Lighting Details Preserving Photon Density Estimation

Robert Herzog
MPI Informatik, Germany

H.-P. Seidel
MPI Informatik, Germany

Abstract

Standard density estimation approaches suffer from visible bias due to low-pass filtering of the lighting function. Therefore, most photon density estimation methods have been used primarily with inefficient Monte Carlo final gathering to achieve high-quality results for the indirect illumination. We present a density estimation technique for efficiently computing all-frequency global illumination in diffuse and moderately glossy scenes. In particular, we compute the direct, indirect, and caustics illumination during photon tracing from the light sources. Since the high frequencies in the illumination often arise from visibility changes and surface normal variations, we consider a kernel that takes these factors into account. To efficiently detect visibility changes, we introduce a hierarchical voxel data structure of the scene geometry, which is generated on GPU. Further, we preserve the surface orientation by computing the density estimation in ray space.

1 Introduction and Related Work

Since the beginning of computer graphics, many work has been devoted to the problem of generating realistic images with physically plausible lighting in synthetic scenes referred to as *global illumination* (GI). In this work we focus on approximate and biased algorithms for solving the GI problem efficiently. Our algorithm uses a novel photon splatting technique capable of handling all-frequency illumination using photon density estimation (PDE) in ray space [5], which does not introduce visible bias as in standard PDE algorithms. This is because we explicitly compute the visibility during the density estimation via discrete occlusion testing in a voxelized scene. Using a voxel representation of a polygonal scene is not new but has regained attention due to modern programmable graphics hardware. In [3] the authors present a robust method to generate an oct-tree hierarchy of a solid voxelization of arbitrary polygonal scenes in a slow offline process. In [1, 2] it was shown how to efficiently generate a voxelized grid in real-time on the GPU. Surfaces of the scene are rasterized to uniform grid slices in multiple passes. Such approach has also been used in this paper, although we target on efficient hierarchical data structures. Moreover, we also show how to approximate the average surface normal per voxel, which is ignored in [1, 2].

Our work is related to [4], which proposed to perform the nearest neighbor search, essential for all density estimation techniques, for a disc in the tangent plane of a surface point. This leads to the elimination of *boundary bias* inherent to photon maps. However their method needs complex data structures for searching and still suffers from *topological bias* and *proximity bias* [9]. Since a direct visualization of the photon density leads to poor quality, expensive final gathering must still be performed to achieve high-quality indirect illumination [6]. Another popular GI algorithm is *instant radiosity* [8] (IR). Similar to PDE, IR caches and reuses lighting information on scene surfaces to suppress noise at the expense of bias. However, for the view-dependent gathering pass, PDE computes the irradiance directly from the local photon density, while IR integrates the incident radiance over the whole scene area shooting a shadow ray to each virtual point light source (VPL). It reproduces high-frequency lighting features but its brute-force nature allows only a small set of VPLs, which is insufficient for glossy light transport. A few extensions to IR have been developed for choosing a subset of VPLs adaptively per pixel sample [11]. Our method can be understood as a trade-off between IR and PDE.

2 Photon Density Estimation in Ray-Space

In standard PDE methods [6] a photon contributes energy $\Delta\Phi$ to its local neighborhood, which is weighted by a normalized 2D density estimation (DE) kernel $\mathcal{K}_h$ with bandwidth (radius) h. The sum over all energy contributions received by a differential surface $dA(y)$ at point y, which we call *eye sample*, is an approximation to the irradiance $E(y)$.

$$E(y) = \int_{\Omega^+} L(y,\omega)\cos\theta\, d\sigma(\omega) \approx \sum_i^K \mathcal{K}_h(y,x_i)\frac{\Delta\Phi_i(x_i,\omega_i)}{\Delta A(y)}, \quad (1)$$

where Ω^+ is the upper hemisphere of incident directions, $L(y,\omega)$ is the incident radiance at y from direction ω for differential solid angle $d\sigma(\omega)$, $\Delta\Phi(x,\omega)$ is the photon flux, and K is the number of photons under the DE kernel with footprint area $\Delta A(y)$. For computing the outgoing radiance along a direction the incident flux is convolved with a BRDF f_s.

