Ray Maps for Global Illumination

Eurographics Symposium on Rendering (2005)
Kavita Bala, Philip Dutré (Editors)

Ray Maps for Global Illumination

Vlastimil Havran[1] Jiří Bittner[2] Robert Herzog[1] Hans-Peter Seidel[1]

[1] MPI Informatik, Saarbrücken, Germany
[2] Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria

Abstract

We describe a novel data structure for representing light transport called ray map. The ray map extends the concept of photon maps: it stores not only photon impacts but the whole photon paths. We demonstrate the utility of ray maps for global illumination by eliminating boundary bias and reducing topological bias of density estimation in global illumination. Thanks to the elimination of boundary bias we could use ray maps for fast direct visualization with the image quality being close to that obtained by the expensive final gathering step. We describe in detail our implementation of the ray map using a lazily constructed kD-tree. We also present several optimizations bringing the ray map query performance close to the performance of the photon map.

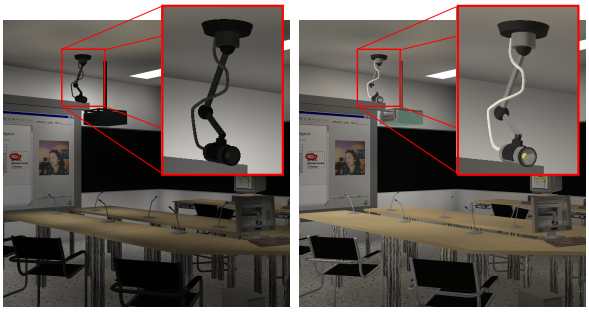
Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Most of the current global illumination techniques simulate light propagation by tracing a large number of rays through the scene. Methods like distribution ray tracing or path tracing construct light paths and evaluate their contribution to the resulting image without storing any auxiliary information. This has an advantage of generality and unbiasedness, but achieving visually pleasing results requires shooting a huge number of rays that often follow the same paths or paths with minor contributions.

One way to tackle this problem is to store some of the computed illumination and use it for generating the image. This idea is exploited in different forms by several approaches, like the photon mapping [Jen96], the light volume [CZS96], the irradiance cache [WRC88], the irradiance volume [GSHG98], or the light field [LH96]. From these algorithms only the photon map stores independent illumination samples. It neither implies any structuring of these samples, nor it restricts the type of illumination stored.

Despite its generality the photon map has one major restriction: the light flux carried by a photon can only be found in proximity of the photon impact. Even if we could store the photon path with the photon, it does not allow us to find paths in proximity of an arbitrary point in space. The well known consequence of this property is the substantial error of the estimate on the boundary of objects, referred to as *boundary bias* (Figure 1). The idea of extending the photon map concept by storing rays was first presented by Lastra et al. [LURM02]. They organize rays in a *ray cache* and use the ray cache for boundary bias elimination. In this paper we build on this idea and define the concept of *ray map* — a data



(a) (b)

Figure 1: *The direct visualization with (a) photon maps and (b) ray maps. Notice the boundary bias removal on the lamp for ray maps.*

structure for storing the rays and evaluating ray proximity queries. The ray map queries open new possibilities for retrieving the illumination information for arbitrary points and directions in space. We demonstrate this by reducing several bias sources of density estimation.

This paper aims at three main contributions: (1) It formalizes the ray map concept by enumerating ray proximity queries required by global illumination algorithms. (2) It presents a case study of density estimation using ray maps versus photon maps. (3) It describes an efficient implementation of the ray map which provides more than an order of magnitude speedup for density estimation compared to the ray cache [LURM02].

2. Related Work

Our work is closely related to the photon map [Jen96,Jen01] that stores light flux on the object surfaces. Each point in the photon map represents a photon carrying certain energy. The illumination on the object surfaces can then be reconstructed from photon hits using density estimation techniques [Sil86, WJ95] introduced to computer graphics by Heckbert [Hec90]. Note that using a finite number of observations in density estimation always leads to a systematic error referred to as *bias*.

To decrease the bias in the context of photon mapping a number of techniques have been proposed. Hey and Purgathofer dealt with the boundary bias using the average computed from several oriented photon maps [HP01]. Lavignotte and Paulin extend the object boundaries for storing of photons in polygonal scenes [LP02]. Other photon map optimization techniques such as the density control of photons were presented by Suykens and Willems [SW00] and Peter and Pietrek [PP98]. The use of convex hull to decrease boundary bias is described in [Jen01, Jen02]. However, the computation of the convex hull leads to overestimation and decreases the performance. More importantly, the convex hull does not help in cases of density estimation on small or elongated objects, where the number of photons stored on surfaces is too low. More efficient data structures for photon search were discussed by Wald et al. [WGS04]. An extension of photon maps to account for the temporal domain was introduced by Cammarano and Jensen [CJ02].

There are numerous other data structures for storing illumination such as irradiance cache [WH92], light maps [WTP00], line-space hierarchy [DS97], light vectors [ZSP98], or irradiance volume [GSHG98]. These methods aim either at representing reconstructed illumination function (light volume), regularly sampled illumination (light field), or a particular type of illumination (irradiance cache, irradiance volume). The underlying data structures are tightly coupled with the particular illumination reconstruction algorithm. The data structures represent illumination either approximately or with a systematic error: for example the irradiance cache turns a bundle of rays intersecting the given cell into a single scalar, which is used further on for interpolation.

This paper deals with an efficient way for organizing rays in space which is a subject that penetrates into the field of computational geometry. Although there are theoretical analyses of complexity of maintaining lines or rays in space [Pel04] we are not aware of a practical and efficient data structure maintaining a large number of rays (i.e. line segments) that would support efficient proximity queries and dynamic insertion and removal.

The usage of photon paths for density estimation was first proposed by Lastra et al. [LURM02]. Their method locates rays intersecting a disc on a tangent plane in order to eliminate boundary bias inherent to photon maps. This advantage is however penalized by the two orders of magnitude increase of computational time compared with the photon maps.

3. Ray Map Overview

In this section we present an overview of the ray map data structure by first discussing the way the ray map represents light paths and then enumerating the desired proximity queries.

3.1. Representation of Light Paths

The ray map stores information about light paths traced during a global illumination simulation. Unlike the other data structures storing directionally dependent illumination (light field, light volume) the ray map does not aim to reconstruct the illumination per se, but only to provide an efficient tool for such a reconstruction. Hence the ray map is independent of the algorithm used to sample illumination as well as the algorithm used to reconstruct the illumination. Additionally, the ray map poses no restriction on the structure of the samples.

