Fast Final Gathering via Reverse Photon Mapping

EUROGRAPHICS 2005 / J. Marks and M. Alexa
(Guest Editors)

Volume 24 (2005), Number 3

Fast Final Gathering via Reverse Photon Mapping

Vlastimil Havran Robert Herzog Hans-Peter Seidel

MPI Informatik, Saarbrücken, Germany

Abstract

We present a new algorithm for computing indirect illumination based on density estimation similarly to photon mapping. We accelerate the search for final gathering by reorganizing the computation in the reverse order. We use two trees that organize spatially not only the position of photons but also the position of final gather rays. The achieved speedup is algorithmic, the performance improvement takes advantage of logarithmic complexity of searching in trees. The algorithm requires almost no user settings unlike many known acceleration techniques for photon mapping. The image quality is the same as for traditional photon mapping with final gathering, since the algorithm does not approximate or interpolate. Optionally, the algorithm can be combined with other techniques such as density control and importance sampling. The algorithm creates a coherent access pattern to the main memory. This further improves on performance and also allows us to use efficient external data structures to alleviate the increased memory requirements.

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

1. Introduction

Computing indirect illumination in synthesized images robustly and efficiently is well known to be difficult. The *unbiased* algorithms (path tracing, bidirectional path tracing, Metropolis light transport, for survey see [DBB03]) suffer from high-frequency noise in the images and the required computation times exceed typically far the time limits given by a user. For this reason *biased* algorithms are more popular in practice, including the production rendering [TL04, CB04].

A popular algorithm computing the indirect illumination is photon mapping [Jen96, Jen01]. It reproduces many global illumination effects, but computes various contributions in different ways in order to improve the efficiency of the whole computation. The direct illumination is computed by common means sampling the light sources [SWZ96]. In this paper, we deal only with computation of indirect diffuse illumination as it is performed by global photon mapping combined with final gathering.

Photon mapping decomposes the indirect illumination computation into diffuse and caustics parts, that are computed separately by special algorithms. First, the photons are shot from the light sources into the scene and they are classified as *caustics photons* and *global photons*. Second, an efficient search data structure such as a kd-tree is constructed for the *global photon map* and the *caustic photon map*. Third, primary rays are shot from the camera to the scene. At hit points of the primary rays with the scene the illumination is computed by density estimation using k-nearest neighbor (kNN) search in the caustics photon map. Fourth, for a single hit on the object with a diffuse or glossy surface hundreds to thousands of *final gather rays* (FGRs) are shot. For each FGR the indirect illumination is computed by density estimation using kNN search in the global photon map.

Our main concern in this paper is the computation of indirect illumination on diffuse and moderately glossy surfaces using final gathering with global photon maps. This is one of the methods used in the production rendering [KNE*02]. The computation consists of repeatedly computing the density estimation given the point in the object space and estimating the radiance along FGRs. This is the most consuming part of the whole rendering. In this paper, we show how to speed up final gathering in the context of photon mapping. We reverse the whole computation of density estimation. We store all FGRs and construct the search data structure over the hit points of FGRs with the scene. Then we distribute the energy from photons to stored FGRs. A speedup can be achieved thanks to the logarithmic complexity of the searching. Furthermore, we reorganize the computation in a coher-

ent way, which allows us caching of only a small part of the data in the main memory, while the whole data is stored on a disk. This makes main memory requirements for our algorithm acceptable.

Originally, the photon mapping algorithm was designed to run on a computer with only 32 MBytes of RAM [Jen96]. For this reason the data structures for photons were highly optimized for storage. Nowadays, 32-bit computers with 1 or 2 GBytes of main memory are commonly used. The main memory chips get cheaper over the years and the bottleneck between the main memory and CPU becomes even more important. This gave origin to the cache-aware and cache-oblivious algorithms and data structures [Dem02]. Our algorithm is in fact a cache-aware algorithm. By reversing the order of density estimation we achieve an algorithmic speedup for searching by factor 2–3 without any decrease of image quality.

In the following section we discuss briefly the related work. In sections 3–6 we present our algorithm with the description of the data structures, the time complexity analysis, the reverse density estimation, and several algorithmic improvements. In Section 7 we present the results. Finally, Section 8 concludes the paper with some prospects for future work.

2. Related Work

The time consumed by diffuse indirect illumination computation is typically a significant part in the whole rendering. This issue has been addressed by several papers. In this paper, we focus on the method that aims at high quality solution of the indirect illumination, not allowing any artifacts (due to some approximation in the algorithm). We exclude from discussion methods based on radiosity, such as [SSS02], as well as other techniques that explicitly work over polygonal representations [Mys97], or severely approximative algorithms prone to visible artifacts [LC04]. The recent global illumination real-time oriented algorithms such as Instant Radiosity [Kel97] and its followers [WBS02] work with sufficient quality only for diffuse surfaces (radiosity) and are not suitable for high-quality rendering taking into account general light transport.

The irradiance cache [WH92] interpolates indirect illumination over diffuse surfaces. The solution works efficiently for slowly changing indirect illumination. The indirect illumination is computed only sparsely across the image plane and the irradiance is stored in an octree. For other rays is possibly interpolated. Usually, the irradiance caching is acceptable and it accelerates by one order of magnitude. However, it can introduce visible artifacts, in dependence on the scene and user settings.

Christensen [Chr99] proposes to precompute the irradiance at the positions of photons for one quarter of all photons. While the storage space required for photon maps is increased, the irradiance for FGRs is approximated by a piece-wise constant function. Instead of the density estimation based on kNN search the closest photon with the normal similar to the one at the FGR hit is found and used. The technique offers the speedup from 25% to factors of 500–800%, if the number of final gather rays is 10-100 times larger than the number of photons. However, the problem is the approximative nature of the algorithm and the dependence on the scene. In order to improve on performance while avoiding artifacts the user is required to tune user settings such as "similar enough" for photon normals. In our experience, we have also found that the speedup varies strongly.

Several other techniques are discussed by Christensen [Chr02]. He discusses these techniques: better estimate of the initial radius used for kNN search, the iterative algorithm for kNN search, and the technique described in the previous paragraph. Further on, he describes the density estimation via the convex hull estimate and combining the results from several photon maps.

Further possibilities are described in the same course notes by Kato et al. [KNE*02], for Kilauea parallel global illumination renderer based on the photon mapping. Several tricks for more efficient FGR evaluation for transparent materials are discussed. Second, the efficient technique based on irradiance caching reducing severely number of FGRs is proposed. Third, the reprojection of FGRs is described for purposes of interpolation. The authors state that the proposed techniques and their variants can result in both good speedup and also slowdown compared to simple final gathering in dependence on the user setting. More importantly, the inappropriate settings can result in inferior quality of rendered images and unacceptable artifacts. This can be corrected only by recomputing the whole image with different settings.

