1]

[1]University of Erlangen-Nuremberg, Germany
[2]Walt Disney Animation Studios

Abstract

Path-traced global illumination (GI) becomes increasingly important in movie production. With offscreen elements considerably contributing to the path traced image, geometric complexity increases drastically, requiring geometric instancing or a variety of manually created and baked LOD. To reduce artists' work load and bridge the gap between mesh-based LOD (Mip-maps) and voxel-based LOD (brickmaps), we propose to use an SVO with averaged BRDF parameters, e.g. for the Disney-BRDF, and a normal distribution per voxel. During shading we construct a BRDF from the averaged BRDF parameters and evaluate it with a random normal sampled from the distribution. This is simple, memory-efficient, and handles a wide variety of geometry scales and materials seamlessly, with proper filtering. Further it is efficient to construct, which allows quick artist iterations as well as automatic and lazy generation on scene loading.

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

1. Introduction

As production rendering is moving towards physically based GI, existing ray tracing approaches have to be able to handle heavy production scenes in reasonable time. Existing renderers focus on ray sorting, which allows for coherent ray traversal and shading [ENSB13] increasing the overall rendering performance for in-core scenes. However, in a production environment, the need for highly detailed geometry in large-scale scenes grows fast and many of the current scenes exceed main memory forcing out-of-core processing. Memory requirements become a serious issue for efficient rendering as in [YLM06] accurately described. Solutions such as geometric instancing are neccessary and currently widely used. However, geometric instancing exposes difficulties as it leads to undesirable repetitions. Geometric simplification with according Mip-Map textures is possible, but usually involves artist intervention. Alternative voxel based approaches, e.g. Brickmaps [CB04], involve difficult data setups and long baking processes, which hinders quick artist iterations. Further, managing those data structures has shown difficult in a production environment where assets change frequently and prior bakes become quickly outdated.

In this paper, we propose a simple LOD approach. First, during scene loading, appropriate LOD candidates are selected based on a view-dependent criteria. Then, we generate our compact LOD data structure from these candidates and use this for path tracing, while discarding the underlying geometry. This allows us to reduce the memory footprint and increase the overall performance due to less out-of-core scenarios for large-scale environments. Further, we achieve good quality by approximating the geometry within a voxel with tight slabs and capturing the surface reflectance by appropriate material filtering of the Disney BRDF parameters. We use these parameters during shading as well as a randomly selected normal from our fitted normal distribution to build up a BRDF and sample from it. This approach is simple and efficient and allows for soft highlights filtering of fine grained surface variation.

2. Related Work

Level of Detail is a broad field in computer graphics. In this section we differentiate between mesh-based LOD, such as geometric simplification or Mip-Mapping, and voxel-based LOD, which include point-based approaches.

Mesh-Based LOD For an extensive survey on polygonal simplification of complex geometries see [LWC*02]. A similar approach is R-LOD [YLM06], which approximates the geometry inside each BVH node with a plane. This works well for primary visibility, however, especially for highly curved geometry, the plane approximation introduces holes,