The ray map *organizes photon paths* by indexing the rays which form the path. The ray indexing then allows to determine photons passing in the proximity of an arbitrary point in space or all photons passing near the boundary of an object. These photons can be found independently of the distance of their actual impacts on surfaces. As we will show later, it has several advantages and enables reduction of different types of bias.

3.2. Ray Proximity Queries

We distinguish between two different classes of ray proximity queries: *intersection queries* and *nearest neighbor queries*. The intersection queries determine all rays intersecting a given spatial domain of fixed size. The nearest

neighbor queries find K nearest rays using a particular distance metric. The queries can use different spatial domains and distance metrics (see Figure 2):

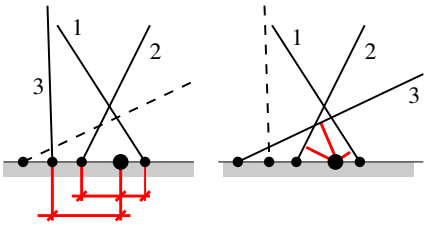


Figure 2: *Three nearest neighbors according to different ray distance metrics. (left) Euclidean distance of intersection of the ray with a tangent plane. (right) Euclidean distance of the ray itself and the center of the query.*

I. **Intersection Queries**
   Intersection domain: (a) disc, (b) hemisphere, (c) sphere, (d) axis aligned bounding box.

II. **Nearest Neighbors Queries**
   Proximity metric: (a) distance to the intersection of the ray with the tangent plane, (b) distance to the ray segment, (c) distance to the supporting line of the ray.

In practice we should be able to use combinations of queries from the two classes. That is we should determine K nearest neighbors under a condition that they intersect a given spatial domain.

3.3. Maintaining rays

The task of maintaining rays in 3D is significantly different from the task of maintaining points or polygons. The rays extend through a large portion of the scene which makes most techniques of spatial indexing [Sam89] unsuitable for their maintenance. On the other hand rays can be parameterized quite easily which suggests that they could be organized according to such a parameterization [Pel04]. Although the theoretical studies done in computational geometry follow this idea, as we will discuss later in the paper, such techniques do not seem practical for ray proximity queries used in the context of global illumination.

4. Ray Map versus Photon Map Density Estimation

In this section we describe the sources of bias in the context of photon maps. Then we describe how the density estimation is applied to the ray maps. Further, we present the case study of the density estimation that we have made for ray maps and photon maps on a set of simple scenes and ray distributions.

4.1. Sources of bias

Every density estimation technique results in a systematic error, referred to as bias [Sil86, WJ95]. In the application of density estimation for global illumination, the bias in photon maps can be classified as follows [Suy02, Sch03]:

- **Proximity bias** – given by a finite number of observations in the proximity of the evaluated point X (see Figure 3 (a)). The proximity bias leads to blurring of edges in the photon maps, since the neighborhood of X is of non-zero size. This problem can be alleviated by increasing the number of photons or by using better density estimation techniques, such as a varying kernel-width estimation [Sil86, WJ95]. Note that proximity bias is inherent to any density estimation technique.
- **Boundary bias** – a visible underestimation of illumination on the boundary of objects due to the overestimation of the surface area (see Figure 3 (b)). The darkening on the visible surfaces is well visible.
- **Topological bias** – the error due to the assumption that the surface in the neighborhood of the estimated illumination is planar (see Figure 3 (c)). The underestimation of the area for the curved surface leads to an overestimated result from the density estimation.

4.2. Ray Map Density Estimation

The ray map allowed us to design a novel density estimation technique which makes use of a combination of metrics II.(a), II.(b), and II.(c). More specifically, we use a K-nearest neighbor search which takes a maximum of the distances given by II.(a) and II.(b), i.e. the distance to the point on the tangent plane and the distance from the ray segment. Either the distance to the supporting line of the ray (II.(c)) or the distance II.(a) is then used as a weight for the density estimation kernel.

We call the resulting method a *hemisphere-disc intersection*: it considers all rays which intersect an expanding hemisphere (metric II.(b)) and which after prolongation also intersect the disc corresponding to the hemisphere (metric II.(a)). Such a combination has several advantages:

- it is consistent with the rendering equation formulated for photon maps over the disc since it normalizes the estimate over the area of the disc.
- it removes boundary bias completely since the rays passing near the boundaries of objects are also taken into account.
- it reduces topological bias, since only the rays intersecting a disc are taken into account, which is consistent with the density estimation formula.
- the distance used in the kernel metric II.(c) is invariant with respect to the rotation of the surface normal, which makes the estimate consistent on wrinkled surfaces. However, it can lead to the overestimation for strongly directional ray distributions. The metric II.(a) in this case does not lead to overestimation, but the results are more dependent on surface normal orientation. We have used the metric II.(a) below.

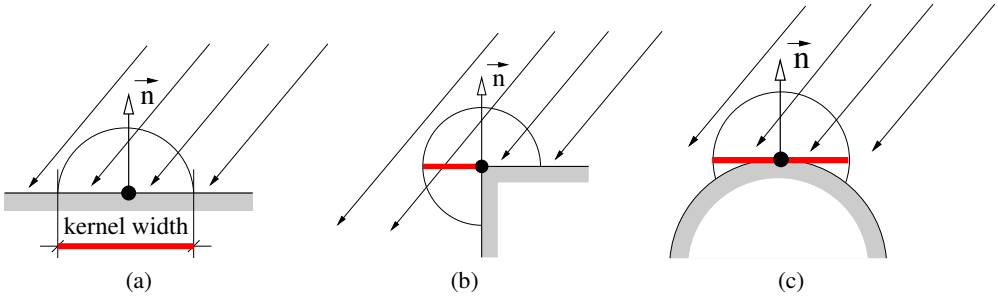


Figure 3: *Visualization of the bias sources (a) Proximity bias (b) Boundary bias (c) Topological bias.*

In the next section we document the above stated advantages of our method experimentally on three simple representative scenes.

4.3. Experimental evaluation

Using photon maps the irradiance $E(x)$ at the point X is computed as:

$$E(x) \approx \sum_{i=1}^{K} \frac{\Delta\Phi(x, dA, \omega_i, d\omega_i)}{\Delta A} \qquad (1)$$

$$\Delta A = \pi \cdot R^2$$

For Lambertian surfaces this is equivalent to density estimation. We have used the following four methods of density estimation:

- the reference irradiance computed by a reference algorithm from several magnitudes higher number of photons following the same distribution.
- the irradiance from density estimation over the photon map.
- the irradiance from density estimation using the ray map with the combination hemisphere-disc.
- the irradiance from density estimation using the ray map with the disc only [LURM02].