Christensen and Batali [CB04] proposed the concept of an irradiance atlas as approximative irradiance representation. They associate the photon map with every object and using caching the kNN search is limited to a single photon map. They propose an irradiance atlas represented by brickmaps: the precomputation of irradiance at every photon similarly to [Chr99]. It enables to render huge scenes using hundreds of millions of photons.

The SIGGRAPH'02 course notes #43 contain many details about using photon mapping in practice and possible problems and solutions. The contributors show the problems met in the industry. Further, they show that the secondary final gather is often necessary when the distance traveled by FGRs is too small in order to avoid artifacts. It is obvious from the course notes that there are many open problems for photon maps and the achieved speedups by various techniques are highly dependent on scene and user settings. The user has to rely on his or her intuition and experience for setting parameters for basic photon mapping algorithm and acceleration techniques. A simple photon mapping algorithm with a single button "Render" is needed.

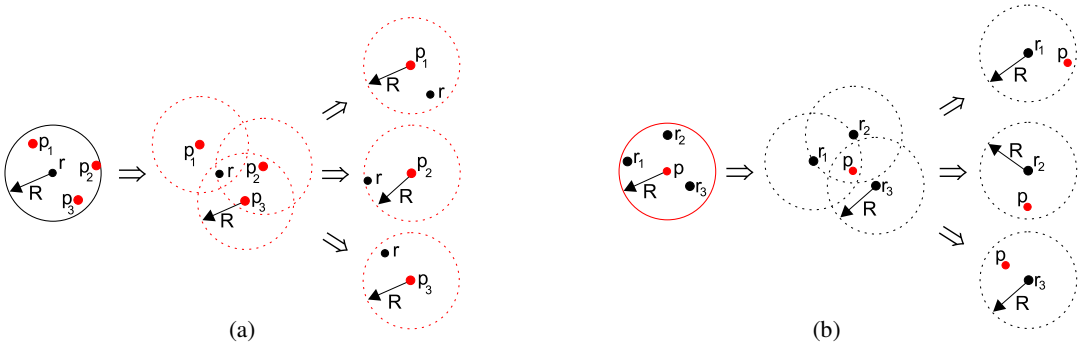

Figure 1: *Visualization of kernel density estimation (a) from the perspective of final gather rays (normal photon mapping) and (b) from the perspective of photons (reverse photon mapping). Symbol r marks reverse photons (hits of final gather rays, in black color), symbol p marks photons (in red color), and R is the radius used for the search.*

In this paper, we propose an acceleration technique that requires virtually any user settings. We compute a relatively high number of FGRs which guarantees sufficient image quality. Then we accelerate the computation for FGRs. An algorithmic speedup for searching required by the density estimation is achieved at the expense of increased memory requirements. The proposed algorithm is not a heuristic, but an algorithmically valid concept that always leads to decrease of computation time due to faster searching carried out during density estimation for a large number of queries. No additional artifacts are introduced by our algorithm.

Below we discuss some settings for final gathering as used in practice. The number of FGRs per pixel is 600–4,255 [Chr02, KNE*02, TL04]. The number of pixels in an image is typically 1–3 millions. This gives in total 10^9 to 10^{10} FGRs to be computed. It involves creation of the FGRs via BRDF importance sampling, tracing the rays through the scene, and estimating the radiance for every FGR at the hit point with the scene.

Finally, we would like to mention that density estimation as a non-parametric estimation of quantities is a well established mathematical themes. Introductory texts on the density estimation are accessible also for applied sciences [Sil86, WJ95].

3. Reverse Photon Mapping

In this section we give an algorithm overview followed by more detailed description of the algorithm.

3.1. Algorithm Overview

The basic idea of this paper is to reorganize the computation of the density estimation for diffuse indirect illumination in the reverse order. Below, we call by *normal photon mapping* the way how the global photon map is used traditionally [Jen01]. The photons are shot to the scene, the photon map is constructed. For every primary ray hitting objects in the scene FGRs are shot. For every final gather ray hitting an object the irradiance is computed by density estimation based on kNN search.

We propose a *reverse photon mapping* algorithm. We use the same set of FGRs and the same set of photons as for normal photon mapping. The idea is to compute the density estimation in reverse order. For a photon we find all the hits from FGRs and distribute the energy to corresponding pixels. The reverse order results in faster computation.

The difference between density estimation starting from FGRs (normal photon mapping) and starting from photons (reverse photon mapping) is shown in Figure 1. Below, we describe a simple version of the proposed algorithm without issues related to caching for the sake of simplicity.

3.2. Construction of Reverse Photon Maps

In the first step we shoot all the FGRs and store *reverse photons* corresponding to hits of FGRs with the scene objects in the reverse photon array A_r. The reverse photons are similar to the concept of importons [PP98]. Since we use reverse photons differently than importons, we prefer a different name to distinguish between these concepts. The reverse photon contains the following information:

- Position in 3D – 12 Bytes.
- Spectral weight – 12 Bytes for 3 spectra (XYZ or RGB).
- Incoming direction – 2 Bytes (compressed).
- Pixel position on the image – 4 Bytes.

The storage of reverse photon requires 30 Bytes, however, for better alignment in the main memory 32 Bytes are used. In addition to photons the reverse photons contain the position of the pixel in the image. This way the reverse photon can transfer the radiance to the image pixel during final gather. Next we construct the *reverse photon map*: an efficient data structure over reverse photons, which allows us to compute efficient fixed-radius or kNN search.

3.3. Efficient Search using a Kd-tree

In the second step, we construct an efficient data structure for fixed-radius search over the reverse photons. We tested different search data structures such as a balanced kd-tree in various traversal order (inorder, preorder), as well as grids and recursive grids. We evaluated as the most time and space efficient a kd-tree constructed with sliding-midpoint rule analysed theoretically [DDG00]. Unlike the balanced kd-tree proposed for photon maps [Jen01], where the interior nodes and leaves correspond to single photons, the reverse photons are only pointed to in the leaves of our kd-tree. Both the interior node and leaf of the reverse photon kd-tree take 8 Bytes. Similar approach was used by Wald et al. [WGS04]. The kd-tree is constructed over the array A_r in such a way that a single leaf can contain a contiguous sequence of photons in the array A_r. The maximum number of photons in a leaf is T_m. The maximum depth of the tree is T_h. The pseudocode for the recursive top-down kd-tree construction starting at the root node is:

- If the number of photons associated with the current node H is smaller than threshold T_m or the depth of the tree is equal to T_h, declare H a leaf. Return.
- Select the axis a – the longest side of the cell associated with the current node to be split.
- Compute a spatial median position P_H along axis a.
- Sort by swapping in $O(N)$ time photons in the array A_r so that all the photons with coordinate in axis $a < P_H$ are on the left side of A_r and the ones with $\geq P_H$ on the right side.
- If either the left or right halfspace given by splitting plane P_H in axis a does not contain any photons, move the splitting plane to the nearest photon on the right or on the left (sliding-midpoint rule).
- Create two child nodes of the current node H.
- Recurse for the left and right child of the current node H using the appropriate bounding box and subarray of A_r.
- Return.