Since all DE techniques need to consider a finite neighborhood to estimate the density, they suffer from bias [10]. To reduce the bias, we can decrease the kernel bandwidth. This, however, increases the noise in our estimate. Another possibility is to split the irradiance function we are trying to estimate into different components that we solve explicitly. The low-frequency components of the function are still computed via DE while the high-frequencies are computed accurately since they are the main source of bias in PDE, which mostly result from surface normal variation and wrong visibility assumptions in the neighborhood. The former is solved by operating in ray space [5] and decoupling the PDE from

the surfaces. The latter is more complex because it requires to evaluate the visibility function inside the DE footprint between the origin of the photon (x_{i-1}) and all eye samples in the footprint, which ruins the efficiency of PDE. Nevertheless, we provide an efficient approximation for the visibility estimation. We can rearrange (1) to exclude the visibility function $V(y,x)$ and the surface orientation $\cos\theta$ from the DE to be computed explicitly:

$$E(y) \approx \sum_{i}^{K} \mathcal{K}_h(y, x_i, \omega_i) V(y, x_{i-1}) \frac{\Delta\Phi_i(x_i, \omega_i)\cos\theta_i}{\Delta A^{\perp}_{\omega_i}(y)}, \quad (2)$$

where $\mathcal{K}_h(y, x_i, \omega_i)$ is the DE kernel whose domain is oriented perpendicular to the direction ω_i. The kernel evaluates the distance of y to the photon ray (x_i, ω_i) [5]. The point x_{i-1} is the origin of the photon ray and θ_i is the angle between the photon ray and the surface normal at point y. $\Delta A^{\perp}_{\omega_i}(y) = \Delta A(y)/\cos\theta_i$ is the area of the unprojected DE footprint. See Fig. 1 for a geometric interpretation in 2D. To preserve the photon energy $\Delta\Phi_{rgb}$, the axis-aligned kernel footprint (shaded parallelogram) is tested for occlusion along its traversal. Fig. 1 shows the masked kernel K_h (blue) and the occluded areas (thick black curves) and visible areas (thick grey curves). The photon flux is only splatted to eye samples e_1 because e_2 is back-facing, e_3 is occluded, and e_4 is outside the kernel footprint.

Note that we partially compute the geometric term $G(x,y) = V(x,y)\frac{\cos\theta_x \ \cos\theta_y}{||x-y||^2}$. In contrast to instant radiosity [8], our method does not suffer from singularities near corners since the squared distance term in the geometric term is implicitly handled by the density estimation. Since we use a splatting approach, one photon ray spreads its energy to all visible eye sample points y inside its kernel footprint as shown in Fig. 1.

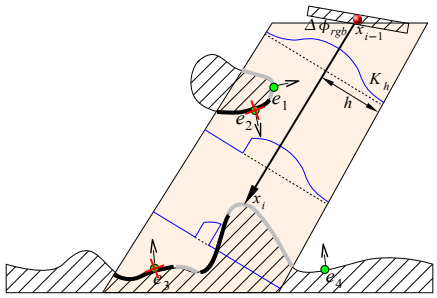


Figure 1. Photon splatting in ray-space with a kernel K_h (blue) preserving occlusions and surface orientation.

3 Algorithm Outline

Our density estimation algorithm extends the splatting technique described in [5]. First, all primary rays are shot from the eye. Then hit-point records with the scene surfaces referred to as *eye samples* are stored. Second, the scene is voxelized into a grid using the GPU and a voxel hierarchy is constructed over the grid. Next, the photons are traced through the scene using Quasi-Monte Carlo sampling until a desired number of direct, caustics, and global indirect photons has been stored. During the photon sampling a splat radius is computed from the photon path probability density for each photon and each type of light transport [5]. In the following phase all photon rays splat their energy to the image plane using a novel density estimation technique (Section 2) including a search for the nearest-neighbor eye samples in a cylindrical volume. For accelerating the nearest neighbor search and at the same time testing for occlusion, the generated voxel hierarchy is traversed in front to back order in the spirit of raytracing with kd-trees.

4 Synthetic Scene Voxelization

The basis of our algorithm is the discretization of the scene model into voxels. As in [1, 2] the voxelization process runs as follows: First the scene model is, if not yet present, tessellated to triangles. Optionally, we test the polygonal scene for wrongly oriented triangles by performing a random walk by means of raytracing through the whole scene. Triangles that are detected as back-facing are flipped and triangles, which were flipped more than once are marked as two-sided. Next, all triangle vertices are passed to the GPU memory as vertex buffer objects. For all three dimensions, the axis-aligned bounding box of the scene is divided into intervals composed of multiple grid slices (maximum 64). For each interval all triangles are rasterized by an orthographic camera with its frustum defined by the current grid interval boundaries. All triangles falling into the currently processed interval are voxelized in the fragment shader according to their interpolated z-distance (logic 'OR' mode) such that each bit in a color channel determines whether a voxel of a slice is empty or not [2]. We also need to distinguish between front and back-facing fragments. In addition, we apply 4×4 super-sampling per voxel for anti-aliasing purposes and also for the reconstruction of a discretized average normal per voxel assuming the surface inside the voxel is piecewise linear. The outcome is a 3D grid consisting of cubic voxels each storing the discrete occlusion ratios for all six sides (6 bytes).

