Ray Maps for Global Illumination

Paper by Vlastimil Havran et. al.

Presentation by

Teddy Ni and Ian Lenz

Overview

- Quick intro to photon mapping
- What's ray mapping?
- What does it solve?
- What do we need to have?
- How do we do that?

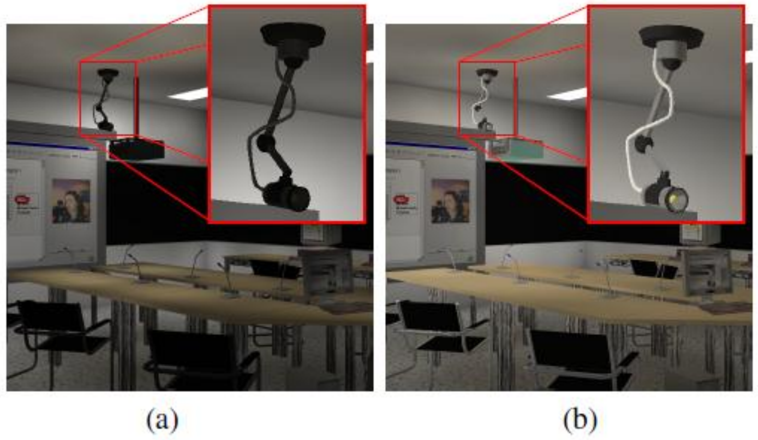
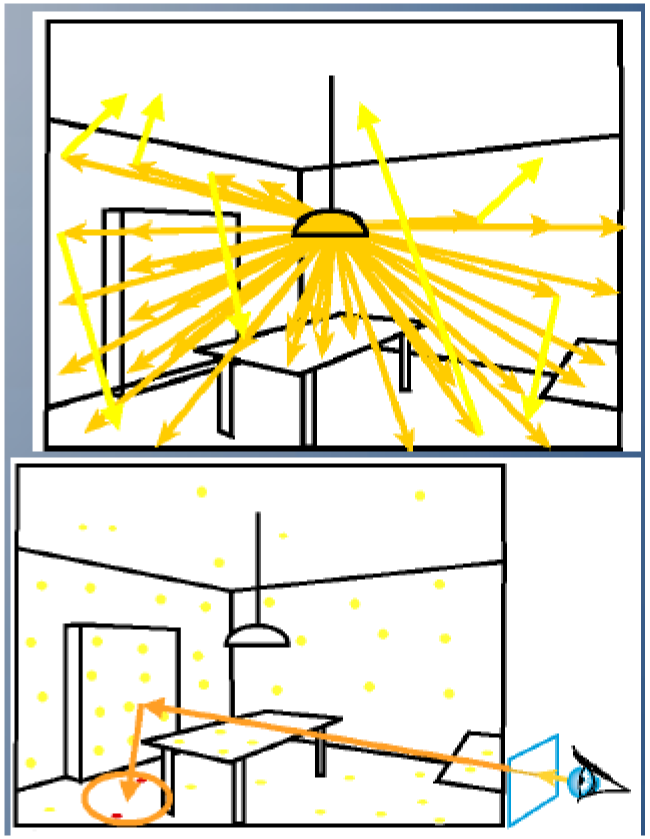


Figure 1: *The direct visualization with (a) photon maps and (b) ray maps. Notice the boundary bias removal on the lamp for ray maps.*

Photon Mapping

2 pass rendering scheme:

- First: Trace photons from light source
  - Similar to MC path tracing
  - Record location when photon hits surface
- Second: Use photon hit densities to estimate indirect radiance



Density Estimation

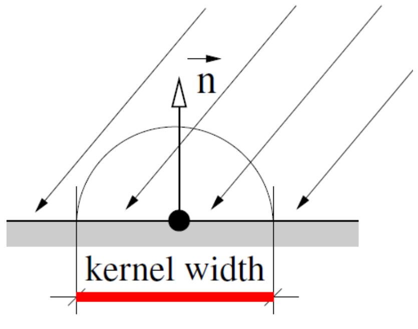
- At a point x in the scene, estimate indirect irradiance as a weighted sum of N nearest neighbor photons x_j with irradiance φ_j

$$IR(x) = \sum_{i=1}^{N} K(|x - x_j|)\varphi_j$$

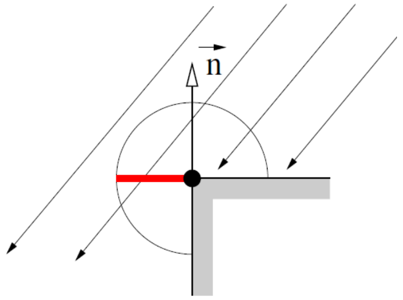
where K is some weighting function (kernel).