The time complexity of this kd-tree construction is $O(N \cdot \log_2\lceil N/T_m \rceil)$.

3.4. Memory Layout for Kd-tree

The *interior* node of the kd-tree stores the splitting plane position (32 bits), a single index to its right child (28 bits), and the type of the node plus the splitting plane orientation (2+2 bits). The left child of the interior node directly follows the parent node in the memory (preorder layout).

The *leaf* of the kd-tree contains the index to the first reverse photon in the array A_r (30 bits) plus the type of the node (2 bits shared with interior nodes). Further, we store the number of the reverse photons associated with the leaf (8 bits), and the length of the diagonal of a spatial cell associated with the leaf (24 bits). The use of the diagonal length is explained in Section 6.1.

The sliding of the splitting plane, if one child does not contain any data, is a key issue for good performance of the kd-tree for non-uniform distribution of point data. Although such a rule looks quite simple, it was proved to be bounded in the worst case by a polylogarithmic time for approximate geometric search queries [DDG00, MM99].

A single leaf can reference several reverse photons. This results not only in a slight improvement of searching performance, but also in a substantial decrease of construction time and memory space. The impact of having several data entries (photons) in the leaves of a kd-tree for searching was studied by Talbert and Fisher [TF00]. They conclude that 10-30 photons does not decrease performance, but the memory requirements can be highly reduced (by up to a factor $\log_2 30$). As a result, the total memory consumed by the proposed memory layout for kd-tree is in fact smaller than for balanced kd-tree layout used by Jensen [Jen01]. The memory required by the nodes of the kd-tree is 30–60 times smaller than the memory for reverse photons. The whole tree can be stored in the main memory of a standard PC even for high number of reverse photons.

3.5. Establishment of Coherent Accesses

In addition to the reverse photon map we also construct a second kd-tree, which we call here a *kd-tree over photons*. We shoot photons from the light sources and store them in the photon array A_p if they hit a diffuse surface exactly as it is carried out for normal photon mapping. Except that we do not use it for searching. By building up the kd-tree over photons we spatially sort the photons in the photon array A_p. Queries formed by sequentially traversing the photon array A_p establish spatial and therefore memory coherent accesses to the reverse photon map.

3.6. Reverse Density Estimation

For reverse density estimation we have highly coherent queries in the reverse photon map. For every photon position y_j we find all reverse photons not farther than the radius R_j. The precomputation of radius R_j uses diagonal length of cell mentioned earlier and is described in Section 6.1. Instead of the classical kNN density estimation we use a *kernel density estimation* (KDE). KDE with adaption to the local density produces slightly better results than kNN density estimation, since it better preserves high density gradients and does not suffer from heavy tails in low density regions. The KDE methods are well studied in statistics [Sil86, WJ95].

We compute Euclidean distance between photon and reverse photon. Using an arbitrary kernel (box, cone, Gaussian, Epanechnikov, etc.), we distribute the flux after multiplication by the weight given by the kernel function.

Below we recall the rendering equation in the context of final gathering, where for every primary ray hit we shoot

N final gather rays (FGRs) using BRDF importance sampling. The radiance contributed by diffuse indirect illumination along a primary ray in the direction ω_o hitting a surface at point x with normal $\vec{n}$ is computed as:

$$L_d(\vec{\omega}_o,x) = \frac{1}{N} \cdot \sum_{i=1}^{N} L_i(x,\vec{\omega}_i) \cdot f_r(x,\vec{\omega}_i,\vec{\omega}_o) \cdot (\vec{n} \cdot \vec{\omega}_i) \quad (1)$$

For normal photon mapping the radiance along the FGR $L_i(x_i,\vec{\omega}_i)$ using density estimation with K nearest neighboring photons found at maximum distance R_i from the hit point x_i is computed as:

$$L_i(x_i,\vec{\omega}_i) = \sum_{p=1}^{K} \mathcal{K}(\frac{||x_i - y_p||}{R_i}) \cdot f_r(x_i,\vec{\omega}_p,\vec{\omega}_i) \frac{\Delta\Phi_p(y_p,\vec{\omega}_p)}{\pi \cdot R_i^2}, \quad (2)$$

where $\mathcal{K}(\ldots)$ is the kernel function used in the density estimation. The radiance along the FGR $L_i(x,\vec{\omega}_i)$ is computed using reverse photon mapping as follows:

$$L_i(x_i,\vec{\omega}_i) = \sum_{j=1}^{V} J(x_i,y_j,R_j) \cdot f_r(y_j,\vec{\omega}_j,\vec{\omega}_i) \frac{\Delta\Phi_j(y_j,\vec{\omega}_j)}{\pi \cdot R_j^2},$$

$$\text{where } J(x_i,y_j,R_j) = \begin{cases} \mathcal{K}(\frac{||x_i-y_j||}{R_j}), & ||x_i - y_j|| < R_j \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

for all photons that during the search found the reverse photon at position x_i at the maximum distance R_j from the photon position y_j. The number of all photons is V. The analogy between normal density estimation and reverse density estimation is shown in Figure 1.

4. Time Complexity Analysis

In this section we explain the source of the speedup for searching using reverse photon mapping compared to normal photon mapping. Let us denote by G the number of FGRs shot per pixel, by I the number of pixels in the image, by K the number of photons searched for normal photon mapping, and by R the radius used by reverse photon mapping for fixed-radius search. For the analysis we assume the ideal kNN search finding always K nearest photons in $O(K + \log_2 V)$ time.

