Data Structures for Computer Graphics

Proximity Search and its Applications II

Lectured by Vlastimil Havran

Content

- Approximate range, NN, k-NN search
- Some applications of range search, NN search, and k-NN search in computer graphics

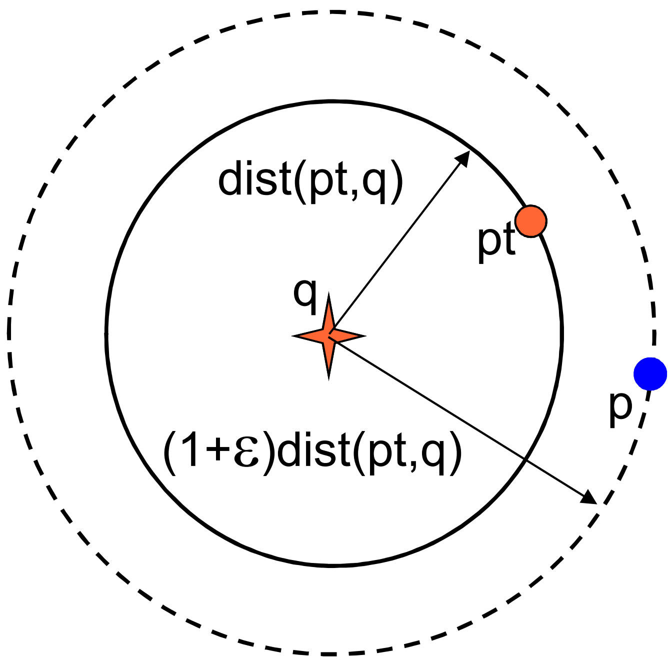
ε-approximate Nearest Neighbor

- Definition: for any $\varepsilon > 0$, we define a point "p" to be an

  ε–approximate nearest neighbor if

  $$dist(p,q) < (1+\varepsilon)*dist(pt,q)$$

  where "pt" is a true nearest neighbor.