5 Constructing the Voxel Hierarchy

As for raytracing, a grid has the disadvantage that it cannot adapt to local scene complexity and does not scale well with its resolution. Therefore, we build a hierarchy, more precisely a kd-tree, on top of the precomputed grid that merges all empty voxels during its construction.

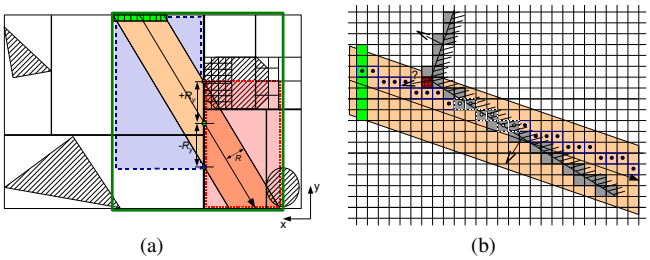
The kd-tree is built on the CPU in a recursive top-down fashion and the axis-aligned splitting plane of a node is always aligned with the boundary of a voxel. The plane is either positioned at the spatial-median voxel for the largest dimension of the node's bounding box or at the closest non-empty voxel if either half-space is empty. For efficiently culling empty space, we build a 3D summed area table (SAT) of the occlusion grid, which is only kept temporarily during the tree construction. The construction time for the 3D SAT is negligible compared to the rest of the computation. The computation for the best splitting plane according to our heuristic can then be computed in constant time. The discretization of the tree construction increases not only the performance but also allows for higher compression of the size of the kd-tree nodes. Each node consists of only 8 bytes and there are

4 basic node types: *Empty Nodes*, *Splitting Nodes*, *Occlusion Nodes*, and *Visible Nodes*. Empty nodes represent empty space that can be skipped. Splitting nodes sub-divide their associated bounding box into half-spaces. An occlusion node is created when either a voxel is reached (the smallest entity) or the occlusion in the sub-tree is uniform. Occlusion nodes contain information about local occlusion (1 byte) and the average quantized normal (3 bytes), which is used to determine if occlusion is feasible. A visible node stores the index and the number of eye samples associated with the node and has always one child node. The hierarchical voxelization results in a small memory footprint even for high resolution grids.

5.1 Traversal of the Voxel Hierarchy

Similar to raytracing with kd-trees, we recursively traverse the tree from the root to the leaves in near to far order. This requires maintaining a stack. In standard raytracing the recursive tree traversal works in 1D ray space. Since we deal with a volumetric ray, we need to keep track of the minimum and maximum bounds tightly encompassing the ray volume in three dimensions. The ray volume is represented by a cylinder, whose radius is defined by the photon's splat radius. A volumetric traversal is more complex than a 1D ray traversal and we need to make simplifying assumptions: first, the kd-tree and most parameters are discretized to the grid resolution and second, the kd-tree and its traversal are axis-aligned and we restrict the recursive sub-division of the ray volume to axis-aligned bounding boxes starting with the whole bounding box of the ray volume (Fig. 2a). For the sub-divisions of the ray volume, we only need to consider cross-sections with the cylinder and the axis-aligned splitting planes of the kd-tree in order to compute the new bounding boxes for the front and back side of a splitting node. The height and width of each cross-section in each dimension are precomputed by projecting the cylinder onto the three axis-aligned planes. The major traversal axis N corresponds to the largest component of the ray direction. The axes U and V span an elliptically shaped slice of the cylinder.

Having precomputed these values, we can quickly compute the new ray bounds (Fig. 2a) for the near (blue box) and the far side (red box) of a splitting node in one of the three splitting axes. Occlusions by surface voxels are stored in an



(a) (b)

Figure 2. (Left) A volumetric 2D traversal step for a splitting node in the kd-tree. (Right) Self-occlusion (dashed point-squares) on a flat surface.

(U,V)-aligned 2D buffer, which we call the *occlusion mask*. All non-empty voxels update the occlusion in the corresponding cell of the occlusion mask until a cell is fully occluded. To avoid self-shadowing, only voxels in front of the tangent plane at the ray's origin update the occlusion. For all visible voxels, the photon energy is splatted to pixels associated with their eye samples. The photon energy contribution to an eye sample is weighted by the factors given in (2). The term $V(y, x_{i-1})$ is approximated by the occlusion weight in the cell of the occlusion mask, which can be bilinearly interpolated between the four nearest neighbor cells according to the eye sample's position in the voxel. Note that all positions and distance values are represented in grid coordinates for ease of computation.