4.1. Normal Photon Mapping

For normal photon mapping we shoot V photons, we construct the kd-tree in time $T^C_{NPM} = c_1 \cdot V \cdot \log_2 V$. During rendering we shoot $F = G \cdot I$ FGRs. For every FGR we find and process the K nearest photons in time $c_2 \cdot K + c_3 \cdot \log_2 V$. The total number of found photons for all FGRs is $U = F \cdot K$. The total computation time for normal photon mapping without construction time is then:

$$T^S_{NPM} = c_2 \cdot F \cdot K + c_3 \cdot F \cdot \log_2 V \quad (4)$$

The constant c_1 is a factor behind the sorting of the photons during the construction of the tree, c_2 is the factor for processing a single photon found, and c_3 is the time required for a single traversal step in the kd-tree during kNN search. The space needed for the photon mapping is $O(V)$.

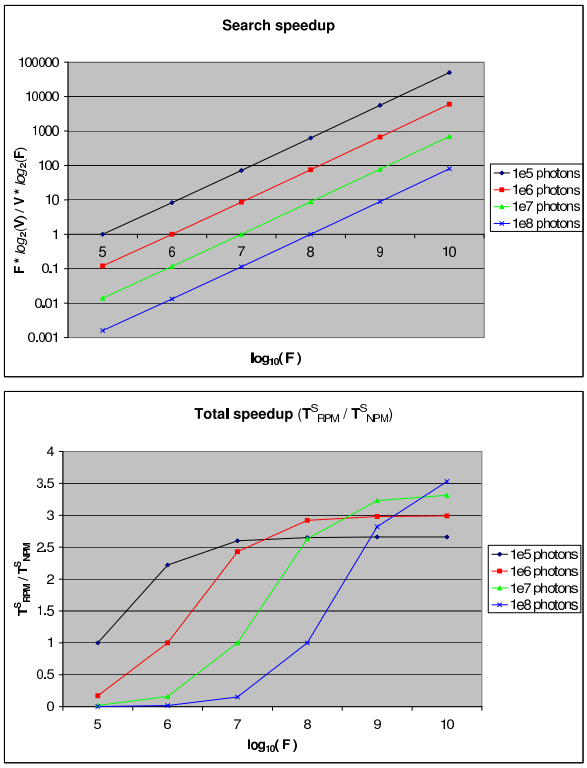

Figure 2: *The speedup between computation of the normal photon mapping and reverse photon mapping. The number of pairs photon-reverse-photon (all photons found for all FGRs in normal photon mapping) is the same. The constants are $K = 100$, $c_2 = 0.1$, and $c_3 = 1.0$. (top) Theoretical speedup only for searching. (bottom) Total speedup taking into account the necessity to process the found data.*

4.2. Reverse Photon Mapping

For reverse photon mapping we construct two kd-trees, the kd-tree over V photons and the reverse photon map kd-tree over F reverse photons. Since we increase the number of photons in a single leaf to T_m, the construction time is $T^C_{RPM} = c_1 \cdot (V \cdot \log_2 \lceil V/T_m \rceil + F \cdot \log_2 \lceil F/T_m \rceil)$. The searches are carried out in the reverse photon map. Let us assume that the total number of found pairs *photon-reverse-photon* is also U. Since the number of searches is V, the number of reverse photons found per search is on average $K_r = F \cdot K/V$. The search time in the reverse photon map for a single photon is then $c_2 \cdot K_r + c_3 \cdot \log_2 F$. The total computation time for reverse photon mapping is then:

$$T^S_{RPM} = c_2 \cdot V \cdot (F \cdot K/V) + c_3 \cdot V \cdot \log_2 F$$

$$= c_2 \cdot F \cdot K + c_3 \cdot V \cdot \log_2 F \quad (5)$$

The space for the reverse photon mapping is $O(V+F)$.

In our analysis the computation time to process all pairs *photon-reverse-photon* is the same for both normal and reverse photon maps (factor $c_2 \cdot F \cdot K$ in formulae 4 and 5). However, the searching time is significantly smaller for reverse photon mapping. The theoretical speedup excluding the factor $c_2 \cdot F \cdot K$ is shown in Figure 2 (top). The ratio of computation times T^S_{NPM}/T^S_{RPM} are shown in Figure 2 (bottom) for various number of total FGRs and several settings for number of photons V.

From the analysis it follows that the reverse photon mapping decreases the searching time only if the number of final gather rays to process is by orders of magnitude higher than the number of photons. This is a standard case for rendering images, in particular high resolution images used in the production rendering. It should be mentioned that if the computation time for processing the found pairs photon-reverse-photon $c_2 \cdot F \cdot K$ dominates the computation, the reverse photon mapping does not give any speedup.

5. Algorithm Workflow

In this section we describe the algorithm workflow along with the efficient organization of the data structures for reverse photon mapping. The algorithm decomposes the computation to tasks to deal with increased memory requirements. There are several algorithmic design goals fulfilled by our algorithm. First, the computation over the data structures results in highly coherent access in the main memory, mostly read only. Second, the write-back traffic from the CPU cache to the main memory is efficiently reduced. Third, the algorithm requirements for the main memory are acceptable. Fourth, the increased main memory requirements are alleviated by an efficient caching scheme using the disk and/or tiling of the screen.

5.1. Task Scheduler

Rendering images in high resolution and high quality requires a large number of reverse photons (i.e. FGRs) that do not fit in the physical memory of consumer-level computers. Todays, the physical memory of PCs is often limited by 1–4 GBytes. One way to solve this issue is to subdivide the execution in many tasks and combine the results of these tasks. The major part is the computation of the radiance along final gather rays. Since the computation of density estimation via reverse photon mapping does not depend on the number of FGRs or the number of pixels, we can subdivide the whole computation across the image to tiles and/or the number of FGRs per pixel.

In order to make it practical to the user, we hide the burden for setting these parameters manually. Therefore, we implemented a memory scheduler that does the job of computing the number of tiles and final number of gather rays for the tasks in advance. The input for the scheduler is the size of available memory for the renderer, the image resolution, the number of FGRs per pixel, the number of samples per pixel, and the user preferences as a trade-off between image quality and the tile size. The latter gives faster results with lower quality and vice versa.

As a result the scheduler creates the *sequence of tasks* to be computed by the renderer. The task is specified by the image tile coordinates, number of samples per pixel, number of final gather rays for a sample ray, weight for this tile (i.e. its contribution to the overall image), initial random seed, etc. The order of tasks can also allow progressive previewing: first the whole image is computed with some small number of FGRs per pixel, and then the quality is improved by computing the tiles, increasing the number of FGRs per pixel. The proposed scheduling of tasks is a natural candidate for parallelization. Note that the photons from light sources are computed only for the first task and reused for all subsequent tasks. When the tile is computed, all the computed tiles are combined to a single image. This can implement progressive previewing.

5.2. Data Workflow

Below we describe the algorithm along with the data workflow for a single task given by the scheduler. The overall scheme of processing is depicted in Figure 3: data flow view on the left and data structure view on the right. The computation has three phases: preprocessing, processing, and completion phase.