Problems with Photon Mapping

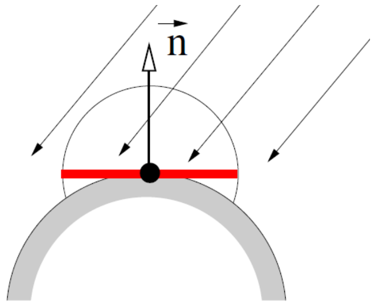
- a: Proximity bias – finite number of samples leads to blurring
  - General problem w/density methods
- b: Boundary bias – underestimate boundary illumination
  - Overestimating effective SA
- c: Topological bias – underestimate SA for curved surfaces
  - Leads to overestimating illumination



(a)



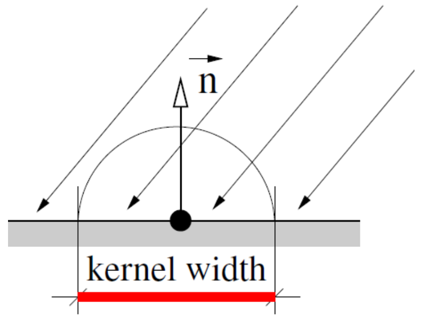
(b)



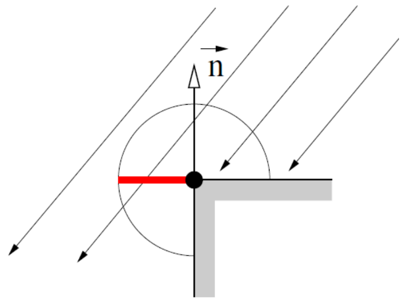
(c)

Ray Mapping to the Rescue

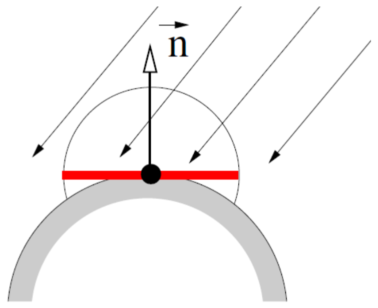
- Proximity bias (a) will always be present
- But, can solve (b) and (c) by considering rays rather than contacts
- So: record entire rays rather than just photon contact points in first pass



(a)



(b)



(c)

Basic Operations

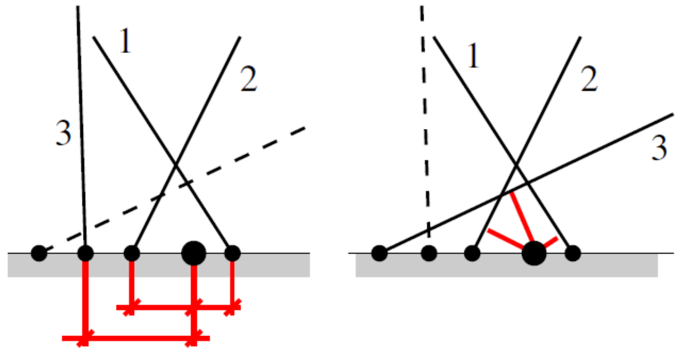


Figure 2: *Three nearest neighbors according to different ray distance metrics. (left) Euclidean distance of intersection of the ray with a tangent plane. (right) Euclidean distance of the ray itself and the center of the query.*

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, I.(a), I.(b) most useful
  - Check if ray hit the right side of a disc
- II.(a) and II.(b) used together for KNN
- Either II.(a) or II.(c) used to estimate density (usu. II.(a))

Implementation

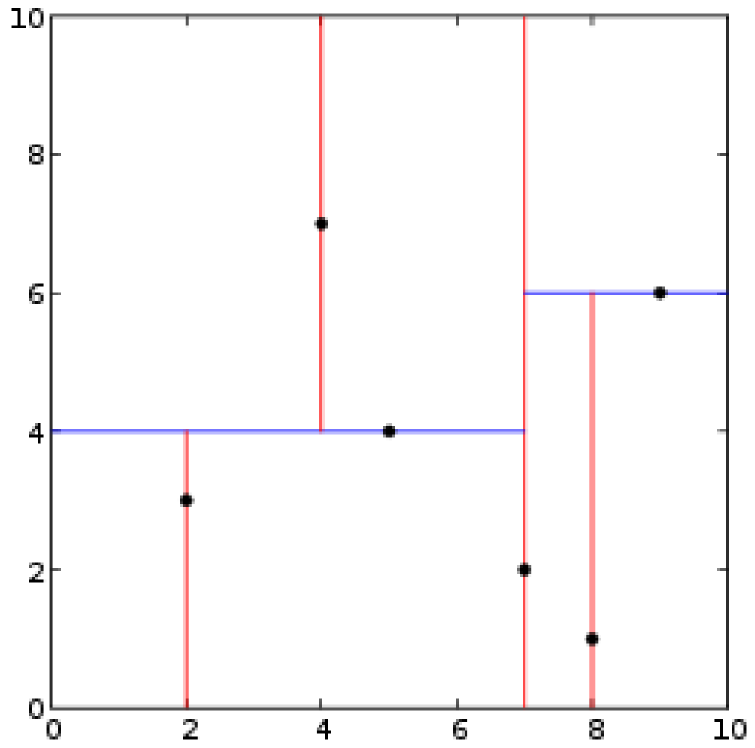
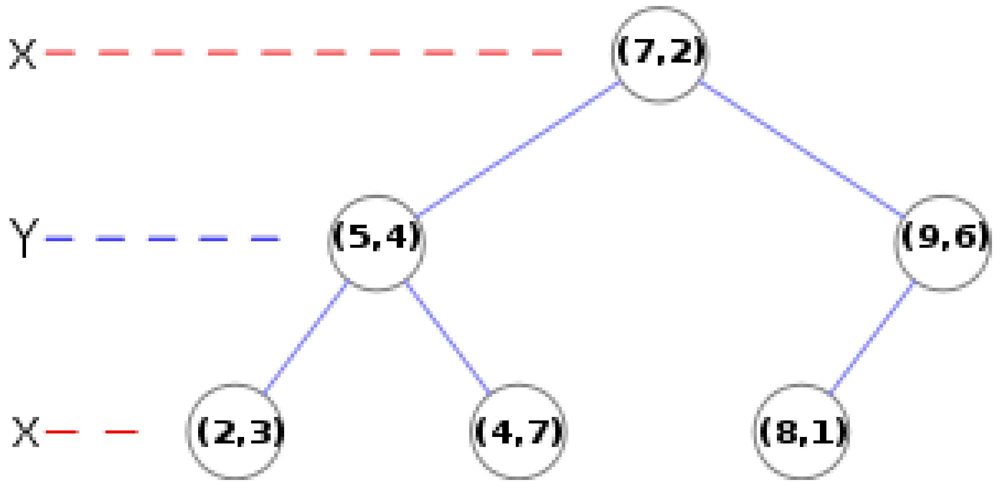
Requirements