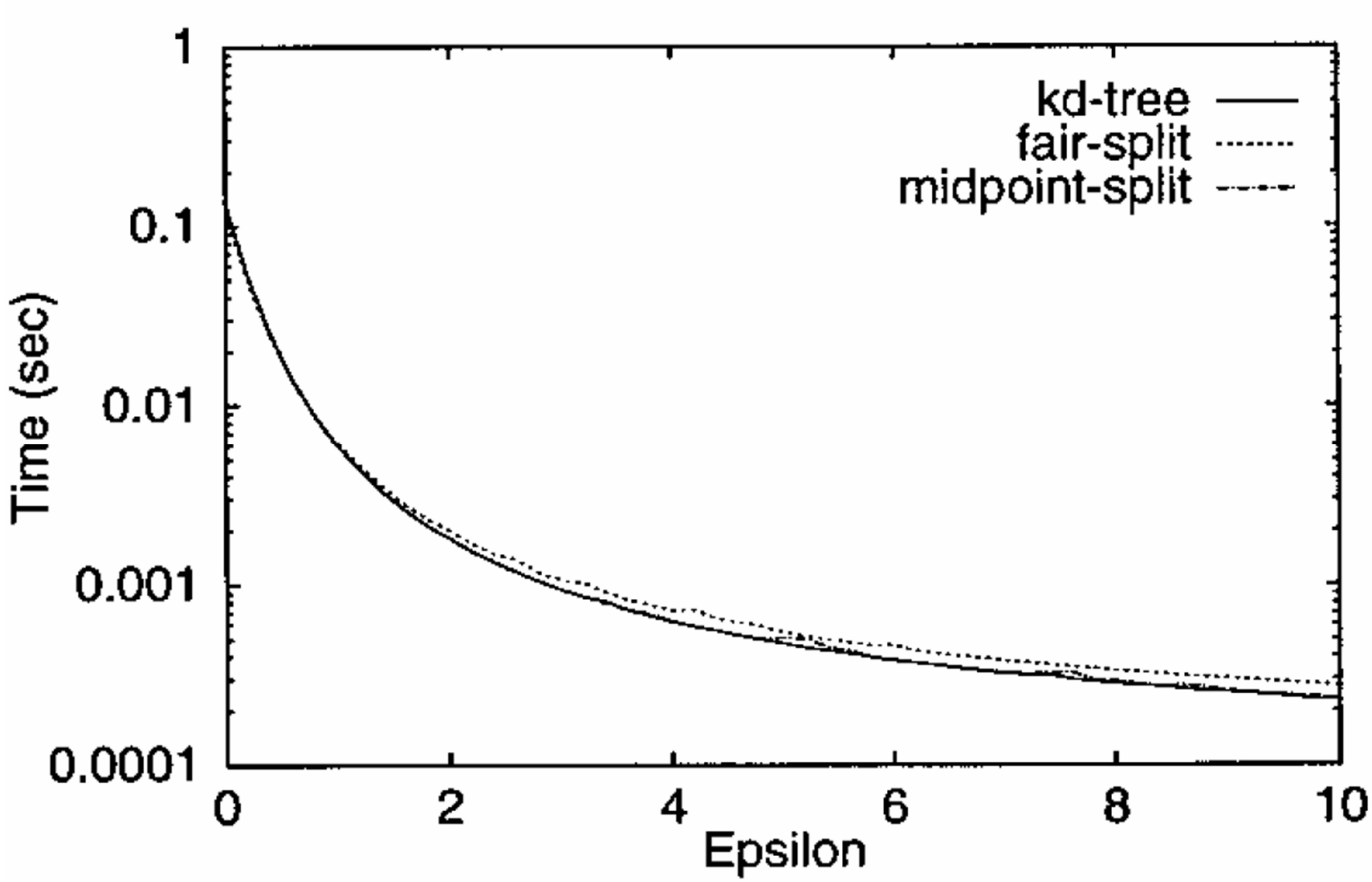
Algorithm for Approximate NN

- How to find an approximate nearest neighbor if we do not know the true nearest neighbor?
- Why do we need such an algorithm?
- Can our application use an approximate algorithm?