In the *preprocessing phase* photons are shot and the kd-tree over photons is constructed (only for the first task). Second, for each task given by the scheduler, the FGRs are shot to the scene and reverse photons are saved to the array A_r. Then the reverse photon map is constructed.

In the *processing phase* the density estimation is computed in the reverse order. The photons from the photon array A_p are processed sequentially, which creates a highly coherent search pattern in the reverse photon map. The energy from a processed photon is distributed to the entries in the write-back cache-array A_w. The entries of A_w map one to one to the entries of A_r. Note, that during the search the kd-tree over photons and the reverse photon map are used for read-only access. Only the write-back cache-array A_w is read to the CPU cache and written back to the main memory. This cache-aware data organization efficiently reduces the data throughput between the main memory and the CPU cache for the main part of the computation. We checked that accumulating the energy into the pixels directly produces too many cache misses, since the whole image tile does not fit to the CPU cache.

In the *completion phase* the accumulated radiance per reverse photon (FGR) is multiplied by the corresponding

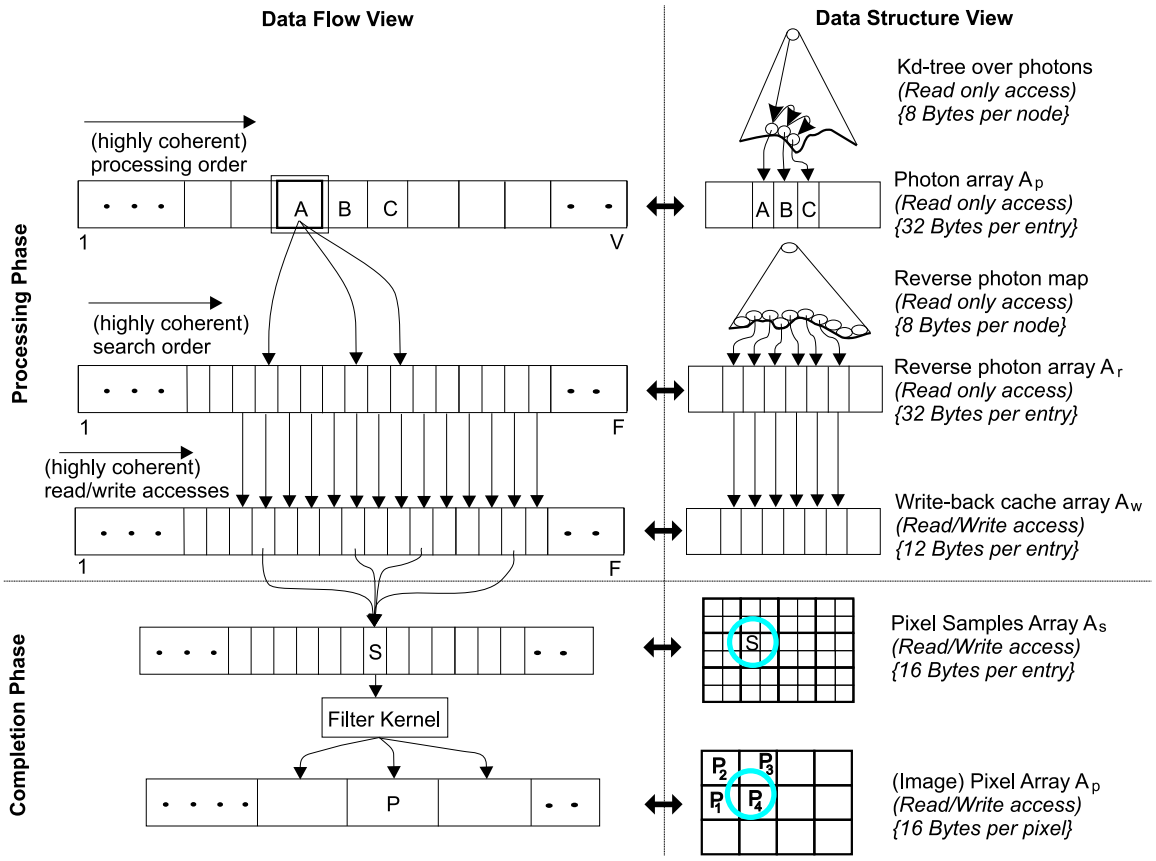

Figure 3: *Visualization of the data flow on the left and the data structures on the right. In the processing phase the density estimation for all FGRs is computed (top). In the completion stage (bottom) the contribution to the pixels is computed, possibly using a filter. The majority of the data accesses is either read only or/and memory coherent.*

weight given by BRDF importance sampling of FGRs at directly visible surfaces stored in A_r. The write-back cache-array A_w is processed in the sequential order. The radiance is summed to the radiance along pixel samples that are possibly filtered using some convolution kernel before they are added to the resulting image. The time for the completion phase is negligible.

6. Further Improvements

In this section we describe several additional improvements that we apply for reverse photon mapping. We have implemented these techniques and use them in Section 7.

6.1. Adaptive-Radius Density Estimation

Instead of using *fixed-radius* KDE we use the *adaptive-radius* KDE, where the radius is derived from the approximate density of photons similarly to normal photon mapping. Recall that we store the diagonal length of the cell associated with the leaf of the kd-tree, see Section 3.4. We use the diagonal length and the number of photons in the leaf to estimate the gather-radius over a spatial cell associated with the corresponding leaf. This method can provide us approximatively the desired number of photons when searching that region.

Since photon density for diffuse illumination is changing slowly, the number of photons in neighboring spatial cells is also varying slowly. We observed that the standard deviation for finding the kNN photons per reverse photon (FGR) is stable for different scenes and varying photon densities. The proposed way to select the radius for the density estimate of photons in this context is computed almost for free. It can also be used for normal photon mapping.

6.2. Density Control

Using the adaptive-radius is one method to reduce searching and processing time while decreasing the bias. Another adaptive technique is density control for the photon maps [SW00]. For normal photon mapping density control

is usually used in combination with importance sampling. It helps to reduce the size of the photon map in unimportant regions (i.e. regions not contributing to the image). This can be applied in reverse photon mapping. The density control takes negligible time compared to the rest of the computation. However, for reverse photon mapping it is much more decisive since the time for density estimation strongly depends on the number of photons V, see formula 5. For normal photon mapping, the number of photons V is much less important for the computation time, since it mainly depends on the number of FGRs, see formula 4.