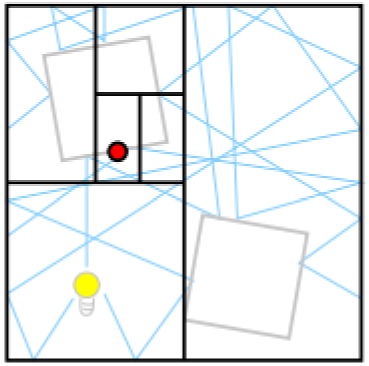
- Need to find rays in proximity to a point
- Minimize number of rays examined
- These are same goals as in photon mapping

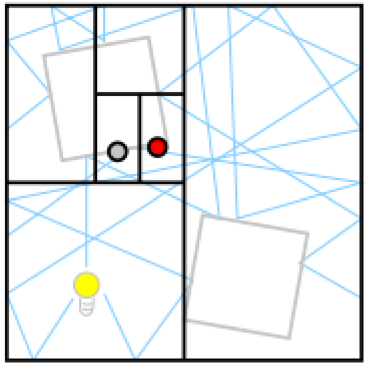
Store Rays in K-d Tree

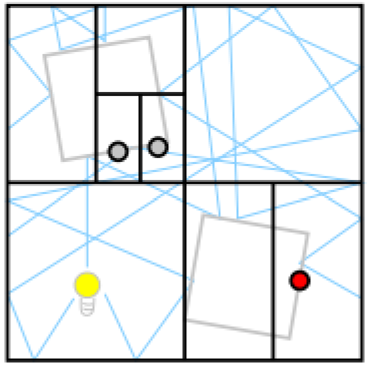
- Binary tree
- Each node corresponds to box in space
- Children correspond to smaller boxes, split by a splitting plane $\perp$ to an axis

- Each node (box) contains a list of all rays that intersect it


0
2
4
6
8
10

X
(7,2)
Y
(5,4)
(9,6)
X
(2,3)
(4,7)
(8,1)


(a)


(b)


(c)

K-d Tree Construction

- Root
- Subdivide node in direction of greatest length
  - # of rays in node > constant
  - diagonal of node is too large
  - depth of node is less than constant
- Place splitting plane in middle of current node

Lazy Tree Construction

- When we use the K-d tree for lookup, subdivide the leaf nodes we traverse when necessary
- Splitting plane in middle of node is O(1) and facilitates fast dynamic construction
- Empirically provides better performance than more complicated splitting plane decisions

Ray Lookup

- *k*-nearest neighbors query about a point
- Independent of distance metric

1. Assign each node of the k-d tree a priority
2. Priority equals minimal distance between queried point and node (box)
3. Examine nodes by increasing priority in a priority queue

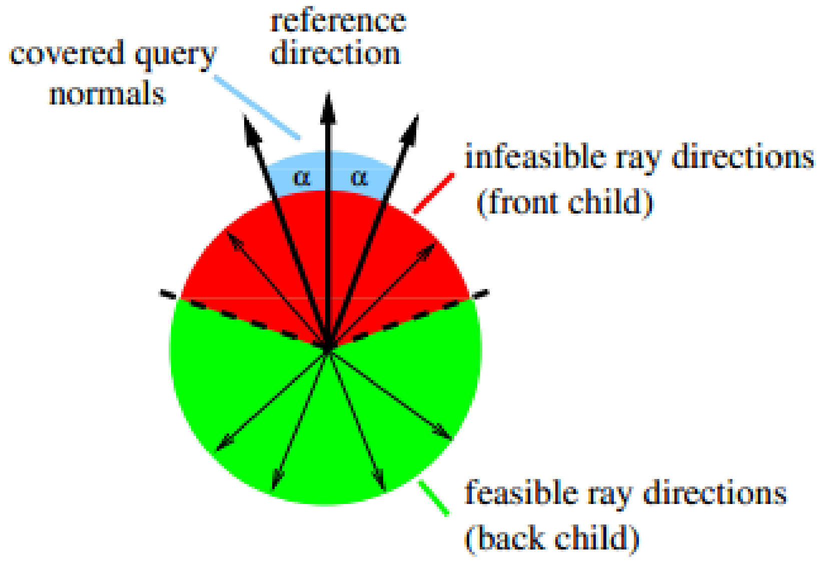
Ray Lookup

4. Keep a current list of *k* closest rays
5. At a leaf node, examine rays in node and update our current list
6. Terminate when farthest ray in current list is closer than priority of current node