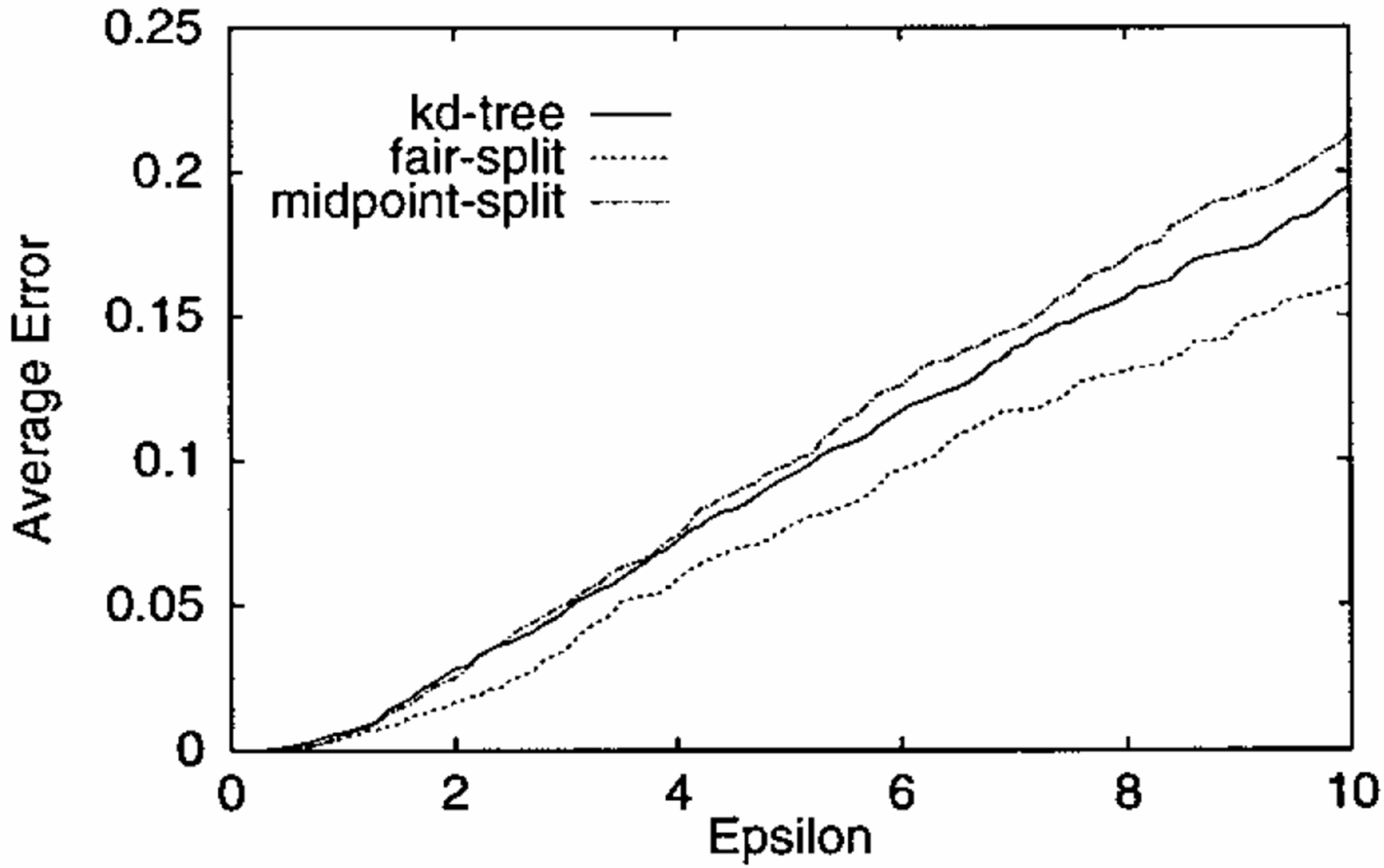
Approximate Search with Balanced Box Decomposition-trees (BBD-trees)

- Approximation can make search significantly faster with small degradation of result quality.
- Approximation is possible to use for range-search, NN search, and kNN search.
- The increased dimensionality makes a factor 2^d in search complexity for kd-trees, which makes approximate search viable for many applications in high-dimensional space.
- BBD-trees properties:
  - depth O(log N)
  - space O(N), (number of nodes O(N))
  - preprocessing time O(dN log N)
  - $(1+\varepsilon)$ approximation to NN-search in $O([1+6d/\varepsilon]^d * \log N)$
  - $(1+\varepsilon)$ approximation to k-NN-search in $O((3+k+6d/\varepsilon)^d * \log N)$ for k>1