Finally, the radiance contribution of the photon ray to all affected pixels in the image is weighted by the local surface BRDF and the pixel weight associated with the eye samples.

5.2 Discretized Occlusion

Since we do not regard infinitesimal thin rays but deal with discrete volumetric rays, many discrete occlusions can map to the same cell of the occlusion mask depending on the incident angle with a surface, which leads to self-occlusion. In Fig. 2b the high-lighted voxels (point-squares) show the state of one particular cell in the history of the occlusion mask (green). The grey squares represent surface voxels. The dashed point-squares indicate the traversed voxels that are possibly self-occluding. Adding a small threshold to the occluder distance when comparing it with the current surface distance, may produce visible light leakage for near occluders. Instead we compute the minimum occluder distance adaptively depending on the incident angle of the ray to the voxelized surface (distance from first to last dashed square in Fig. 2b). Yet a simpler way is to update the occlusion only for back facing voxels, which fails for ambiguous voxels that contain front and back facing surfaces. In either case we need the surface normal in the occluder voxel.

5.3 Estimating the Surface Normal

The approximate surface normal in a voxel is implicitly computed during the 3D rasterization. Assuming that a voxel does not contain front and back facing surfaces simultaneously, the average quantized normal $\vec{N}$ can be approximated from the 6 discrete front and back facing occlusion ratios stored in a voxel, see Fig. 3.

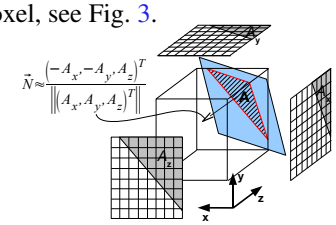


Figure 3. Estimating the average surface normal in a voxel from the 3 axis-aligned projections A_x, A_y, A_z of a surface (blue) with clipped area A inside the voxel.

For the back facing test the normalization of $\vec{N}$ can be avoided since we are only interested in the sign of the dot product between ray direction and $\vec{N}$. In the case of ambiguous voxels (i.e. front and back facing), we add an offset to the occluder distance, which is measured along the dimension of highest occlusion in the voxel (z in Fig. 3). This dimension is precomputed and stored in the corresponding node during the hierarchy construction.

Scene	Method	Memory	Build	Eye	Light	Total
GLOSSY BOX (81 % diffuse)	Ray-splat 192×192×192	(5+26+6)	3.2	1.4	54	59
	Lightcuts $\bar{s}$=431	(14+5)	0.8	626	1.2	628
	Path-tracing (2500)	–	–	2720	–	2720
Scene settings	$500 \times 500 \times 4$; $M = 100,000$; (20%, 50%, 30%)					
OFFICE (91 % diffuse)	Ray-splat 384×240×384	(13+23+3)	5.2	1.5	58	65
	Lightcuts $\bar{s}$=261	(8+3)	0.9	390	0.5	391
Scene settings	$500 \times 500 \times 4$; $M = 55,000$; (40%, 60%, 0%)					
CONFERENCE (86 % diffuse)	Ray-splat 560×368×160	(17+23+9)	10.6	3.4	98	112
	Lightcuts $\bar{s}$=321	(21+7)	5.6	680	3.4	689
Scene settings	$500 \times 500 \times 4$; $M = 150,000$; (35%, 65%, 0%)					

Table 1. Computation times (seconds) and memory consumption (megabytes) for the rendering phases of our ray splatting method, instant radiosity with lightcuts, and path tracing. The images are shown in Fig. 5.

6 Results

We have evaluated our method in comparison with instant radiosity (IR) combined with lightcuts [11] using three scenes of different complexity and lighting condition. The scene setting and the photon distribution is the same for both methods. The resulting images are shown in Fig. 5. The rendering times and parameters are given in Table 1. All results were computed with an Athlon 64 X2 2.2 GHz Processor using one core. The scene settings are: image resolution times the number of samples per pixel, total number of stored photons (M); and the percentage of stored direct, global indirect, and caustics photons. For our ray-splatting method the entries in the table show (from left to right): grid resolution, memory for the occlusion tree plus eye samples plus photon rays, time to build the raytracing kd-tree and the occlusion hierarchy, time for shooting eye rays, time for photon sampling and ray splatting, and total time needed to compute the image.