In order to model basic geometric features occurring in rendered scenes we constructed three simple scenes depicted in Figure 4. These scenes exhibit boundary and topological bias, which allows us to compare properties of different density estimation methods and draw conclusions about the utility of our hemisphere-disc intersection using ray maps.

Figure 4 shows three selected test scenes from the analysis together with the associated ray-photon distributions. Just a small portion of the rays is shown for aesthetic reasons.

We have used ray distributions corresponding to an area light source and parallel light source. The area light source distribution approximates indirect light of scenes consisting mainly of Lambertian surfaces, whereas the parallel one simulates highly specular illumination. The graphs in Figure 4 visualize the computed density estimation for the three tested estimators along the path depicted in the scenes. The reference estimate (black dotted line) was computed from two orders of magnitude higher number of samples (5×10^6 photons). For the actual tests we used 100,000 samples, K-nearest neighbor search with K=200, and Epanechnikov kernel [Sil86].

The two corner scenes illustrate how the boundary bias is eliminated using our hemisphere-disc method. Clearly, the estimation from photon maps either underestimates the surface area (Figure 4 (a)) by including wrong samples from the neighbor plane or overestimates the surface area on the boundary (Figure 4 (b)). The density estimation with photon maps on the wave scene (Figure 4 (c) and (d)) results in topological bias arising from the curvature of the surfaces. It also shows the problem when using only the disc intersection query [LURM02] at concave surfaces (Figure 4 (c) blue dashed line, point B and D).

5. Ray Map Implementation Using a kD-tree

In the previous section we have shown that ray maps provide advantages in illumination reconstruction compared to photon maps. In the rest of the paper we present an efficient implementation of the ray map and associated queries. We describe the algorithms for ray map construction and ray map queries. We also present a strategy that keeps the memory requirements of the method low while not causing significant performance penalty in practice.

5.1. Overview

We represent the ray map using spatial subdivision based on kD-trees. The rays are organized in a way similarly to organizing objects for ray shooting acceleration. We construct a hierarchical spatial subdivision that contains references to rays intersecting the cells of the subdivision. For each elementary cell of the subdivision we maintain a list of references to rays that intersect the cell. Then for a given query domain we first determine which cells of the subdivision it intersects and evaluate intersections only with rays referred at these cells. The resulting spatial subdivision should address the following two points during querying:

- Distribution of rays and queries

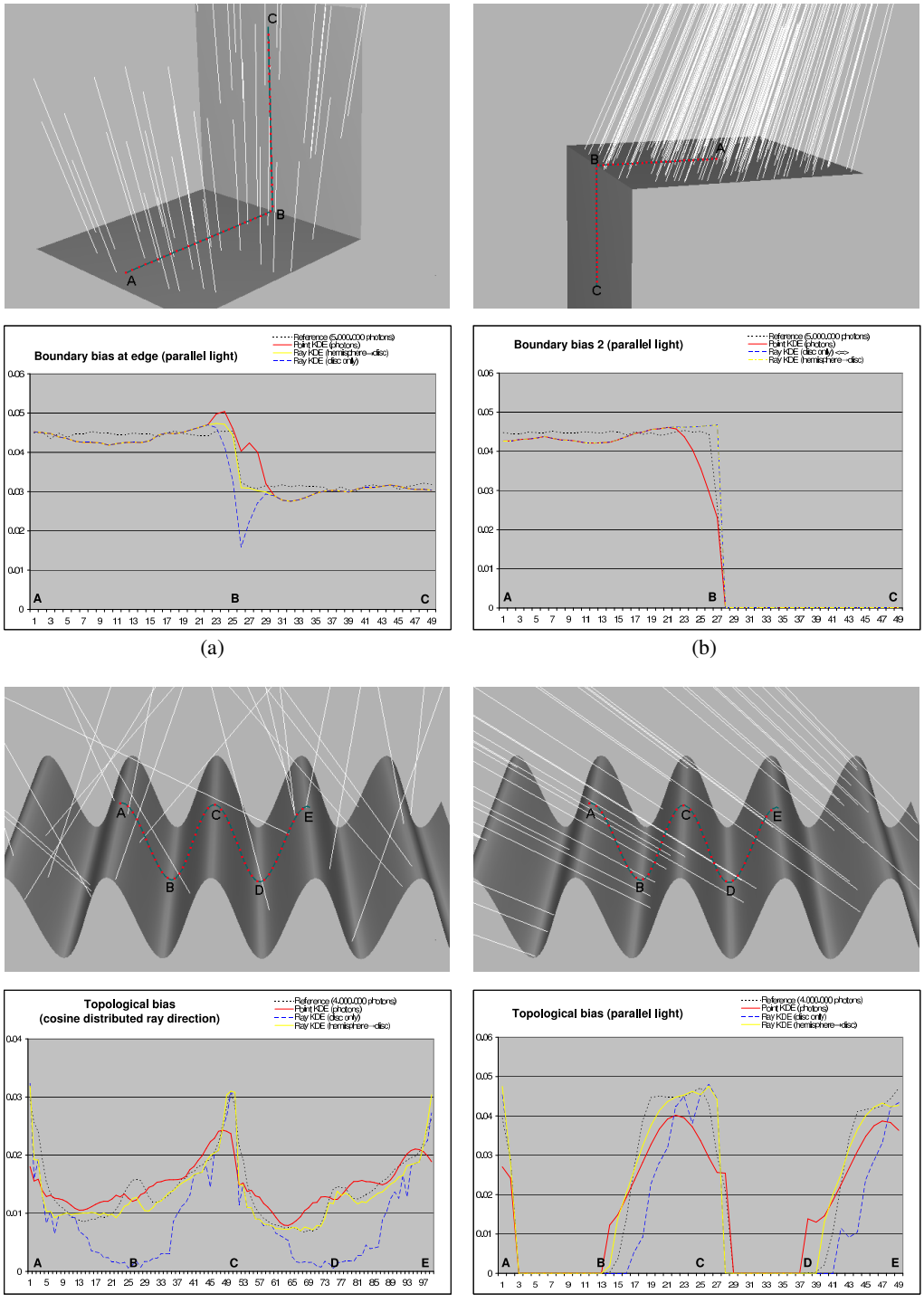


Figure 4: *Scenes used to study the bias of density estimation methods. The graphs show the resulting irradiance computed by the density estimation along the depicted path on the surfaces (marked in red-blue). The following scenes were evaluated: (a) Convex corner with a parallel distribution of rays. (b) Concave corner with a parallel distribution of rays. (c) Wave scene with a cosine distribution of rays. (d) Wave scene with a parallel distribution of rays. The black dotted line is the reference value. The red full line is computed from the photon map. The blue dashed line is computed from the ray map using a disc [LURM02]. The yellow full line is computed from the ray map using the new hemisphere-disc method.*

- Coherence of the queries

The first point is addressed by hierarchical subdivision with termination criteria based on the number of rays contained in its cells. Additionally as we describe later we use a lazy construction of the kD-tree that adapts to the distribution of the queries.