Running Time Dependence on Epsilon

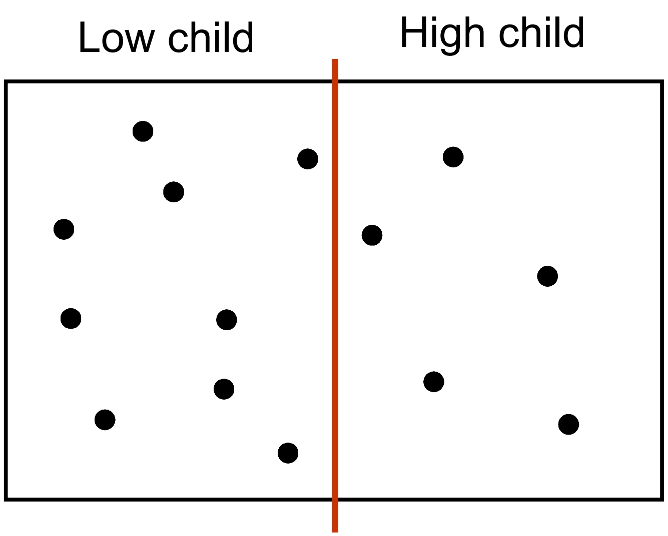


Average Error

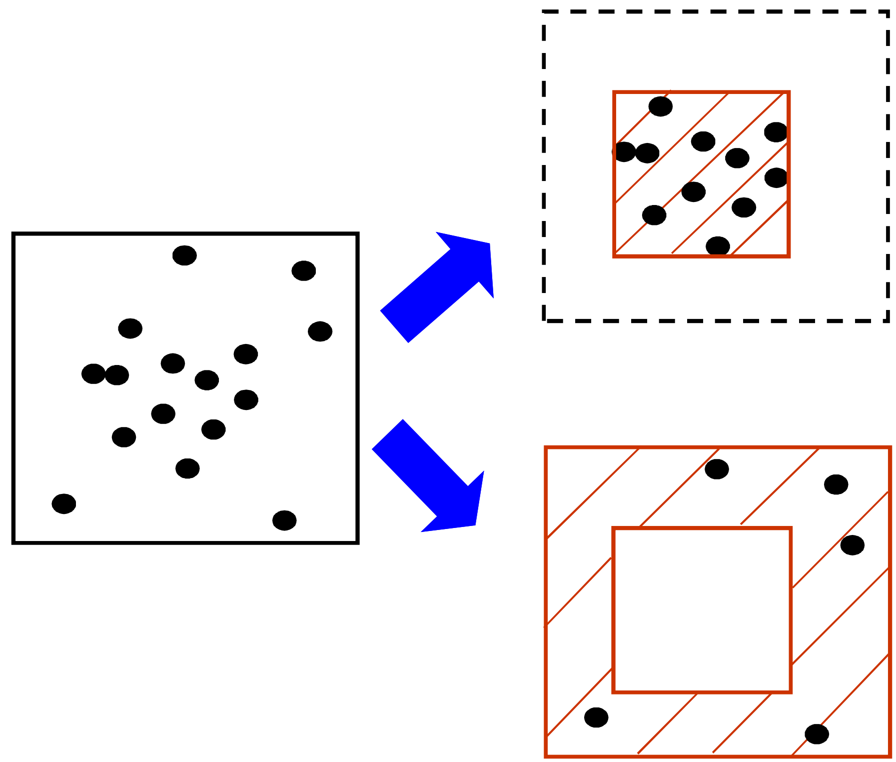


BBD-Construction: fair splits + shrinking

- Fair split: geometric median in the largest extent, axis-aligned hyperplane. Two children have boxes of the same size:



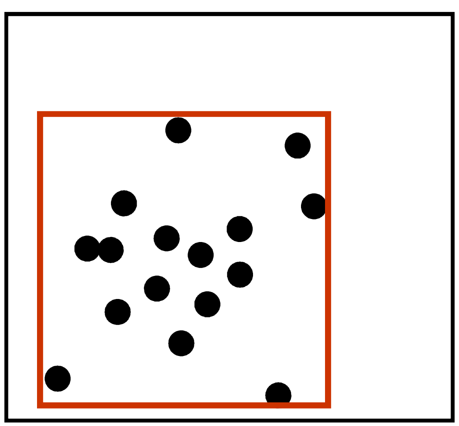
- Shrinking:
  - an inner box
  - outer box (doughnut)



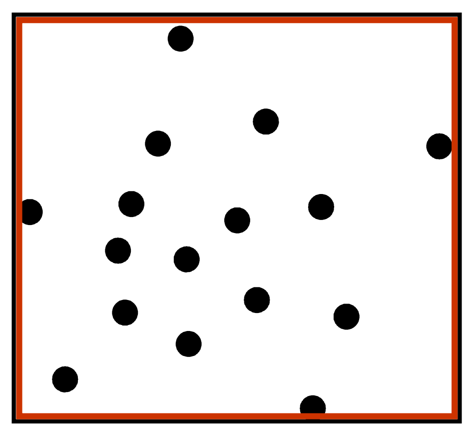
Simplest Build Algorithm

- Either carry out *fair split* or *shrink* such that
  - Fair split is preferred
  - Shrinking is performed only when it makes sense, so at least one of the faces of inner box does not lie on the outer box:

Yes, make shrinking.