- In practice, build priority queue dynamically

Extension: Directional Nodes

- Prune nodes (boxes) that do not contain rays pointing in the correct direction


covered query normals
reference direction
α
α
infeasible ray directions (front child)
feasible ray directions (back child)

Extension: Query Coherence

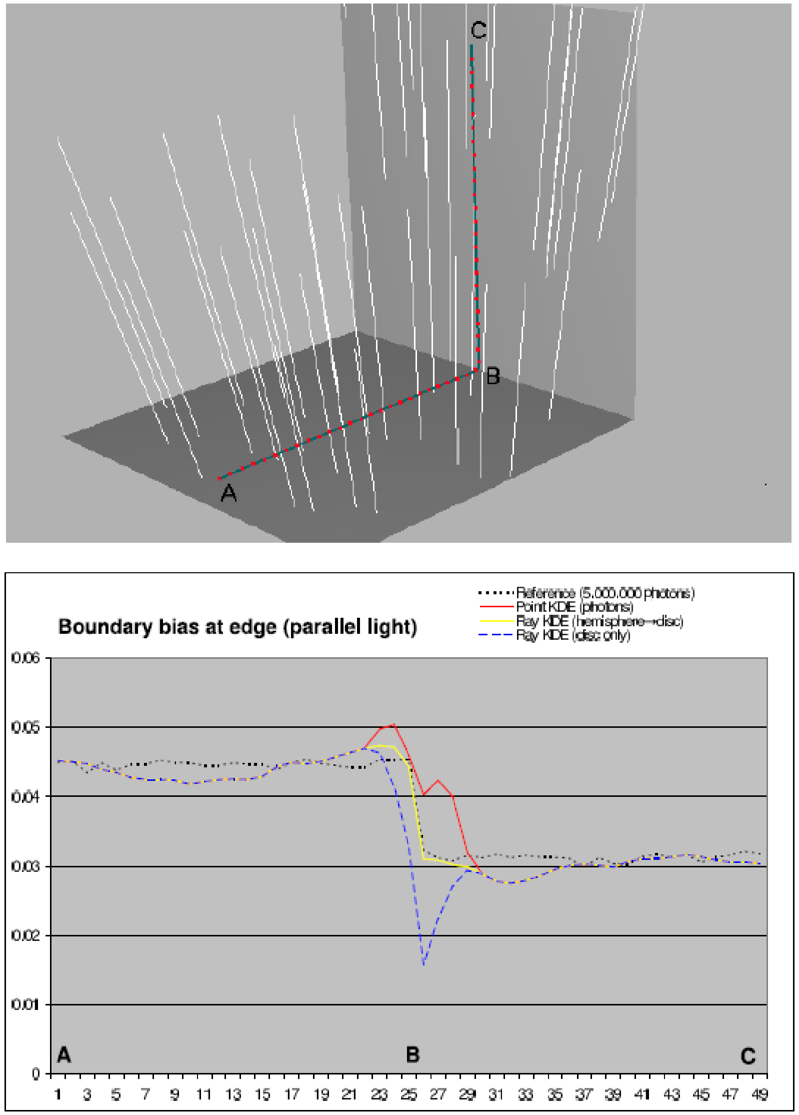
- Consecutive queries may be around points close to each other
- These queries will explore the same nodes
- Remember those nodes
  If new query close enough to last query,
  	pre-dump nodes into priority queue