DOI: 10.2312/vmv.20151260

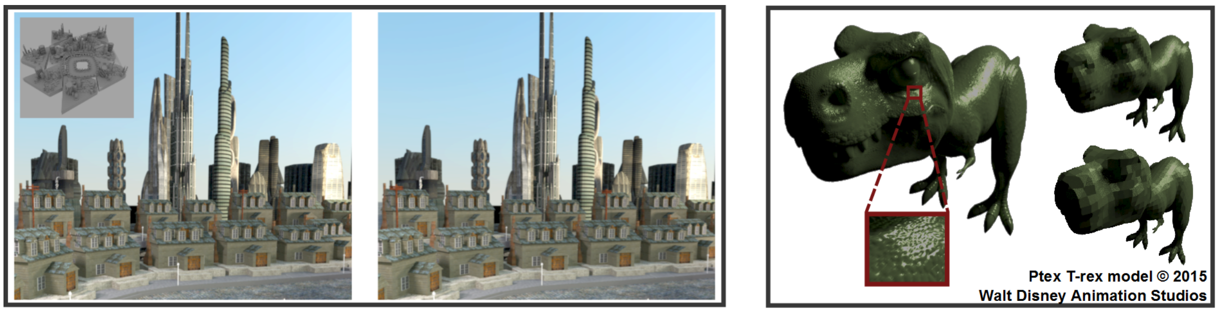

Figure 1: *Our LOD approach reduces memory requirements for ray tracing complex environments. With our simple filtering of BRDF parameters and normal distribution we maintain reasonable results and allow complex scenes to stay longer in-core. The observed memory savings are dependent on the input geometry and acceptable quality degradation: For the city scene (left), mainly consisting of large textured polygons we reduce memory footprint by 50% without noticeable image differences. For the highly tessellated T-Rex with displacement (right) we reduce the memory footprint from 20.4 MB to only 0.4 MB (top) and 0.08 MB (bottom).*

leading to uncontrollable light leaking when used with full GI. Different mesh based simplification approaches target specifically aggregate geometry as in i.e. [LBBS08] or [CHPR07]. Lacewell et al. [LBBS08] prefilter aggregate geometry from different view points resulting in less variance in soft shadows. Cook et al. [CHPR07] on the other hand discard parts of aggregate geometry and adjust the remaining objects to have similar lighting behaviour. However, both approaches are targeted to aggregate geometry only.

When simplifying geometry, material reflectance has to be considered as well. In real-time graphics often simple and fast linear filtering approaches are employed, e.g.texture mip-mapping [Wil83]. However, when applied on normal or displacement maps they fail to correctly capture reflectance properties coming from microfacetted surfaces. In such case nonlinear prefiltering methods, as surveyed in [BN12], are more accurate. Bruneton et al. [BNH10] for instance, approximate high frequency geometric detail with a rough BRDF model when viewed from afar and smoothly transitions between geometry and the BRDF model. They apply this technique for realtime realistic ocean rendering. A more recent approach is Linear and Efficient Antialised Displacement Rendering (LEADR) as introduced in [DHI*13]. They use simple Gaussian distributions to approximate geometric detail and self shadowing from displacement maps. Their approach is closely related to Linear and Efficient Antialiased Normal (LEAN) mapping ([OB10]) which handles normal maps only. However, both approaches filter in the tangent space and not over geometric boundaries, thus are restricted to single-bounce filtering. Other approaches which handle multi-lobe distributions ([HSRG07], [TLQ*08]) suffer from expensive pre-processing when searching for appropriate filtering lobes.

Voxel-Based LOD In the past years, brickmaps as introduced in [CB04] and thouroughly presented in [Chr10] have been extensively used to compute diffuse global illuminatio for highly complex environments in movie production. Recently, Kontkanen et al. [KTO11] improved this approach by allowing efficient processing of out-of-core point clouds. However, in both approaches setting up those point clouds, generating brickmaps and managing those data structures requires manual intervention. Voxel octrees have been extensively used in many computer graphics applications, e.g. Benson et al. [BD02] introduced octree textures for simple 3D painting. Later, GigaVoxels were proposed by Crassin et al. [CNLE09], which efficiently compute only primary visibility for large volumetric data sets. Another efficient voxel structure are Sparse Voxel Octrees (SVO), proposed by Laine et al [LK10], which have been used in many interactive illumination approaches, such as in the voxel cone tracing framework [CNS*11] as well as in production environments [PMA14]. Filtering approaches, which consider fine surface detail([HN12]) improve on that. Our work is closely related to the work of Laine et al. [LK10] as we use an SVO to encode the geometry. However, we store additional parameters for each voxel to reconstruct accurate material reflectance to be able to derive secondary paths for ray tracing.

3. Overview

We use SVOs [LK10] as a main data structure for LOD in a ray tracing environment. SVOs are well suited for LOD; they implicate simple filtering by their hierarchical data structure and, most importantly, their memory footprint is much smaller compared to geometric representations.

On scene loading, we automatically detect appropriate candidates for aggregation (Section 4). These groups are then approximated (Section 5) and stored in a compact data structure (Section 6). We use this LOD approximation for primary as well as secondary rays, and present according results in Section 7 before concluding the paper (Section 8).