The second point is addressed as follows: once a cell with a set of rays is subdivided, the resulting ray classification (i.e. rays associated with the subdivided cells) is reused by all subsequent queries. If we access a cell that needs further subdivision, it is very likely that it already contains a limited set of rays due to the previous subdivisions. Thus if the queries are coherent, only a small number of splits is performed for each query, since we mostly access cells already subdivided by the previous queries.

5.2. Static kD-tree Construction

The static construction of the ray map kD-tree proceeds as follows: Starting at the root of the tree we check if it satisfies the subdivision criteria. If the subdivision criteria are met, we subdivide the node and distribute all rays it contains to the new leaves. A ray gets associated with a leaf only if it intersects the cell corresponding to the leaf. After the subdivision we continue by recursive traversal of the newly created children.

We have used three subdivision criteria: The node is subdivided if all of the following three conditions hold:

- The number of ray references in the node is greater than a predefined constant C_{min} (we use $C_{min} = 32$).
- The diagonal of the corresponding cell is longer than a fraction of the diagonal of the scene bounding box R_{min} (typically $R_{min} = 0.1\%$ of the size of the scene bounding box).
- The depth of the node is smaller than a predefined maximal depth D_{max} (we use $D_{max} = 30$).

We have used splitting planes positioned at the spatial median of the current node that is perpendicular to the axis with greatest spatial extent. The resulting algorithm has several desirable properties:

- Since we use a spatial median the kD-tree is spatially balanced.
- Due to the later presented lazy construction the kD-tree automatically adapts to the query distribution.
- Except for the termination criteria the subdivision is independent of the actual distribution of rays. While this might be a drawback for a static set of rays, it turns out to be a benefit for a dynamically changing ray set and the caching strategy we use.
- We do not have to evaluate a cost function which is required for more advanced splitting plane selection. In asymptotic complexity bounds this brings down the cost of the plane selection from O(n log n) (sorting according to the cost) to O(n). Additionally the constants hidden by the O-notation for the spatial median split are several times lower than for the cost based one, which is important for the on-the-fly construction.

5.3. Intersection Query

An intersection of a given spatial domain with the rays in the ray map is carried out by constrained traversal of the kD-tree and computing intersections with rays stored at the leaf nodes. The traversal is constrained only to nodes intersecting the spatial domain of the query. In fact we constrain the traversal to a bounding box of the spatial domain and use the actual domain (disc, sphere, hemisphere) only for evaluating the ray/domain intersection.

5.4. K-Nearest Neighbors Query

A K-nearest neighbors query aims to locate K nearest rays for the given query center. It uses a similar traversal as the intersection query, however it requires that the leaf nodes are processed according to their distance from the center of the query. This can be achieved by using a priority queue in which the priority of the node is inversely proportional to its distance. The approach is similar to K-nearest neighbors over the point data [AMN*98].

Initially we push the root node in the priority queue and proceed as follows: we pop the node with the highest priority from the queue. If it is an internal node, we compute minimal distances d_l, d_r of its children from the query center and insert them in the priority queue with priorities equal to $-d_l$ and $-d_r$, respectively. When reaching a leaf node we evaluate the distance of all rays associated with the node with respect to the query center and add these rays to the ray candidate list. If the ray candidate list gets larger than K, we apply the K-median algorithm to select the K rays with minimal distance. If the distance of the K-th selected ray is smaller than the distance of the unprocessed node on the priority stack, we can terminate the algorithm, since no unprocessed ray can be closer that the already found K-th ray.

The described technique considers the whole scene as a query domain. It is advantageous to constrain the query domain even for the K-nearest neighbor queries. This is easily incorporated in the algorithm by pushing only those nodes in the priority queue that intersect the query domain. This limits the number of nodes in the queue and thus provides a minor speedup. The set of leaves traversed is mostly identical to that of the unrestricted query. Note that the priority queue provides a natural adaptation of the traversed part of the scene to the range where the K-nearest rays are actually found, without the need of any complicated estimation techniques. The nodes further away from the K-nearest neighbors are accessed at the higher levels of the hierarchy,

but they do not get accessed any further. The process of K-nearest neighbor query without and with the restriction of the query domain is illustrated in Figure 5.

There may be different distance metrics used for evaluating ray distance. We can use some of the traditional techniques that take points of intersection of the ray with scene objects (photon maps) or with a tangent plane (Lastra et al. [LURM02]). Following the analysis in Section 4.3 we use the minimal distance to the intersection of the (prolongated) ray with the tangent plane (metric II.(a)).

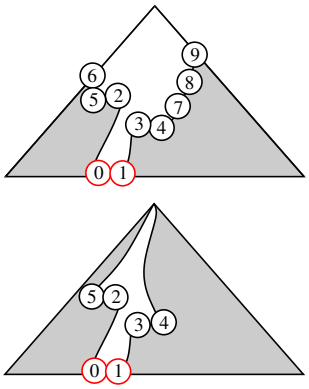


Figure 5: *K-nearest neighbor queries using a priority queue. (top) Traversal of the kD-tree for the unconstrained query. The nodes are labeled according to their processing order. The rays at the red leaves were actually tested for intersection. After testing these two nodes the query was terminated since we have found a required number of rays that were closer than any of the unprocessed nodes in the queue. (bottom) Traversal of the kD-tree constrained by the query domain. Note that although we have accessed a smaller part of the hierarchy we actually test the same number of leaves as for the unconstrained query.*

6. Ray Map Enhancements

This section presents several enhancements of the kD-tree based ray map implementation. The first three presented methods aim to improve the speed of the queries. The last method limits the size of the ray map allowing the user to tradeoff the total memory consumption for speed.

6.1. Lazy Ray Map Construction

In order to concentrate the splits in areas really accessed by the queries we use a lazy kD-tree construction. Note that a similar idea has been used by Ar et al. [AMT02] for dynamic collision detection. The kD-tree is constructed by interleaving the traversal of the already existing part of the tree with subdivision performed on its leaf nodes that satisfy the subdivision criteria (e.g. contain too many ray references).