Extension: Memory Cap

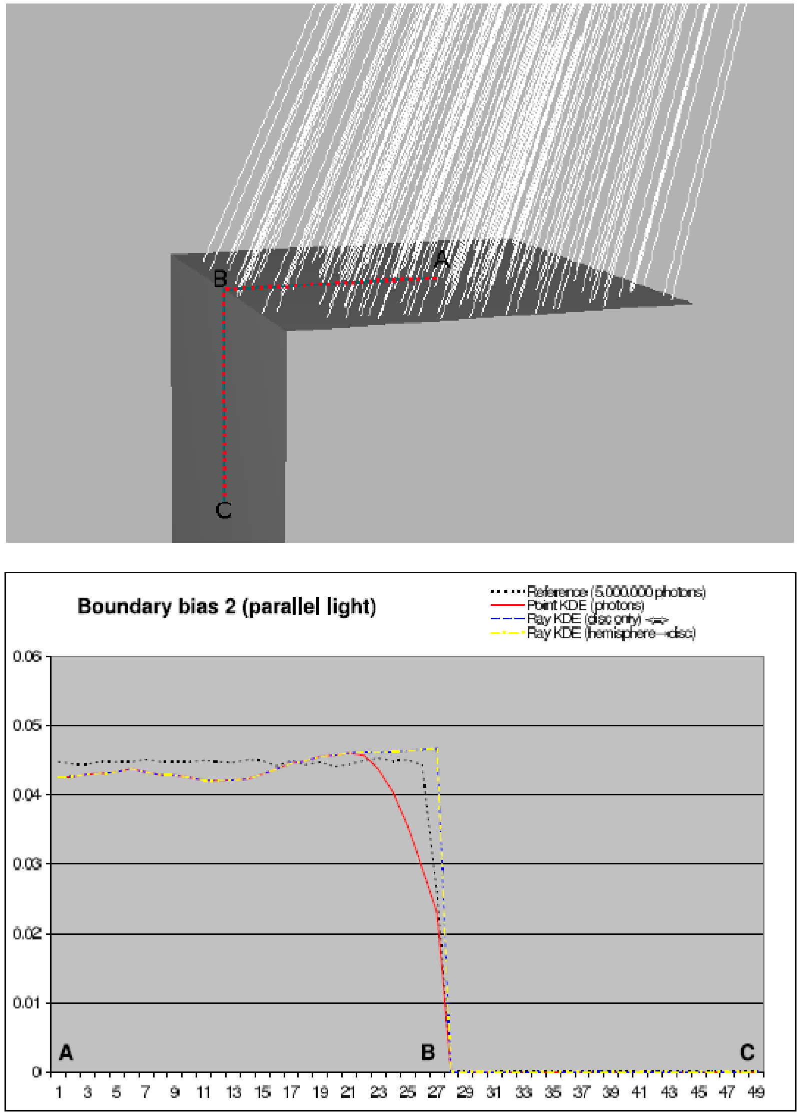
- Tag each node in k-d tree with time of last access (i.e. last put onto priority queue)
- When subdividing, if k-d tree is too large, collapse LRU nodes into single node

- Works well with coherent queries

Results

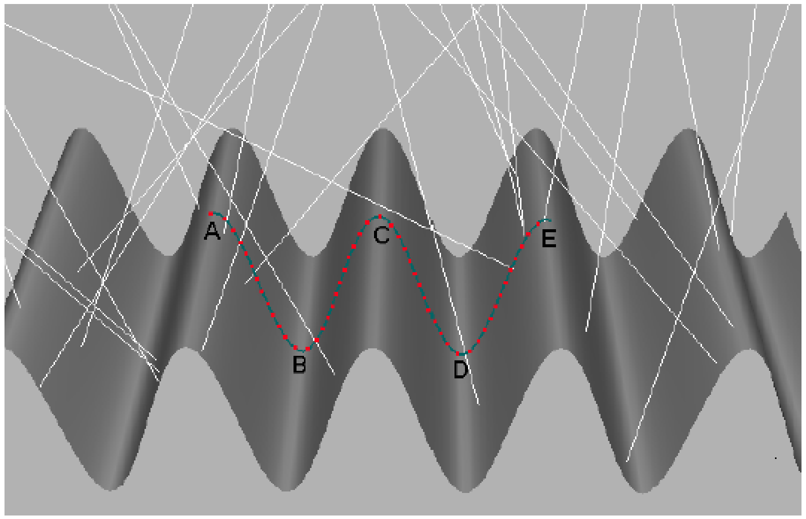

C
B
A
Boundary bias at edge (parallel light)
Reference (5,000,000 photons)
Point KDE (photons)
Ray KDE (hemisphere→disc)
Ray KDE (disc only)
0.06
0.05
0.04
0.03
0.02
0.01
0
A
B
C
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49

(a)

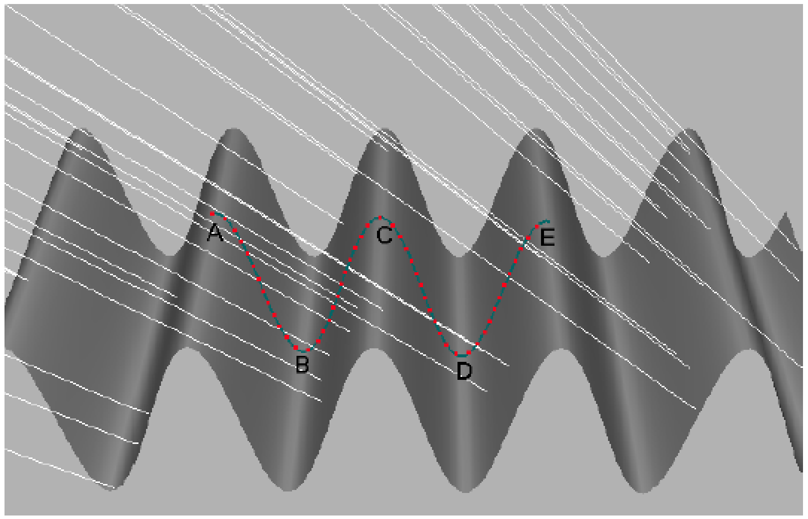

A
B
C
Boundary bias 2 (parallel light)
Reference (5,000,000 photons)
Point KDE (photons)
Ray KDE (disc only) ⇐
Ray KDE (hemisphere→disc)
0.06
0.05
0.04
0.03
0.02
0.01
0
A
B
C
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49

(b)