4. LOD Aggregation

For our LOD selection we use a view-dependent metric. We compute the voxel resolution based on the object's bounding box b with edge lengths b_x, b_y, b_z and ray diameter d:

$$v_{\text{res}} = \frac{\max(b_x, b_y, b_z)}{d(||e-b||)}. \tag{1}$$

Similar to [YLM06] $d(||e-b||)$ describes the ray diameter at the shortest distance between the camera position e and box b, which is an isotropic approximation of ray differentials [Ige99].

For larger and close objects this can lead to SVOs with high resolution, and memory footprint larger than the original geometry. To avoid the expense of computing such high-resolution SVO, we use a simple threshold $v_{\max}$: If the SVO requires a higher resolution than $v_{\max}$ we do not compute the SVO, and render the original geometry instead. $v_{\max} = 256$ has shown to be a good trade-off between quality, performance and memory consumption. In addition, we expose a user-defined LOD tolerance factor λ, which denotes the maximal voxel extent in pixels. In order to determine whether an LOD is appropriate for an object we then use

$$v_{\text{res}} \leq \lambda \cdot v_{\max}. \tag{2}$$

For offscreen objects, we assume that they are hit by wide secondary rays only, thus we apply an additional multiplier on λ to allow more aggressive simplification.

To account for spatial as well as logical grouping, i.e. similar elements should be grouped into one LOD object in order to preserve shading context and exploit cache coherency, we apply the above selection scheme in a recursive two-step algorithm. We first create a logical scene hierarchy $H_{\log}$ and check the current node nd, e.g. in the first iteration this is the scene's root node itself, whether it satisfies Equation (2). If this is the case we apply our LOD scheme on nd; otherwise, we build a spatial hierarchy $H_{\text{spa}}(nd_{\text{child}})$ over all first descendants of nd. $H_{\text{spa}}(nd_{\text{child}})$ is then traversed in a top-down manner in order to find spatially nearby aggregation partners. If there are leaf nodes of $H_{\text{spa}}(nd_{\text{child}})$ left after the traversal we return to $H_{\log}$ and proceed with the next finer level. Each group satisfying Equation (2) is converted into a separate SVO while the underlying geometry is discarded.

5. LOD Filtering

To build our LOD structure, we generate points by casting rays orthogonal to the six sides of the bounding box. This gives us uniform samples and prevents color leaking between nearby geometry and also allows us to discard invisible geometry leading to less memory consumption. At each sampled point we then compute its position p, its geometric normals n_{geom}, as well as its shading normal n_{shade}. Further, each point contains also material properties, such as color, e.g. from texture, and a set of BRDF parameters.

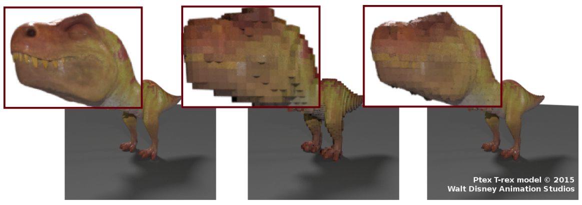

Figure 2: *From left to right: input geometry, SVO approximation without and with contour planes.*

5.1. Filtering geometry

Similar to [LK10], we use additional contour planes to improve geometric detail, where we restrict the planes' direction to the average normal $\bar{n}_{\text{geom}}$. During traversal of the SVO we additionally intersect rays against the approximating planes. Figure 2 shows the T-Rex model, the voxelized approximation at a resolution of 64^3 without and with contour planes. Besides improving the overall silhouette of the model, the plane approximation additionally removes undesired self-shadowing caused by nearby voxels. Notice, by using two planes the voxel approximation stays watertight, guaranteeing no light leaking, which may lead to undesirable fireflies under full GI.

5.2. Filtering material properties

We choose a subset of ten parameters from the Disney BRDF [Bur12]. This material model can approximate a wide range of physical materials. Further, its parameters are designed to be perceptually linear and produce a *plausible* BRDF when blending the parameters of different materials.