Given a query with its domain corresponding to an axis aligned box, we start at the root node and proceed as follows:

- If the current node is an interior node, we check the position of the box with respect to the associated plane and continue the traversal recursively for the subtrees intersected by the box.
- If the current node is a leaf, we check if the subdivision criteria are met:
  - If the subdivision criteria *are met*, we subdivide the node and distribute all rays it contains to the two new leaves. Recall that a ray is associated with a leaf only if it intersects the corresponding cell. After the subdivision we continue by the traversal of the newly created children.
  - If the subdivision criteria *are not met*, we test all rays associated with the node for an intersection with the query box.

The lazy construction is visualized in Figure 6.

6.2. Directional Splits

Two important ray map queries use query domains that not only restricts the spatial range of the rays but also their directional range. In particular the disc query and the hemisphere query only consider rays with a negative dot product with the normal of the disc or the hemisphere. If we only group rays according to the spatial positions, we cannot efficiently cull groups of rays with opposite directions than those desired by the query.

To tackle this problem we extended the kD-tree by *directional nodes*. Unlike the usual kD-tree node the directional node does not provide a split in the spatial domain, but rather in the directional domain. The directional node contains a reference direction. The node subdivides the current range of directions into those having a positive and negative dot product with the reference direction. For the density estimation the negative directions are those feasible for the disc or hemisphere intersection queries since they represent incoming rays. To allow sharing of the same directional node by several queries with slightly different normals we enlarge the set of feasible directions by a specified α angle (see Figure 7). Such an α-extended directional node does not cull all infeasible rays, but allows to reuse the directional node by all queries with normals within α angle from the reference direction. The directional nodes are traversed as follows: if the angle between its reference direction and the query normal is less than α. In this case the query is *covered* by the directional node and we only traverse the feasible subtree of the directional node (and thus successfully cull all the rays stored in the infeasible subtree). If the query is not covered by the directional node we have to traverse both subtrees of the directional node.

The directional nodes are introduced based on the normal of the actual disc or hemisphere queries. In the optimal case we would place the directional nodes as high in the tree

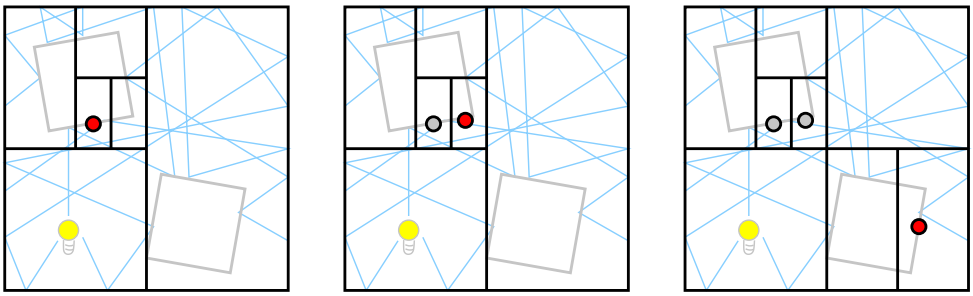


Figure 6: *Lazy construction of the kD-tree driven by the queries. (a) The first query is depicted by the red disc. (b) The second query does not require any subdivision. (c) The third query requires subdivision of the kD-tree by two new internal nodes.*

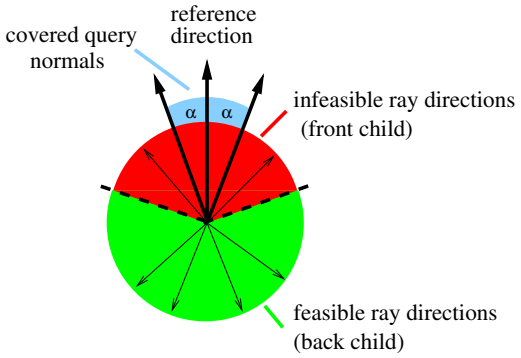


Figure 7: *The subdivision of the ray directions according to a directional node. Note that for the covered query normals only the back child of the directional node is traversed and all the infeasible rays are culled.*

as possible while making sure that the whole subtree corresponds to a spatial domain where directionally coherent queries are expected. We use a simplified strategy suitable for dynamic tree construction that uses two predefined constants: the minimal depth of a node and minimal diagonal size of the cell. If these two criteria are met for the node to be subdivided, we first check if there is no other directional node on the path to the root that covers the given query.

If we do not find such a node we introduce the directional node and split the current set of rays according to the angles between their directions and the reference direction as shown in Figure 7. The reference direction for directional node corresponds to the normal of the first query, when the insertion of directional node is decided. More complicated and efficient strategies for determining the reference direction could be used for offline processing of queries.

In scenes with directionally coherent queries, we have observed that using $\alpha = 10$ degrees, about 90% of the query normals were within the α range. This means that we could successfully cull the whole subtree maintaining rays within the remaining $180 - 2\alpha$ degrees.

6.3. Exploiting Query Coherence

Subsequent queries in the ray map are likely to be coherent, if they are induced by the direct visualization of visible surfaces for coherent pixel order on the image. We exploit query coherence by reducing repeated traversals of the same interior nodes of the kD-tree. The tests at the leaf nodes are carried out as usual since for most applications we need to evaluate the actual ray distances for each particular query.

Our design goal was to provide a mechanism that does not require preprocessing of the queries but it allows us to use the coherence between subsequent queries if there is some. We only describe a modification to the K-nearest neighbor algorithm. The modification of the intersection algorithm works similarly.

For the first query we create a list of nodes corresponding to the reached leaves and unprocessed nodes on the priority stack that are within an ε-distance from the K-th ray found. The created list becomes a reference list for the subsequent queries and the query center is a reference center. For the next query we first check if the query center lies within the ε distance from the reference center. If this is the case, we push all the nodes from the reference list to the priority queue using the actual priorities with respect to the new query center. The query then proceeds as usual, but the reference list and the reference center are not modified. If the query center is not within the ε distance from the reference center, we start the traversal at the root node and create a new reference list that will possibly be used by subsequent queries. In our tests this method provided performance gain typically ranging from 1% to 30% for $\varepsilon = 0.5\%$ of scene size.

6.4. Limiting Memory Usage

The number of rays stored in the ray map can be very large. The ray map implementation should however be able to limit the size of the indexing structure and so to balance the query performance and memory costs. As we show later we can successfully limit the memory used by the ray index to the

amount comparable to the actual ray representation with no or only minor performance decrease.