We use a simple iterative algorithm for density control similar to the original algorithm [SW00]. We shoot many photons and organize them spatially in the kd-tree. We pick up randomly a photon in the leaves (photons are stored in the array A_p), so in $O(1)$ time. We compute the density around the photon using kNN search. If the density is higher than a user specified threshold ($photons/m^2$), we distribute its energy to neighboring photons, accounting for the photons with similar normal and incoming direction. We invalidate the photon for the next searches in the kd-tree.

Note that the density control does not interpolate on the level of incoming radiance as the irradiance caching does, since the visibility for all FGRs is computed. Then we can compute the images without artifacts that are introduced by irradiance caching interpolation.

6.3. Batched Search in the Reverse Photon Map

Another optimization we have applied is the traversal of the reverse photon map kd-tree for several photon queries simultaneously, so photons are processed *in batches*. We find the nearby photons in the leaves of kd-tree over photons (which are already close in space) and cluster them into a larger bounding sphere. During the process the bounding sphere is extended until more than a maximum number of photons are clustered inside or until the sphere exceeds the maximum radius depending on the upper threshold for the gather radius. We traverse in the reverse photon map kd-tree using this large bounding sphere to find the first interior node whose assigned splitting plane cuts the bounding sphere. This is the starting node for the kd-tree traversal for all individual queries inside the bounding sphere. This batched search avoids the traversing in upper level nodes in the kd-tree. The achieved speedup for the searching in the kd-tree depends on the photon density and the depth of the reverse photon kd-tree. We have found out the speedup for traversal between 30–65% compared to traversing of individual photons.

6.4. Ray Shooting Cache for Final Gather Rays

Ray tracing of final gather rays (FGRs) is carried out for subsequently created FGRs from BRDF sampling on visible surfaces. Since their generation is basically random (Monte Carlo) process, the directions of subsequently created FGRs are highly incoherent. According to our tests for visibility data structures, the ray tracing of highly coherent rays can be up to 3 times faster than for highly incoherent rays. We propose a single organization scheme to improve on coherency in the spirit of memory coherent ray tracing [PKGH97]. We create a single cache for FGRs by subdividing a sphere surface to C directional cells ($C = 400$). Each directional cell is associated with an array to store Q FGRs to be ray traced ($Q = 50$). We store created FGRs to the cache and when the array of a directional cell is full, we ray trace all the FGRs in the array and create reverse photons.

6.5. External Cache for Coherently Accessed Arrays

The memory requirements for the reverse photon array A_r in our algorithm can be high. In our current implementation, we allow for A_r up to $2^{30} \approx 1.073 \times 10^9$ entries. The storage space required is then $M = 2^{30} \times 32 \approx 3.43 \times 10^{10}$ Bytes, which is far from affordable size of main memory for a consumer level computers nowadays. Therefore, we take the advantage of coherent access to the arrays A_r and A_w and create an efficient caching scheme that uses external memory (a disk). Only a small portion of the arrays A_r and A_w is cached in the main memory. We use a traditional software cache implemented via hashing, splitting the array into blocks of the same size. The size of the block is $2^{20} Bytes$ and cache associativity 8. A simple hashing function uses the index of the block on the disk to keep only a small portion of the blocks in the main memory.

The array A_r is used in four steps: array creation, spatial searching, density estimation, and summing the results for FGRs to pixel samples. During creation of the FGRs, we save the reverse photons to the disk. During the reverse photon map (kd-tree) construction we process the array A_r from the first entry in the current node forwards and from the last entry backwards. This creates highly coherent access, since the size of the block is relatively large. During the main processing phase, we need to keep only a small part of the array in the main memory for every leaf node of the tree, see Section 5.2. During summation of radiance along FGRs, we access the array A_r and A_w also in sequential order.

7. Results

In this section we give numerical results for rendering using normal photon mapping and reverse photon mapping on eigth scenes of various complexity and lighting conditions, see Figure 4. We compute only the diffuse indirect lighting as carried out by normal photon mapping by the global photon map. The computed images are virtually the same for normal and reverse photon mapping.

We have implemented normal photon mapping for comparison purposes using the same data structures as for reverse photon mapping. For kNN search we used a recursive version of kNN search described in the book [Jen01].

The iterative version in [Chr02] decreases the total rendering time by 2-10%. The kd-tree constructed with sliding-midpoint rule decreases the total rendering time by 15%.

The timings for normal and reverse photon mapping are in Table 1. For reverse photon mapping we used adaptive-radius. In both cases we used density control resulting in 500,000 photons. In practice the total achieved speedup for reverse photon mapping varies between 30–230% in dependence on resolution and number of final gather rays per pixel.

Scene/N_{OBJ}	*res*	$\tilde{K}$	T^T_{NPM}	T^T_{RPM}	speedup
Cornell Box	300^2	65	605	420	1.44
(12 objs.)	500^2	55	1,540	710	2.16
Corner Room	300^2	30	1,100	860	1.27
(57 objs.)	500^2	50	2,200	1,610	1.36
MGF Office	300^2	50	941	399	2.35
(540 objs.)	500^2	60	3,600	1,125	3.20
Gallery	300^2	50	1,250	386	3.23
(8,607 objs.)	500^2	50	2,600	1,248	2.08
Sponza	300^2	50	840	493	1,70
(66,650 objs.)	500^2	50	2,613	1280	2,04
Apartment	300^2	45	810	343	2.36
(72,270 objs.)	500^2	45	2,231	926	2.40
Sibenik	300^2	50	610	460	1.32
(76,643 objs.)	500^2	50	1,710	1,250	1.36
Aizu Atrium	300^2	55	1,793	541	3.31
(948,371 objs.)	500^2	55	4,712	1,450	3.24

Table 1: *The timings for 8 scenes, using 600 final gather rays per pixel and resolution* 300×300 *and* 500×500. *The table contains the number of objects in the scene, the average number of found photons per final gather ray* $\tilde{K}$, *the overall computation time for normal photon mapping* T^T_{NPM}, *the overall computation time for reverse photon mapping* T^T_{RPM}, *and the speedup achieved for reverse photon mapping. The fastest searching technique was used in both cases.*

We tested the reverse photon mapping for various size of the available memory. The results are in Table 2. The dependence is very low. Probably due to the caching the computation for larger memory setups is even slightly slower.

Memory [MBytes]	150	300	500	750	1000	1500
# Tasks	72	30	16	12	9	6
Time [s]	1,214	1,172	1,182	1,258	1,284	1,263

Table 2: *The number of tasks and timings for scene* Sponza *in dependence of the main memory used for the reverse photon map array* A_r *and write-back cache-array* A_w. *Rendered in resolution* 500^2 *pixels and 600 final gather rays per pixel.*

We also studied the dependence on the number of photons. Due to the batched search and the adaptive-radius (the number of pairs photon-reverse-photons varies only a little), the timings differed only slightly for the range of photons 100,000-1,000,000. Therefore we believe that reverse photon mapping together with density control is an important step to a parameter-free photon mapping algorithm.