To obtain a *filtered material* for each voxel, we simply average those Disney BRDF parameters and texture colors. Note that this filters the material *properties* and not the material *response*. The response of the average material will be different from the average response of the original asset.

5.3. Normal Filtering

In order to capture high-frequency detail, e.g. from displaced surfaces, a proper description of the surface normal variation is required. We use the $\bar{n}_{\text{geom}}$ from Section 5.1 to define a normal distribution around it. To compute the distribution for each voxel we use all shading normals n_{shade}, which may come from displaced geometry or bump maps within that voxel. As opposed to normal filtering approaches described in [OB10], or [HSRG07], where the normal distribution is integrated into the BRDF model, we use the normal distribution to randomly draw a normal during shading. This approach is simple and allows us to use any, even more complex shading system, without any further modifications. In the following we describe three different distributions.

The simplest normal distribution is an isotropic Gauss distribution, which can be efficiently described by a von Mises-Fisher (vMf) distribution [Sra12]. The function features two parameters: the average normal direction $\bar{n}_{\text{geom}}$ and κ, the variance around $\bar{n}_{\text{geom}}$.

For anisotropic input normals, we can use a 3D Gaussian distribution, which requires a covariance matrix, that can be computed linearly as in [YLM06]. However, storage requirements are heavy, because in addition to $\bar{n}_{\text{geom}}$ we have to store the covariance matrix. Alternatively, we can use an ellipse distribution, which captures anisotropy and only needs little additional memory. We first project all normals n_i inside a voxel onto the tangent plane defined by $\bar{n}_{\text{geom}}$ resulting in the projected normals $n_{i,\text{proj}}$ (see Figure 3). Then, we perform a principal component analysis (PCA) on $n_{i,\text{proj}}$, retrieving the Eigenvalues $\sigma_0 < \sigma_1 < \sigma_2$, out of which σ_1 and σ_2 define the extents of the 2D ellipse. Storing these two values as well as the angle α between the Eigenvector $u_{\max}$ and the x-Axis of the oriented system around $\bar{n}_{\text{geom}}$ is sufficient. When shading, we sample according to the ellipse defined by σ_1 and σ_2, transform the sampled normal to the local coordinate system around $\bar{n}_{\text{geom}}$ and normalize the vector.

Figure 4 shows the histogram of input normals (a) and compares the histograms of randomly selected normals based on the according distributions. Notice, the 3D Gauss as well as the Ellipse distribution capture the anisotropy of the input, while the isotropic vMf distribution fails.

To evaluate our distributions more accurately we use the Jensen Shannon divergence (JSD), which is a symmetric similarity metric for probability distributions. For each pixel, we compute the JSD by ray casting 512 samples on the approximation and comparing the resulting histograms with the ground truth histogram, which is the five times subdivided and displaced T-Rex model in Figure 1. Figure 5 shows the heatmap coded JSD (red indicating high JSD and blue low JDS) from left to right for a low-poly approximation, as well as for our LOD scheme using the vMf, ellipse, and 3D Gaussian. The ellipse distribution matches closely the results of the 3D Gauss distribution, and both show lower JSD (back of the T-Rex) than the low-poly approximation. The row below displays the rendered results. Notice, we use a uniform color on the T-Rex to highlight the material reflectance, however, we are by no means restricted to uniform colors, e.g. as in Figure 2.

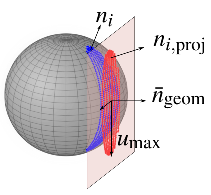

Figure 3: *Fitting the ellipse distribution: Input normals n_i (blue) are projected to the tangent plane defined by $\bar{n}_{geom}$ resulting in $n_{i,proj}$ (red), upon which we perform a PCA analysis in order to obtain the ellipse parameters.*

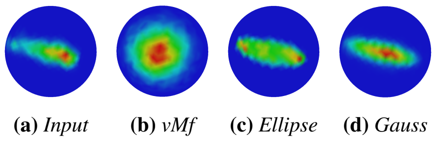

Figure 4: *Histograms of sampled normals using the parameters obtained by analyzing the input normals from (a).*