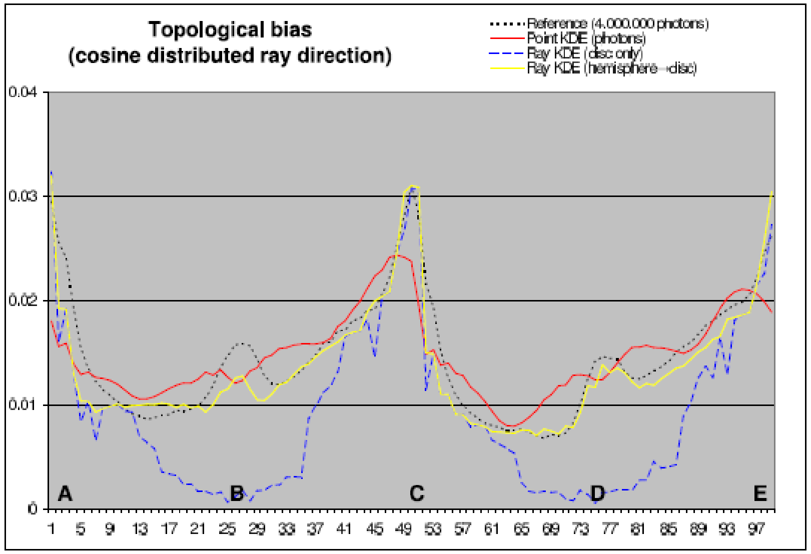
Results



A
B
C
D
E

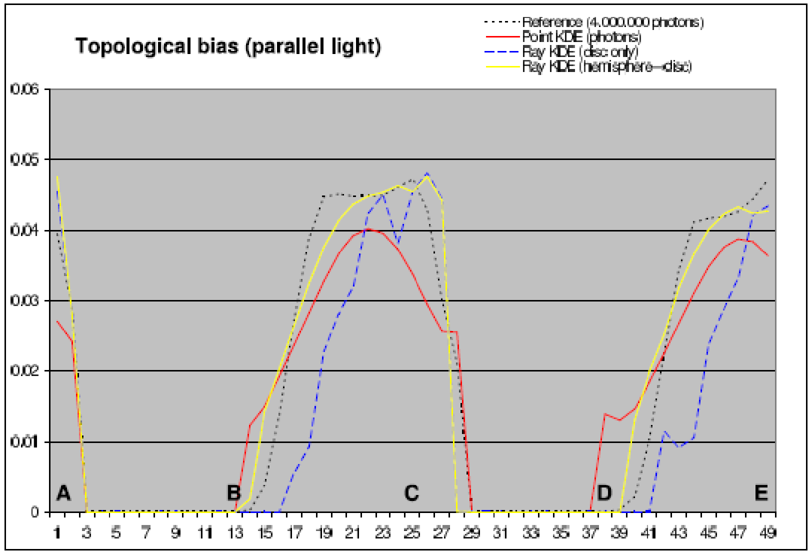


A
B
C
D
E



Topological bias (cosine distributed ray direction)
Reference (4.000.000 photons)
Point KDE (photons)
Ray KDE (disc only)
Ray KDE (hemisphere→disc)
0.04
0.03
0.02
0.01
0
A
B
C
D
E
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97

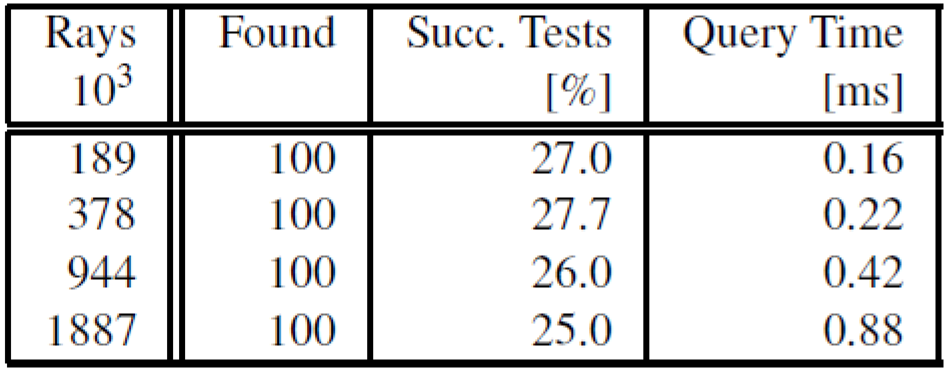
(c)



Topological bias (parallel light)
Reference (4.000.000 photons)
Point KDE (photons)
Ray KDE (disc only)
Ray KDE (hemisphere→disc)
0.06
0.05
0.04
0.03
0.02
0.01
0
A
B
C
D
E
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49

(d)