We carried out unobtrusive profiling of the code execution. In the normal photon mapping, about 70% of the time is spent on the kNN search (This is decreased to about 50% for the kd-tree described in Section 3.3.). About 15% is required for the density estimation with a box filter, and 10% is taken by visibility computations, and 3% for BRDF importance sampling. The remaining 2% are required by other operations.

In the reverse photon mapping searching (without batched search) takes about 20–25% of the total time. About 37% is required for density estimation, 20% for visibility computations, 8% for constructing the kd-trees, 6% for BRDF sampling, and the remaining 4% for other operations.

The time for searching using reverse photon mapping is efficiently accelerated by a factor of 4–7 compared to normal photon mapping. The time required for density estimation is increased although the number of evaluation for pairs photon-reverse-photon is the same. This is due to the increased rate of write-back operations from the CPU cache to the main memory (the array A_w).

If the batched search is carried out, the time required for searching is decreased by another factor of 2–3. The number of traversal steps in the reverse photon map kd-tree by batched search was decreased by additional 35%. In total the time for searching using batched search is up to 25 times smaller than the time for searching in normal photon mapping [Jen01].

The speedup for ray tracing FGRs achieved by the ray shooting cache for FGRs for 400 directional cells and 50 array entries for one cell varied between 45–60%. Note that number of pairs photon-reverse-photon was the same for normal and reverse photon mapping and the kd-tree used for the reverse photons was adapted to normal photon mapping for the sake of fair comparison.

We computed an image from the Sponza scene in a resolution of 2 Mpixels with 600 FGRs per pixel (1.2×10^9 final gather rays). Such setting is used in the production rendering [TL04]. Although the timings is 160 minutes (more than 70 minutes for both direct and indirect illumination by similar scene complexity in [TL04]), we compute more bounces of indirect illumination. The timings are also quite comparable to the rendering with iradiance atlas [CB04] ($\approx$ 200 minutes for 73×10^6 final gather rays for large-scale scene), although the irradiance atlas is clearly more approximative than our algorithm. We believe that the reverse photon mapping with efficient ray tracing in average $O(\log N)$ time could be acceptable for production rendering in close future.

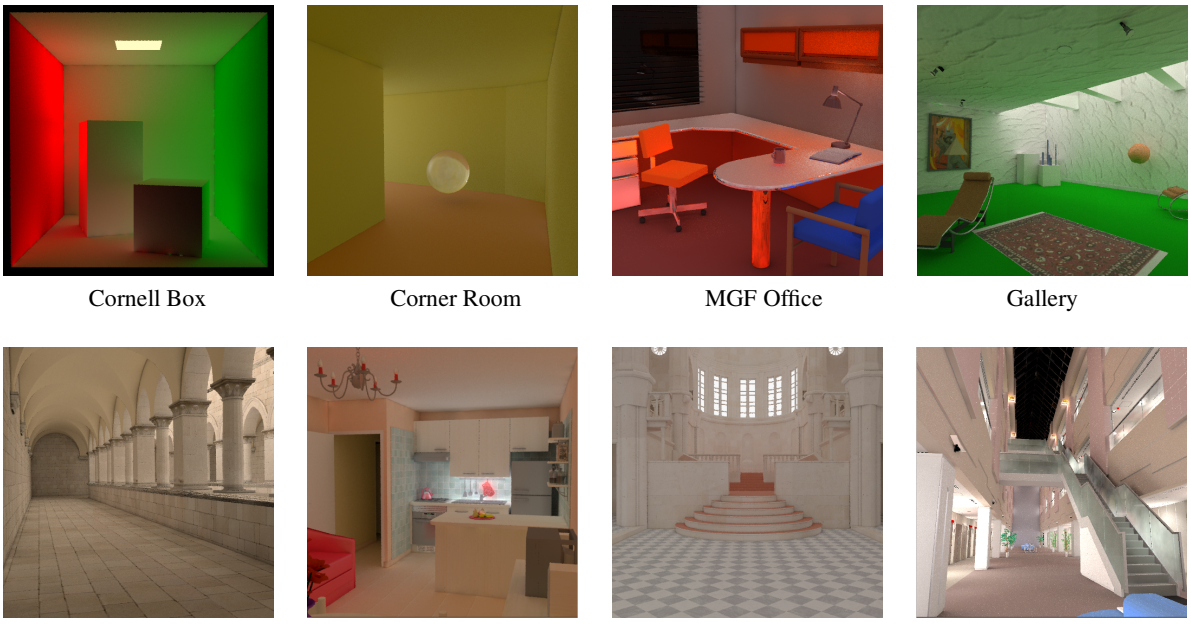

Figure 4: *Visualization for 8 scenes used for testing. The diffuse indirect illumination rendered via reverse photon mapping is shown. The tone mapping was applied to make the indirect illumination more distinct. The scene Apartment is copyrighted by Laurence Boissieux © INRIA 2005. The scenes Sponza and Sibenik were modeled by Marko Dabrovic.*

For measurements we have used a standard PC with a single CPU 3.0 GHz Intel P4 with 512 KB of L2 cache and 2 GBytes of main memory. The program was implemented in C++ and compiled with GNU g++-3.3 with -O3 optimization.

8. Conclusion

In this paper, we have proposed a reverse photon mapping algorithm for diffuse indirect illumination computed with final gathering. We have reformulated the density estimation for many final gather rays as *efficient sorting and highly coherent searching* while exploiting two kd-trees. By reordering the computation, the computation time required for the search for normal photon mapping [Jen01] is reduced by more than an order of magnitude. The algorithmic speedup of the reverse photon mapping comes from the logarithmic factor of searching using trees and highly coherent access pattern to the data. Although our algorithm requires more memory than normal photon mapping, thanks to coherent access to the data we cache only a small portion of necessary data in the main memory. This makes our algorithm practical, even on low-level consumer PCs. In practice, we have received the speedup from 2 to 3 for presented scenes.

We showed several improvements and combinations for the proposed reverse photon mapping with other techniques such as density control. The reverse photon mapping can be applied for efficient density estimation in various global illumination algorithms if the number of final gather rays is high and if we can evaluate radiances for a large set of final gather rays in arbitrary order.