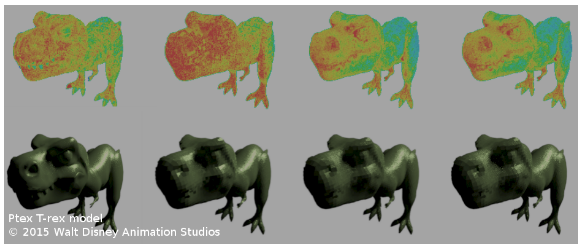

Figure 5: *Comparison of our LOD approach with different normal distributions using JSD and final rendering, where from left to right: low polygon approximation, our LOD approach using the vMF, ellipse, and Gauss distribution.*

6. LOD Data Layout

Table 1 gives a detailed list of all data which is stored for each voxel. Material properties, such as the Disney BRDF parameters, are clustered and stored independently. Specifically, we compress the 10 BRDF parameters by storing half of them as half floats, whereas the other half can be linearly quantized to 8 Bit per parameter, resulting in 15 Byte per parameter set.

The SVO linking uses 4 Byte per node, whereas its geometry is encoded using 8 Byte: 2 Byte for each plane offset, 4 Byte for the geometric normal, which is compressed as in [MSS*10], and 2 Byte to index the correct BRDF parameters. The memory requirements for the normal distributions

Data Structure	Size [Byte]	T-Rex[KB]
Inner Nodes	4	5.2
Geometry	10	50.7
Normal (vMF)	2	10.4
Normal (Ellipse)	6	31.4
Normal (Gauss)	12	62.9

Table 1: *LOD data consumption per voxel, and total for the T-Rex model in Figure 2 at a resolution of 64^3. The original geometry requires 20.43 MB, whereas our LOD structure needs 0.08 MB only. Construction time took 2.9 sec.*

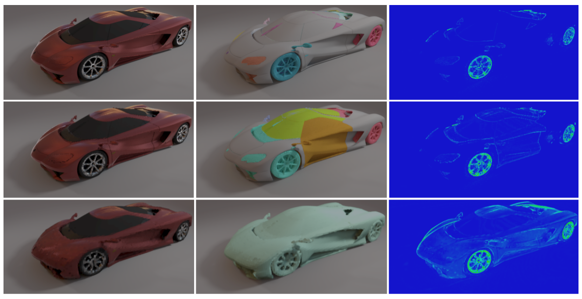

Figure 6: *From top to bottom: car model using increasing λ=1,2, and 7 requiring 62%, 55%, and 0.16% of the original geometry with an RMSE=0.08, 0.10, and 0.12. From left to right: the rendered result (700x500,512 SPP), the aggregation groups and corresponding difference images.*

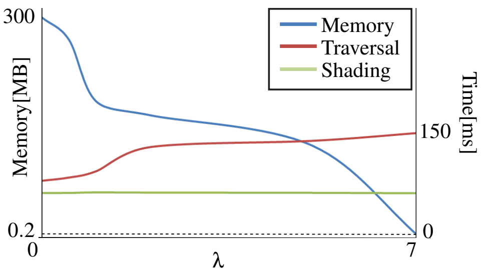

Figure 7: *Timings for traversal, shading and memory consumption at different λ for the car model in Figure 6.*

differ. With half-float precision, the vMf needs 2 Byte, the 3D Gauss distribution is encoded using 12 Byte, whereas the Ellipse distribution requires 6 Byte: 2 Byte encode the angle, whereas the ellipse extents require 2 Byte each.

7. Results

We have integrated our LOD scheme into the Embree CPU raytracing framework [WWB*14]. All results were rendered using 8 bounces of indirect light on a 8-core i7-2600 CPU. We include additional heat map coded difference images comparing our approximation to the ground truth render.