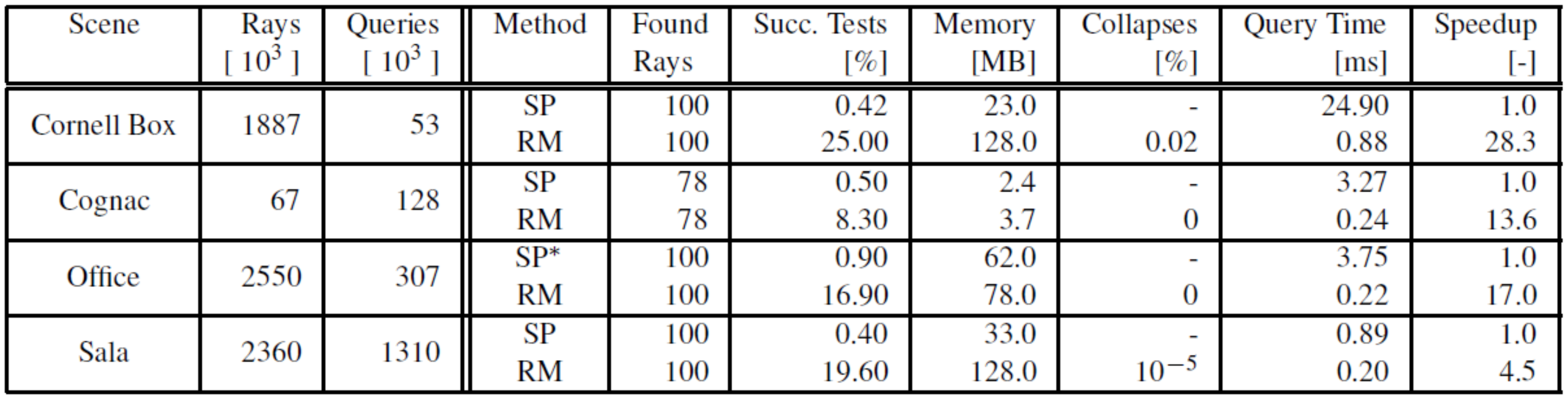
Results



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.*

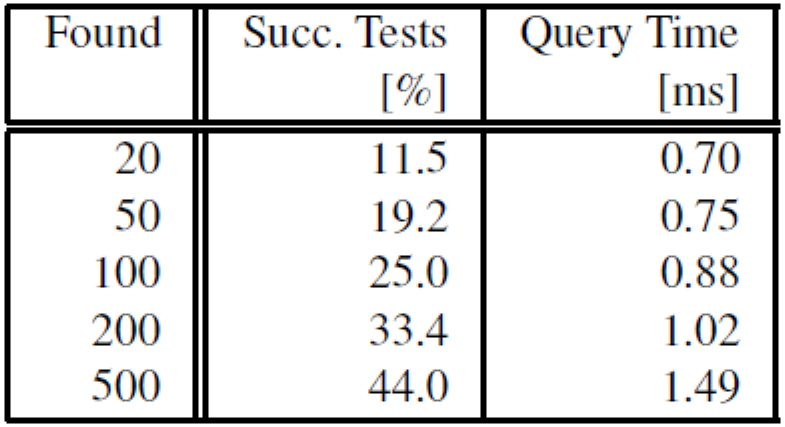
Results



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.*

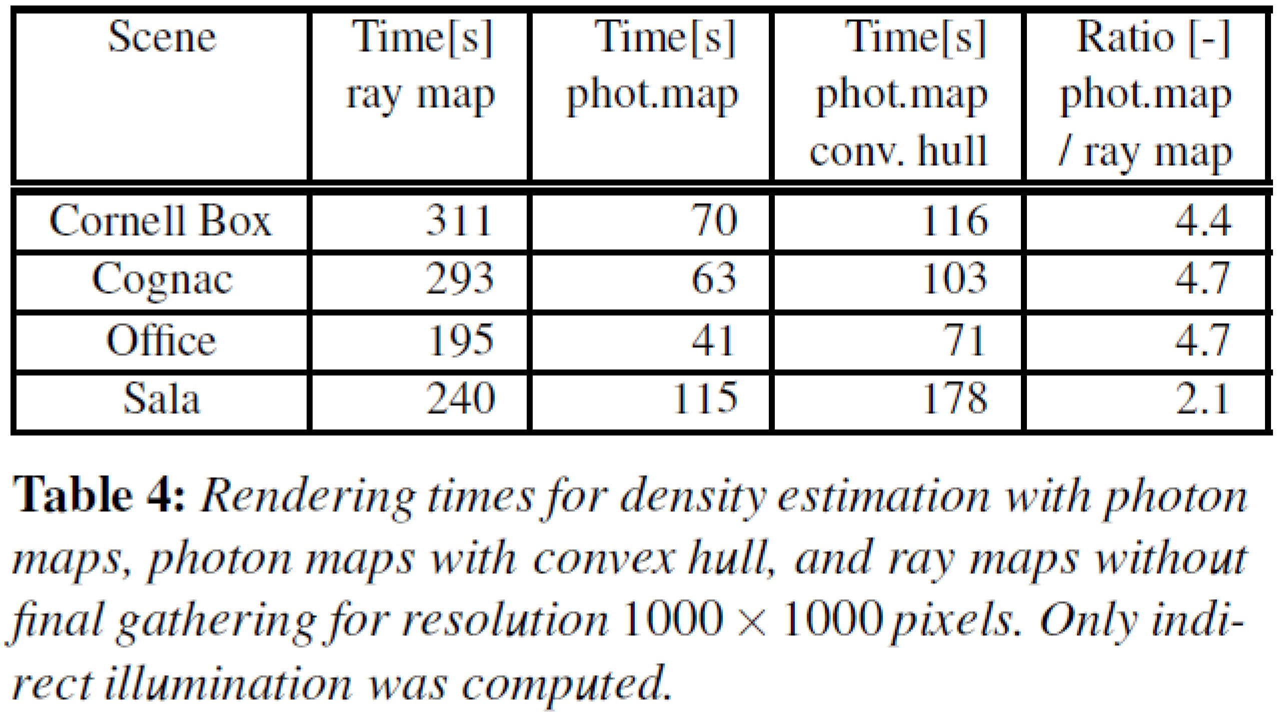
Results



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.*

Results



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.*

Results