Our ray map implementation stores multiple references to a ray in several leaves of the kD-tree. As the kD-tree is constructed lazily, the overall memory consumption grows with the number of processed queries. The actual growth rate depends on the distribution of the rays (mainly their length and direction) and the spatial coverage of the queries. If the rays are very short and the queries cover only a small fraction of the scene, the memory growth is small. On the other hand, for long rays and queries evenly filling the space there can be many references to each ray in the cells of the constructed subdivision. This increases the total memory cost considerably.

We have developed a mechanism allowing to efficiently balance the memory dedicated to the ray map and the computational cost using the least-recently-used (LRU) caching strategy. We set a limit on the memory usage for representation of the kD-tree, such as 100 MBytes. Before each subdivision of a node in the kD-tree we check if the limit has not been exceeded. If it has been exceeded, we find a subtree of the kD-tree using a LRU strategy and collapse it to a single node. The collapses are performed until the desired memory bound is reached. Then a required subdivision of the node is performed.

The described method maintains parts of the kD-tree which were recently accessed. In this way we make sure that the memory usage will not exceed a predefined memory limit, while we can still exploit coherence of subsequent queries.

7. Results

In this section we summarize the results obtained using our implementation of ray maps in the context of density estimation. We compare the achieved results for the direct visualization using ray maps with the direct visualization using photon maps.

We have conducted four different tests. The first test compares K-nearest neighbor queries using our ray map implementation and the ray-cache [LURM02]. The second test illustrates the dependence of the query performance on the number of rays stored in the ray map. The third test evaluates the performance of the queries in dependence on the number of desired nearest rays. The fourth test compares the performance of density estimation from photon maps and ray maps for the direct visualization.

The *first test* is the comparison of ray maps with the method using the dynamic list of spheres. We conducted tests on four different scenes: the Cornell Box, the Cognac, the Office, and the Sala scenes (see Figure 8). We have used K-nearest neighbor queries aiming at finding 100 nearest rays. Additionally, we have restricted the search to the distance corresponding to 5% of the scene radius. For each test we have measured the total number of rays, the number of queries, the number of actually found rays, the percentage of successfully tested rays, the peak memory usage for the ray map, the number of tree collapses per query (forced by the caching scheme), and the average query time. Note that the memory usage counts the memory required for the ray index not the rays themselves. For all tests we set an upper memory limit for the ray map index to 128MB. The results are summarized in Table 1.

Rays 10^3	Found	Succ. Tests [%]	Query Time [ms]
189	100	27.0	0.16
378	100	27.7	0.22
944	100	26.0	0.42
1887	100	25.0	0.88

Table 2: *Dependence of the query performance on the number of rays stored in the ray map for the Cornell box. The results are averaged using 53,000 nearest neighbor queries.*

We can see that the ray map method provides significant speedup compared to the ray-cache. From the running statistics we have identified two main reasons: (1) the ray-cache using the dynamic list of spheres depends strongly on the choice of the appropriate search radius. If this radius is larger than that of the actual neighborhood, where the K-rays are found, we have too many candidate rays that have to be ranked. (2) If all subsequent queries are not coherent enough, the dynamic list of spheres has to be reconstructed which is relatively costly.

The *second test* shows the dependence of the query performance on the total number of rays stored in the ray map. This test was conducted for the Cornell Box using 53,000 nearest neighbor queries. The results are summarized in Table 2.

We can observe that increasing the number of rays causes only a sublinear increase of the average time per query. The increase is however more than logarithmic which is due to the fact that the average query time also includes the costs for the lazy construction of the kD-tree.

The *third test* shows the search performance in the dependence on the number of desired rays for the K-nearest neighbor query. Again we have used the Cornell Box with 1.8×10^6 rays and 53,000 queries. The results are shown in Table 3.

We see a sublinear increase of query time when increasing the number of desired rays. The more rays we require the farther we have to search from the query center. An important property of our ray map implementation is that due to the priority queue based traversal it automatically establishes the neighborhood where the desired number of rays are being found without significant performance loss.

The *fourth test* compares the rendering using the direction

Scene	Rays [10^3]	Queries [10^3]	Method	Found Rays	Succ. Tests [%]	Memory [MB]	Collapses [%]	Query Time [ms]	Speedup [-]
Cornell Box	1887	53	SP	100	0.42	23.0	-	24.90	1.0
			RM	100	25.00	128.0	0.02	0.88	28.3
Cognac	67	128	SP	78	0.50	2.4	-	3.27	1.0
			RM	78	8.30	3.7	0	0.24	13.6
Office	2550	307	SP*	100	0.90	62.0	-	3.75	1.0
			RM	100	16.90	78.0	0	0.22	17.0
Sala	2360	1310	SP	100	0.40	33.0	-	0.89	1.0
			RM	100	19.60	128.0	10^{-5}	0.20	4.5

Table 1: *Comparison of the K-nearest neighbor query performance for the kD-tree based ray map(RM) implementation and the dynamic list of spheres(SP). *For the last SP test we had to reduce the search radius to 0.5% of the scene size to obtain reasonable timings. If the initial radius was larger, there were too many rays in the candidate list leading to running times of more than two orders of magnitude greater than for the ray map method.*

Found	Succ. Tests [%]	Query Time [ms]
20	11.5	0.70
50	19.2	0.75
100	25.0	0.88
200	33.4	1.02
500	44.0	1.49

Table 3: *Dependence of the query performance on the number of desired nearest neighbors. The measurement was conducted for the Cornell Box using* 1.8×10^6 *rays and 53,000 queries.*

Scene	Time[s] ray map	Time[s] phot.map	Time[s] phot.map conv. hull	Ratio [-] phot.map / ray map
Cornell Box	311	70	116	4.4
Cognac	293	63	103	4.7
Office	195	41	71	4.7
Sala	240	115	178	2.1

Table 4: *Rendering times for density estimation with photon maps, photon maps with convex hull, and ray maps without final gathering for resolution* 1000×1000 *pixels. Only indirect illumination was computed.*

visualization with ray maps and photon maps. The time performance results for this test are summarized in Table 4, the images are shown in Figure 8 for photon maps and in Figure 9 for ray maps. Notice the clearly visible boundary bias for photon maps. For comparison purposes we have also implemented a boundary bias reduction technique for normal photon maps using convex hulls [Jen01]. The timings were increased by factor 65–75% and the images still suffered from artefacts on small surface areas, where the number of photons is insufficient. We do not show the rendered images here due to the lack of space.

The comparisons have shown that the ray map based density estimation successfully eliminates boundary bias. The overhead of performing the ray map queries is moderate – the density estimation with ray maps was 2.1 to 4.7 times slower than the direct visualization from photon maps.

For all the tests, we have set the limit for the memory usage for the ray maps as described in the Section 6.4 to 128 MBytes. For time measurements we have used a single PC with a 2.4 GHz Pentium 4 and 1 MByte of L2 cache.