Figure 6 shows the car model for different λ, where from top to bottom we have increased λ. The middle row images depict the different aggregation groups, where each color indicates a separate group and grey the original geometry. For instance, in the top image each tire is divided into a separate aggregation group, whereas in the middle both front tires belong to one group and in the bottom image the entire car is represented by one single SVO. Clearly, using a higher λ has impact on the overall appearance resulting in coarser approximations, but less memory. The according timings and memory requirements are shown in Figure 7. Notice that, already for small λ, we reduce the memory footprint by 38% with almost unnoticably quality decrease (RMSE=0.08) and only

λ	RMSE	Memory [GB]		Time	Build
		Geometry	Material	[sec]	[sec]
0	0	1.21	3.53	109.8	0.0
1	0.03	0.46	1.97	113.2	22.3
3	0.08	0.39	1.25	136.1	28.1
5	0.12	0.38	0.72	162.4	21.5

Table 2: *RMSE, render times and memory consumption for different λ for the city scene in Figure 8.*

slight traversal slowdown. With larger λ, memory is further reduced up to 84% and image quality is degrading. However, there is no noticeable error in the shadow of the car on the ground plane, indicating that for objects outside the viewing frustum, which are hit by wide secondary rays only, this aggressive approximation is already sufficient. When the scene fits in-core, the LOD traversal slows down the overall rendering performance. However, our main concern is to fit scenes into main memory which avoids swapping of data, which increases render times by several orders of magnitude [YLM06].

Figure 8 displays the city model along with its aggregation groups from the camera view as well as from afar for $\lambda = 1,2,7$ (top to bottom). The according rendering times and memory requirements are shown in Table 2. In this particular scene, we reduce the overall memory footprint of 4.74 GB to 2.74 GB with no significant image degradation ($\lambda = 1$). This is due to aggressive simplification of objects outside the viewing frustum. With more aggressive LOD the image quality suffers more noticeably. However, using λ as an artist-controllable parameter, different objects, which are more sensible to minification (e.g. power poles), can be assigned different values of λ. According LOD build times are stated in Table 2. Notice that for $\lambda = 5$ build time is reduced compared to $\lambda = 3$, which is due to the fact that more geometry is aggregated into one SVO, thus, there are less SVOs built.

8. Conclusions and Future Work

We have presented a simple LOD approach for path tracing complex environments. Our approach is well-behaved for geometric approximation as well as material filtering. Using the Disney BRDF and its perceptually linear parameters allows very simple material filtering, which limits our approach to the Disney BRDF model. However, this model can approximate a broad variety of different physical materials. Further, with our normal distribution, we have proposed a simple filtering approach which allows filtering of the Disney BRDF model as well as underlying complex displacement. With this LOD approach we can decrease model complexity and allow even complex scenes to stay longer in-core, thus increasing rendering performance.

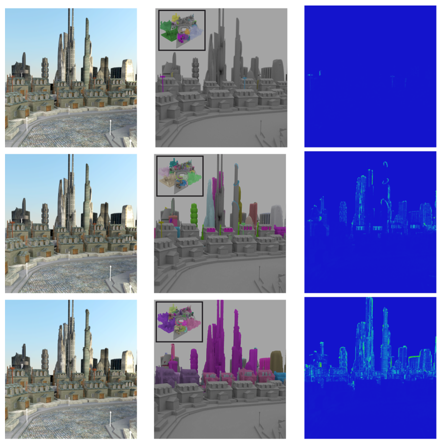

Figure 8: *We show rendered images (700x700, 128 SPP) for the city scene as well as aggregation and difference images for increasing λ=1,3, and 5. Memory requirements and render times are stated in Table 2.*

Our current approach does not consider view-dependent material properties. However, this can be solved easily by additional material parameter per voxel, similar to [BD02]. Further, our approach mainly concentrates on static and non-aggregated geometry as is present in large environments. Future work involves expanding our approach for time-aware objects for motion blur, as well as for aggregate geometry, i.e. hair and fur.

Acknowledgements

The authors would like to thank Disney for giving access to the T-Rex model and its detailed textures. We also thank Herminio Nieves for the city scene and Yasutoshi Mori for the car model.