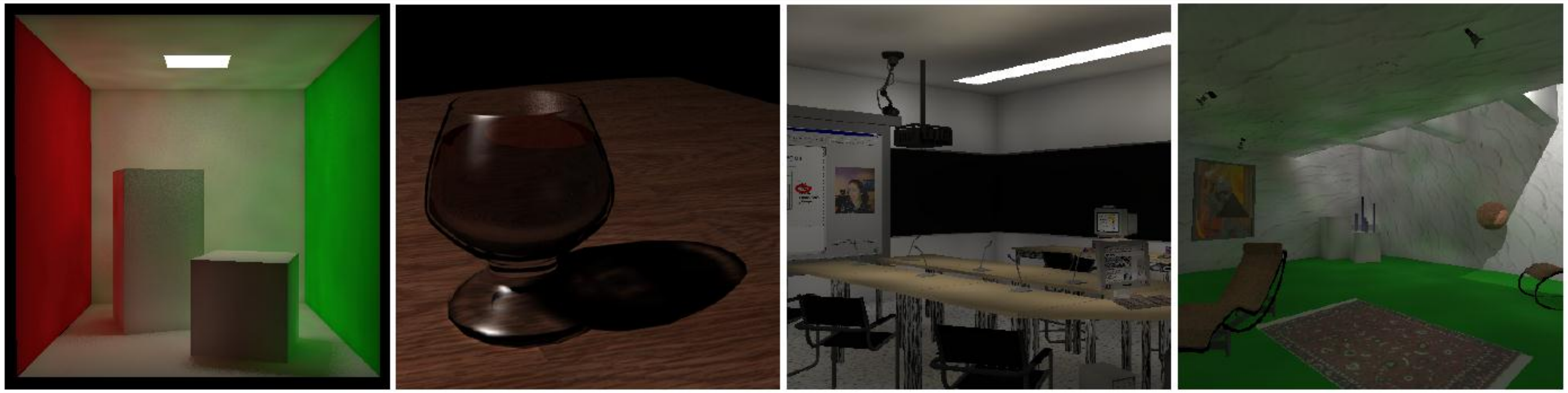


Figure 8: *The test scenes rendered using photon tracing with direct visualization of photon maps using density estimation: the Cornell Box, Cognac, Office, and Sala.*

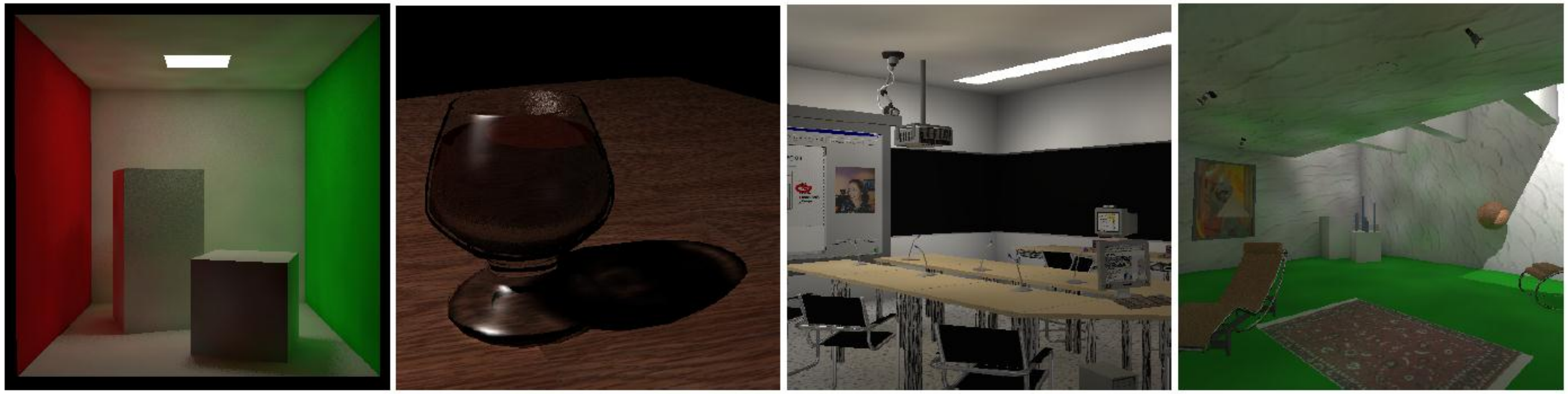


Figure 9: *The test scenes rendered using photon tracing with direct visualization of ray maps using density estimation: the Cornell Box, Cognac, Office, and Sala.*