8. Discussion

The testing of different variants of ray map implementations revealed several interesting features. For the discussion we chose the splitting plane selection and a comparison to a different ray map implementation that uses dual space.

8.1. Splitting Plane Selection

Inspired by the rich literature on kD-trees for ray shooting [Kap87, Hav00] we have experimented with other methods for splitting plane selection such as the ray median or query distribution heuristics. The ray median selects a splitting plane so that the number of rays in the left and right subtrees of the split node is equal. The query distribution heuristics is based on similar idea as the surface area heuristics subdivision for ray shooting. We estimate the costs of a splitting plane position by weighting the numbers of rays in left and right children with the probability that the corresponding child will be accessed by a query.

Surprisingly, the conducted experiments have shown that the best overall query performance was achieved by using the simplest strategy for the spatial median split discussed in Section 5.2. When using the more advanced ray median split or query distribution heuristics we achieved about 10%-30% worse performance per query. We explain this result as follows:

- The computational cost of the splitting plane selection for the advanced techniques is higher. Asymptotically this means $O(n \log n)$ versus $O(n)$, but there is also an additional cost for evaluating the heuristics function hidden in the O-notation. Since we construct the kD-tree lazily and the number of queries is comparable or even lower than the number of rays stored in the ray map this difference becomes significant.

- The heuristics should provide a well balanced tree with respect to the queries. However we deal with a more complicated problem than for traditional kD-trees that store only points in 3D. Any ray can be referenced in a number of leaves and it is difficult to predict how many references will occur at each subtree when performing a split near the root of the tree. This is emphasized by the fact that long diagonal rays can span across the whole scene, although after the subdivision they end up only in a few nodes in proximity of the ray.

8.2. Ray Maps in Line Space

A natural candidate for ray map representation is line space. Rays on a given line in primary space are represented by a point in line space. By representing all rays as line space points we can cluster these points and use classical range searching methods to find points that intersect the line space mapping of the query domain [Pel04]. In fact this technique was used in our first ray map implementation. We have used Plücker coordinates to map supporting lines of the rays to 6D points (points in 5D projective space embedded in 6D). Then we have clustered the resulting points using a bounding box decomposition tree (BBD-tree) [AMN*98]. The query domain (disc) was mapped to a line space convex polyhedron forming a set of 6D hyperplanes [Tel92]. We have then found all points of the BBD-tree that were contained in the polyhedron. To reflect the fact that we actually deal with line segments instead of lines we have interleaved the search with testing intersections of the query domain with the spatial bounding boxes of ray clusters.

Unfortunately, the resulting technique exhibits rather small performance gain compared to the naive implementation of the search. The major problem is performing a computationally efficient intersection of the line space mapping of the query domain (6D unbounded convex polyhedron) and the 6D bounding boxes corresponding to clusters of rays. The query domain is compact in primal space (e.g. disc), but after mapping to line space we typically obtain an unbounded thin polyhedron. The BBD-tree was always traversed almost to the bottom providing only a tenfold speedup compared to the naive implementation of the searching algorithm.

This approach however devotes further investigation: perhaps a combination of primal/dual space data structure could share the benefits of both: compactness of the query domain (primal space) and elegance of the ray representation (dual space).

9. Conclusion

We have presented a data structure for representing light transport called ray map. The ray map extends the concept of the photon map: it provides a general mechanism for storing light path samples as well as retrieving the samples using ray proximity queries. We have discussed the intersection queries, nearest neighbor queries, and their combinations. The ray map not only allows to determine rays in proximity, but it also allows to use new distance metrics unavailable for the photon map. In particular we can detect nearest rays based on the minimal distance of the ray itself from the center of the query instead of using the point of intersection of the ray with scene objects.

We have presented an efficient implementation of the ray map based on a kD-tree. We proposed a number of techniques to achieve searching performance approaching the performance of the photon map: the kD-tree is constructed lazily based on the actual queries, it is extended by directional nodes for efficient culling of infeasible rays, and we exploit query coherence by avoiding repeated traversal of the upper part of the tree. We also described a method limiting the memory usage by caching only the part of the tree that was recently accessed.

We have used the ray maps in density estimation for direct visualization. We have shown that by using our hemisphere-disc method we can avoid boundary bias inherent to photon maps and also reduce the topological bias. These results were achieved at the cost of moderate increase of computational time compared to photon maps.

The ray map concept opens a number of topics for future work. We work on methods detecting or reducing occlusion bias via simple statistical means over the rays in query. We also envision an efficient hybrid rendering algorithm combining seamlessly the results for the indirect illumination computed by density estimation from photon maps, from ray maps, and from the final gathering across the image plane. The ray maps are a good candidate for rendering of volumetric effects. Finally, we want to use ray maps in the context of animation in order to detect and update only the modified light paths.

Acknowledgment

We would like to thank Jaroslav Křivánek for providing us with implementation of photon maps that we used for direct visualization of photon maps. Further, we would like to thank Karol Myszkowski and also to all anonymous reviewers for their comments on the previous version of the paper. 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), the GameTools IST project no. IST-2-004363, and the Kontakt OE/CZ grant no. 2004-20.

References

[AMN*98] ARYA S., MOUNT D. M., NETANYAHU N. S., SILVERMAN R., WU A. Y.: An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Journal of the ACM 45*, 6 (1998), 891–923.

[AMT02] AR S., MONTAG G., TAL A.: Deferred, Self-

Organizing BSP Trees. *Computer Graphics Journal (Eurographics '02) 21*, 3 (2002), C269–C278.

[CJ02] CAMMARANO M., JENSEN H. W.: Time Dependent Photon Mapping. In *Rendering Techniques 2002* (June 2002), pp. 135–144.

[CZS96] CHIU K., ZIMMERMANN K., SHIRLEY P.: The Light Volume: An Aid to Rendering Complex Environments. In *Rendering Techniques '96* (1996), Pueyo X., Schröder P., (Eds.), Eurographics, Springer-Verlag Wien New York, pp. 1–10.

[DS97] DRETTAKIS G., SILLION F. X.: Interactive Update of Global Illumination Using a Line-Space Hierarchy. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)* (1997), vol. 31, pp. 57–64.

[GSHG98] GREGER G., SHIRLEY P., HUBBARD P. M., GREENBERG D. P.: The Irradiance Volume. *IEEE Comput. Graph. Appl. 18*, 2 (1998), 32–43.

[Hav00] HAVRAN V.: *Heuristic Ray Shooting Algorithms*. Phd thesis, Czech Technical University in Prague, November 2000.