References

[BD02] BENSON D., DAVIS J.: Octree textures. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (2002), SIGGRAPH '02, ACM. 2, 6

[BN12] BRUNETON E., NEYRET F.: A survey of nonlinear prefiltering methods for efficient and accurate surface shading. *IEEE Trans. on Visualization and Computer Graphics 18* (2012). 2

[BNH10] BRUNETON E., NEYRET F., HOLZSCHUCH N.: Real-time realistic ocean lighting using seamless transitions from geometry to brdf. *Comput. Graph. Forum 29* (2010), 487–496. 2

[Bur12] BURLEY B.: Physically-based shading at disney. In *ACM SIGGRAPH 2012 Courses* (2012), SIGGRAPH '12, ACM. 3

[CB04] CHRISTENSEN P. H., BATALI D.: An irradiance atlas for global illumination in complex production scenes. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (2004), EGSR'04. 1, 2

[CHPR07] COOK R. L., HALSTEAD J., PLANCK M., RYU D.: Stochastic simplification of aggregate detail. In *ACM SIGGRAPH 2007 Papers* (2007), SIGGRAPH '07, ACM. 2

[Chr10] CHRISTENSEN P.: Point-based global illumination for movie production. In *ACM SIGGRAPH 2010 Courses* (2010), SIGGRAPH '10, ACM. 2

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)* (2009), ACM. 2

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum* (2011). 2

[DHI*13] DUPUY J., HEITZ E., IEHL J.-C., POULIN P., NEYRET F., OSTROMOUKHOV V.: Linear efficient antialiased displacement and reflectance mapping. *ACM Trans. Graph. 32* (2013). 2

[ENSB13] EISENACHER C., NICHOLS G., SELLE A., BURLEY B.: Sorted deferred shading for production path tracing. *Computer Graphics Forum 32*, 4 (2013), 125–132. 1

[HN12] HEITZ E., NEYRET F.: Representing appearance and pre-filtering subpixel data in sparse voxel octrees. In *Proceedings of ACM SIGGRAPH / HPG* (2012). 2

[HSRG07] HAN C., SUN B., RAMAMOORTHI R., GRINSPUN E.: Frequency domain normal map filtering. In *ACM SIGGRAPH 2007 Papers* (2007), SIGGRAPH '07, ACM. 2, 3

[Ige99] IGEHY H.: Tracing ray differentials. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (1999), SIGGRAPH '99, ACM, pp. 179–186. 3

[KTO11] KONTKANEN J., TABELLION E., OVERBECK R. S.: Coherent out-of-core point-based global illumination. *Computer Graphics Forum 30*, 4 (2011), 1353–1360. 2

[LBBS08] LACEWELL D., BURLEY B., BOULOS S., SHIRLEY P.: Raytracing prefiltered occlusion for aggregate geometry. In *IEEE Symposium on Interactive Raytracing 2008* (2008). 2

[LK10] LAINE S., KARRAS T.: Efficient sparse voxel octrees. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010). 2, 3

[LWC*02] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002. 1

[MSS*10] MEYER Q., SÜSSMUTH J., SUSSNER G., STAMMINGER M., GREINER G.: On floating-point normal vectors. *Computer Graphics Forum 29*, 4 (2010), 1405–1409. 4

[OB10] OLANO M., BAKER D.: Lean mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), I3D '10, ACM, pp. 181–188. 2, 3

[PMA14] PALMER S., MAURER E., ADAMS M.: Using sparse voxel octrees in a level-of-detail pipeline for rio 2. In *ACM SIGGRAPH 2014 Talks* (2014), SIGGRAPH '14, ACM. 2

[Sra12] SRA S.: A short note on parameter approximation for von mises-fisher distributions: and a fast implementation of is(x). *Computational Statistics 27*, 1 (2012), 177–190. 4

[TLQ*08] TAN P., LIN S., QUAN L., GUO B., SHUM H.: Filtering and rendering of resolution-dependent reflectance models. *IEEE Trans. on Vis. and Computer Graphics 14* (2008). 2

[Wil83] WILLIAMS L.: Pyramidal parametrics. In *ACM Siggraph Computer Graphics* (1983). 2

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: a kernel framework for efficient cpu ray tracing. *ACM Trans. Graph. 33*, 4 (2014), 143. 5

[YLM06] YOON S.-E., LAUTERBACH C., MANOCHA D.: R-lods: Fast lod-based ray tracing of massive models. *Vis. Comput. 22*, 9 (Sept. 2006), 772–784. 1, 3, 4, 5