Last but not least, we discussed several novel techniques. This is the kd-tree with low memory footprint constructed with the sliding-midpoint rule, estimate of the adaptive radius for KDE during the kd-tree construction, and the cache for coherent ray tracing of final gather rays. The proposed techniques can also be adapted easily to normal photon mapping.

The reverse photon mapping takes the advantage of storing more information about global illumination to improve and accelerate the computation. Therefore, the combination with other techniques such as progressiveness and quality control via adaptive sampling during rendering, irradiance caching [WRC88, WH92], and importance sampling for photons[PP98] appears to be relatively straightforward, although we do not discuss it in detail here due to the lack of space. We believe that there is space left for future work on global illumination algorithms that process global illumination information also in a coherent way. Further, we think that the algorithm can be extended for rendering animations in the spirit of recent techniques [TMD*04], which could bring additional speedup by up to one order of magnitude.

Acknowledgment

We would like to thank Jaroslav Křivánek for his timely comments on the previous version of the paper and also to all anonymous reviewers. This work was partially supported by

the European Union within the scope of project IST-2001-34744, "Realtime Visualization of Complex Reflectance Behaviour in Virtual Prototyping" (RealReflect).

References

[CB04] CHRISTENSEN P. H., BATALI D.: An irradiance atlas for global illumination in complex production scenes. In *Proceedings of Eurographics Symposium on Rendering 2004* (June 2004), Keller A., Jensen H. W., (Eds.), pp. 133–141. 2, 3, 10

[Chr99] CHRISTENSEN P. H.: Faster photon map global illumination. *Journal of Graphics Tools 4*, 3 (1999), 1–10. 3

[Chr02] CHRISTENSEN P.: Photon Mapping Tricks. SIGGRAPH'02 Course #43, 2002. 3, 4, 10

[DBB03] DUTRE P., BEKAERT P., BALA K.: *Advanced Global Illumination*. AK Peters Limited, 2003. 2

[DDG00] DICKEERSON M., DUNCAN C. A., GOODRICH M. T.: K-d trees are better when cut on the longest side. In *ESA '00: Proceedings of the 8th Annual European Symposium on Algorithms* (2000), Springer-Verlag, pp. 179–190. 5

[Dem02] DEMAINE E. D.: Cache-Oblivious Algorithms and Data Structures. In *Lecture Notes from the EEF Summer School on Massive Data Sets*, Lecture Notes in Computer Science. BRICS, University of Aarhus, Denmark, June 27–July 1 2002, pp. 1–29. 3

[Jen96] JENSEN H. W.: Global Illumination Using Photon Maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)* (1996), Springer-Verlag/Wien, pp. 21–30. 2, 3

[Jen01] JENSEN H. W.: *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Natick, MA, 2001. 2, 4, 5, 9, 10, 11

[Kel97] KELLER A.: Instant radiosity. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)* (1997), vol. 31, pp. 49–56. 3

[KNE*02] KATO T., NISHIUMURA H., ENDO T., MARUYAMA T., SAITO J., ADACHI M.: Photon Mapping in Kilauea. SIGGRAPH'02 Course #43, 2002. 2, 3, 4

[LC04] LARSEN B. D., CHRISTENSEN N. J.: Simulating photon mapping for real-time applications. In *Proceedings of Eurographics Symposium on Rendering 2004* (June 2004), Keller A., Jensen H. W., (Eds.), pp. 123–131. 3

[MM99] MANEEWONGVATANA S., MOUNT D. M.: It's okay to be skinny, if your friends are fat. In *Proceedings of the 4th Annual CGC Workshop on Computational Geometry* (1999). 5

[Mys97] MYSZKOWSKI K.: Lighting reconstruction using fast and adaptive density estimation techniques. In *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)* (1997), Dorsey J., Slusallek P., (Eds.), Springer Wien, pp. 251–262. ISBN 3-211-83001-4. 3

[PKGH97] PHARR M., KOLB C., GERSHBEIN R., HANRAHAN P.: Rendering complex scenes with memory-coherent ray tracing. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), Whitted T., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 101–108. ISBN 0-89791-896-7. 9

[PP98] PETER I., PIETREK G.: Importance driven construction of photon maps. In *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop '98)* (1998), Drettakis G., Max N., (Eds.), Springer Wien, pp. 269–280. 4, 11

[Sil86] SILVERMAN B. W.: *Density Estimation for Statistics and Data Analysis*. London: Chapman and Hall, 1986. 4, 5

[SSS02] SCHEEL A., STAMMINGER M., SEIDEL H.-P.: Grid based final gather for radiosity in complex environments. *Computer Graphics Forum (Proceedings of Eurographics 2002) 21*, 3 (September 2002). 3

[SW00] SUYKENS F., WILLEMS Y. D.: Density control for photon maps. In *Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering)* (2000), Peroche B., Rushmeier H., (Eds.), Springer Wien, pp. 23–34. 8, 9

[SWZ96] SHIRLEY P., WANG C., ZIMMERMAN K.: Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics 15*, 1 (Jan. 1996), 1–36. 2

[TF00] TALBERT D. A., FISHER D.: An empirical analysis of techniques for constructing and searching k-dimensional trees. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (2000), ACM Press, pp. 26–33. 5

[TL04] TABELLION E., LAMORLETTE A.: An approximate global illumination system for computer generated films. *ACM Trans. Graph. 23*, 3 (2004), 469–476. 2, 4, 10

[TMD*04] TAWARA T., MYSZKOWSKI K., DMITRIEV K., HAVRAN V., DAMEZ C., SEIDEL H.-P.: Exploiting temporal coherence in global illumination. In *SCCG '04: Proceedings of the 20th spring conference on Computer graphics* (2004), ACM Press, pp. 23–33. 11

[WBS02] WALD I., BENTHIN C., SLUSALLEK P.: Interactive global illumination using fast ray tracing. In *Rendering Techniques 2002 (Proceedings of the Thirteenth Eurographics Workshop on Rendering)* (June 2002). 3

[WGS04] WALD I., GÜNTHER J., SLUSALLEK P.: Balancing Considered Harmful – Faster Photon Mapping using the Voxel Volume Heuristic. vol. 22. (Proceedings of Eurographics). 5

[WH92] WARD G. J., HECKBERT P.: Irradiance Gradients. In *Third Eurographics Workshop on Rendering* (Bristol, UK, May 1992), pp. 85–98. 3, 11

[WJ95] WAND M., JONES M.: *Kernel Smoothing*. London: Chapman and Hall., 1995. 4, 5

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A Ray Tracing Solution for Diffuse Interreflection. In *Computer Graphics (ACM SIGGRAPH '88 Proceedings)* (August 1988), vol. 22, pp. 85–92. 11