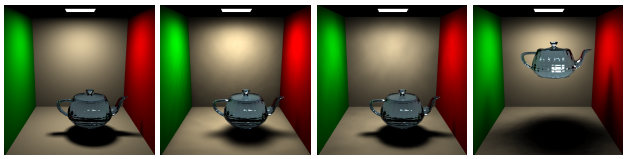
For the lightcuts method the table shows (from left to right): average number $\bar{s}$ of evaluated shadow rays per pixel sample with 2% error metric, memory used for the light clusters [11] plus the photons (VPLs), time for constructing the raytracing kd-tree and clustering VPLs, rendering time, time for the VPL sampling (same as in our method), and total time.

The GLOSSY BOX scene contains a glass sphere and a glossy icosahedron and is rendered with global illumination. Since lightcuts is not able to generate caustics, we rendered the scene with *path tracing* [7] using 2500 samples per pixel. In Fig. 5a-c the illumination in the office scene is decomposed into the direct, indirect, and global illumination. The reference is shown in Fig. 5d and was computed using lightcuts with $\bar{s} = 261$. The darkening in the corners of the lightcut image is due to the automatic clamping of too high contributions of indirect VPLs.

In Fig. 4 we show that our method is also capable of producing high quality direct lighting with sharp and soft shadows using a small number of photons. Reference image (a) is generated by sampling the light source with 100 shadow rays per pixel, (b) is computed using the method in [5] with $80,000$ photons (i.e. ray splatting without explicit visibility), and (c) and (d) are rendered with our method using $10,000$ photons.

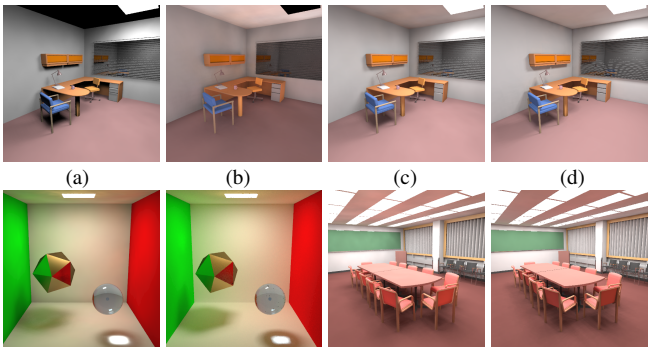
7 Conclusion and Future Work

We proposed an efficient global illumination algorithm based on density estimation that produces good quality images with a small number of photons. In contrast to [5] our algorithm is less sensitive to the bandwidth selection due to explicit visibility evaluation. However, because the costs for processing a single photon are relatively high compared to [6, 5], the computation of fine illumination details (e.g. caustics), which require a high photon density, becomes computationally more expensive.



(a) 14.7 sec (b) 22.5 sec (c) 6.1 sec (d) 9.2 sec

Figure 4. The teapot scene with direct illumination.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 5. 1. row: office scene with (a) direct, (b) indirect (+1 f-stop), (c) GI with ray-splats, (d) lightcuts [11]; 2. row: glossy box, (e) ray-splats, (f) path tracing; conference with GI, (g) ray-splats, (h) lightcuts.

References

[1] Z. Dong, W. Chen, H. Bao, H. Zhang, and Q. Peng. Real-time voxelization for complex polygonal models. *Pacific Graphics*, pages 43–50, 2004.

[2] Elmar Eisemann and X. Décoret. Fast scene voxelization and applications. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 71–78, 2006.

[3] D. Haumont and N. Warze. Complete polygonal scene voxelization. 2002.

[4] V. Havran, J. Bittner, R. Herzog, and H.-P. Seidel. Ray maps for global illumination. In *Rendering Techniques 2005*, pages 43–54, 2005.

[5] R. Herzog, V. Havran, K. Myszkowski, S. Kinuwaki, and H.-P. Seidel. Global Illumination using Photon Ray Splatting, to appear in Eurographics. 2007.

[6] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. AK, Peters, 2001.

[7] James T. Kajiya. The rendering equation. In *Proceedings of ACM SIGGRAPH '86*, pages 143–150. ACM Press, 1986.

[8] Alexander Keller. Instant radiosity. In *Proceedings of ACM SIGGRAPH*, pages 49–56, 1997.

[9] R. Schregle. Bias compensation for photon maps. *Computer Graphics Forum*, pages 729–742, 2003.

[10] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapmann and Hall, London, 1985.

[11] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. Greenberg. Lightcuts: A scalable approach to illumination. In *Proceedings of ACM SIGGRAPH*, pages 1098 – 1107, 2005.