[Hec90] HECKBERT P.: Adaptive Radiosity Textures for Bidirectional Ray Tracing. In *Computer Graphics (ACM SIGGRAPH '90 Proceedings)* (August 1990), vol. 24, pp. 145–154.

[HP01] HEY H., PURGATHOFER W.: *Global Illumination with Photon Mapping Compensation*. Tech. Rep. TR-186-2-01-04, Vienna University of Technology, January 2001.

[Jen96] JENSEN H. W.: Global Illumination using Photon Maps. In *Rendering Techniques '96* (June 1996), pp. 21–30.

[Jen01] JENSEN H. W.: *Realistic Image Synthesis using Photon Mapping*. A. K. Peters, Ltd., 2001.

[Jen02] JENSEN H. W.: A Practical Guide to Global Illumination Using Photon Mapping. In *SIGGRAPH 2002 Course Notes CD-ROM* (July 2002), Association for Computing Machinery, ACM SIGGRAPH. Course 43.

[Kap87] KAPLAN M. R.: The Use of Spatial Coherence in Ray Tracing. In *Techniques for Computer Graphics*. 1987, pp. 173–193.

[LH96] LEVOY M., HANRAHAN P.: Light Field Rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), vol. 30, ACM Press, pp. 31–42.

[LP02] LAVIGNOTTE F., PAULIN M.: A New Approach of Density Estimation for Global Illumination. In *Proceedings of WSCG 2002* (2002), pp. 263–270.

[LURM02] LASTRA M., URENA C., REVELLES J., MONTES R.: A Particle-Path Based Method for Monte Carlo Density Estimation. In *Poster Papers Proceeding of the 13th Eurographics Workshop on Rendering* (June 2002), pp. 33–40.

[Pel04] PELLEGRINI M.: Ray Shooting and Lines in Space. In *Handbook of Discrete and Computational Geometry - second edition*. Chapman & Hall/CRC Press, 2004, pp. 839–856.

[PP98] PETER I., PIETREK G.: Importance Driven Construction of Photon Maps. In *Rendering Techniques '98* (1998), Drettakis G., Max N., (Eds.), pp. 269–280.

[Sam89] SAMET H.: *Design and Analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods*. Addison–Wesley, Redding, Mass., 1989.

[Sch03] SCHREGLE R.: Bias Compensation for Photon Maps. *Computer Graphics Forum 22*, 4 (2003), C792–C742.

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

[Suy02] SUYKENS F.: *On Robust Monte Carlo Algorithms for Multi-Pass Global Illumination*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, September 2002.

[SW00] SUYKENS F., WILLEMS Y. D.: Density Control for Photon Maps. In *Rendering Techniques 2000* (2000), Peroche B., Rushmeier H., (Eds.), pp. 23–34.

[Tel92] TELLER S. J.: Computing the Antipenumbra of an Area Light Source. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, July 1992), vol. 26, ACM Press, pp. 139–148.

[WGS04] WALD I., GÜNTHER J., SLUSALLEK P.: Balancing Considered Harmful – Faster Photon Mapping using the Voxel Volume Heuristic. In *Computer Graphics Forum* (2004), vol. 22, pp. 595–603. (Proceedings of Eurographics 2004).

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

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

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A Ray Tracing Solution for Diffuse Interreflection. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, Aug. 1988), vol. 22, ACM Press, pp. 85–92.

[WTP00] WILKIE A., TOBLER R. F., PURGATHOFER W.: Orientation Lightmaps for Photon Radiosity in Complex Environments. In *Proceedings of CGI 2000* (2000).

[ZSP98] ZANINETTI J., SERPAGGI X., PÉROCHE B.: A Vector Approach for Global Illumination in Ray Tracing. In *Computer Graphics Forum* (1998), vol. 17(3), pp. 149–158. (Proceedings of Eurographics 98).

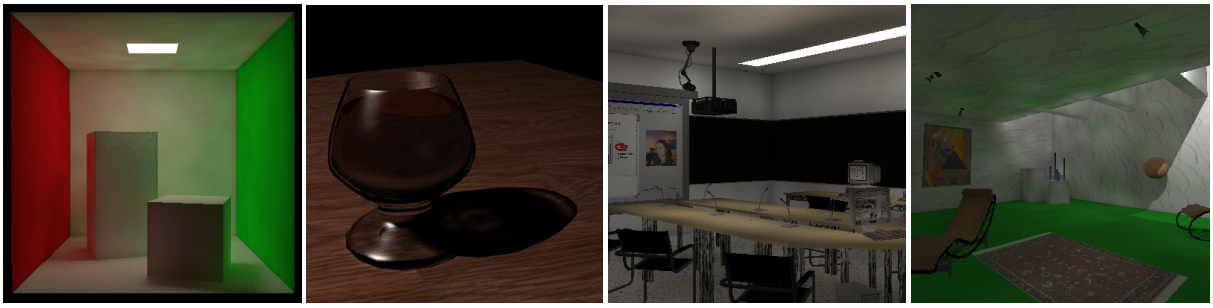


Figure 8: *The test scenes rendered using photon tracing with direct visualization of photon maps using density estimation: the Cornell Box, Cognac, Office, and Sala.*

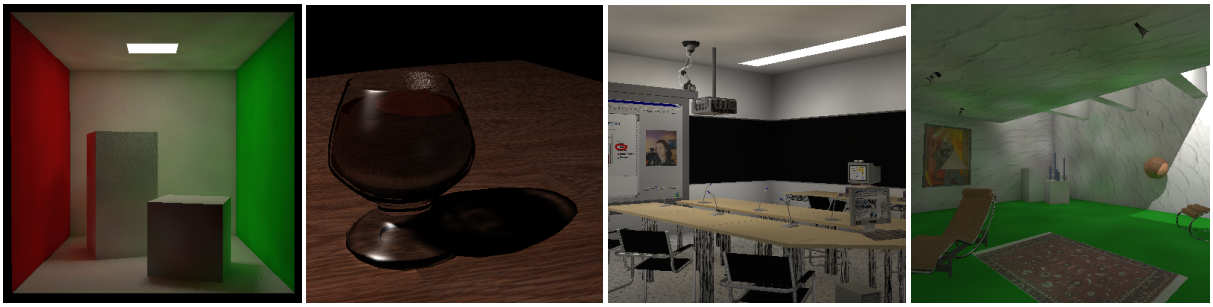


Figure 9: *The test scenes rendered using photon tracing with direct visualization of ray maps using density estimation: the Cornell Box, Cognac, Office, and Sala.*