Do not make shrinking, inner and outer box are equal.



Approximate NN-search with BBD-tree

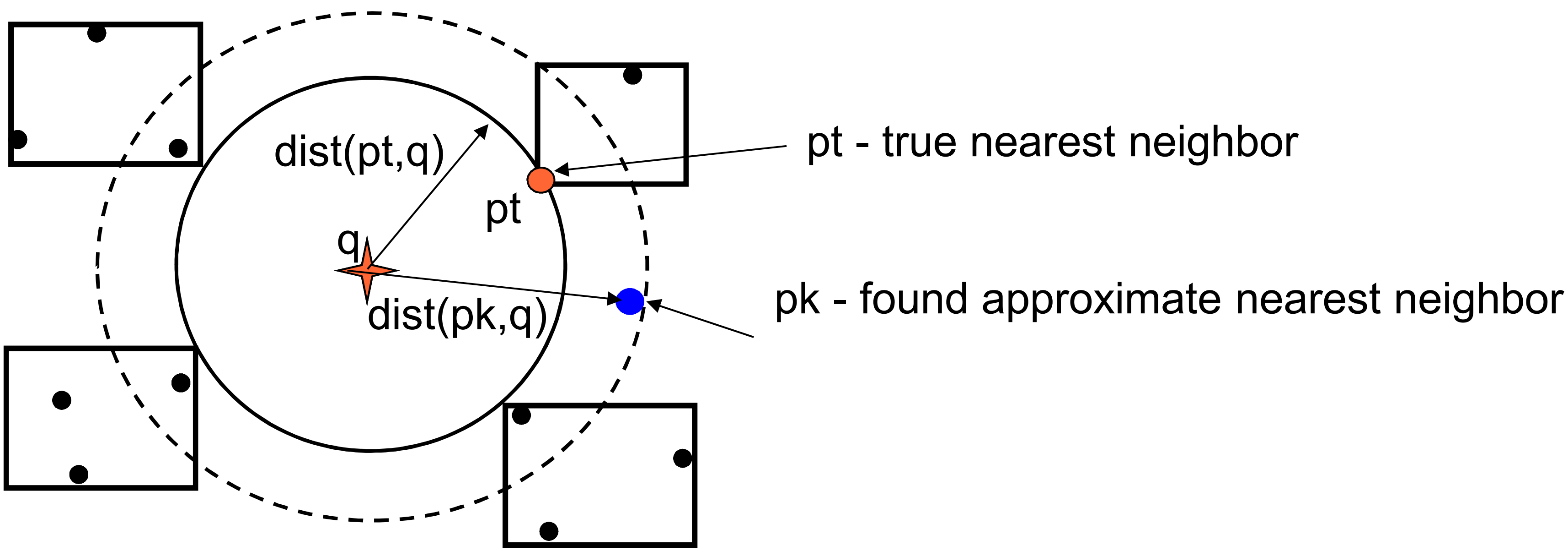
- We use an algorithm similar to NN-search with a priority queue for child nodes to visit.
- *Termination condition*: we finish the search when the closest box in the priority queue is farther than $dist(p,q)/(1+\varepsilon)$, where $dist(p,q)$ is the distance to the nearest neighbor found so far.

Note: the proof for the properties of the algorithm is fairly involved and it can be found in:

Arya et al.: An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions, Journal of ACM, 1998.

Approximate NN-search with BBD-tree

- We keep the distance to the approximate NN (pk)
- We finish the search if the nodes to be traversed in priority que are in the distance farther than *dist(pk,q)/(1+ε)*



Approximate NN Search Algorithm in Brief

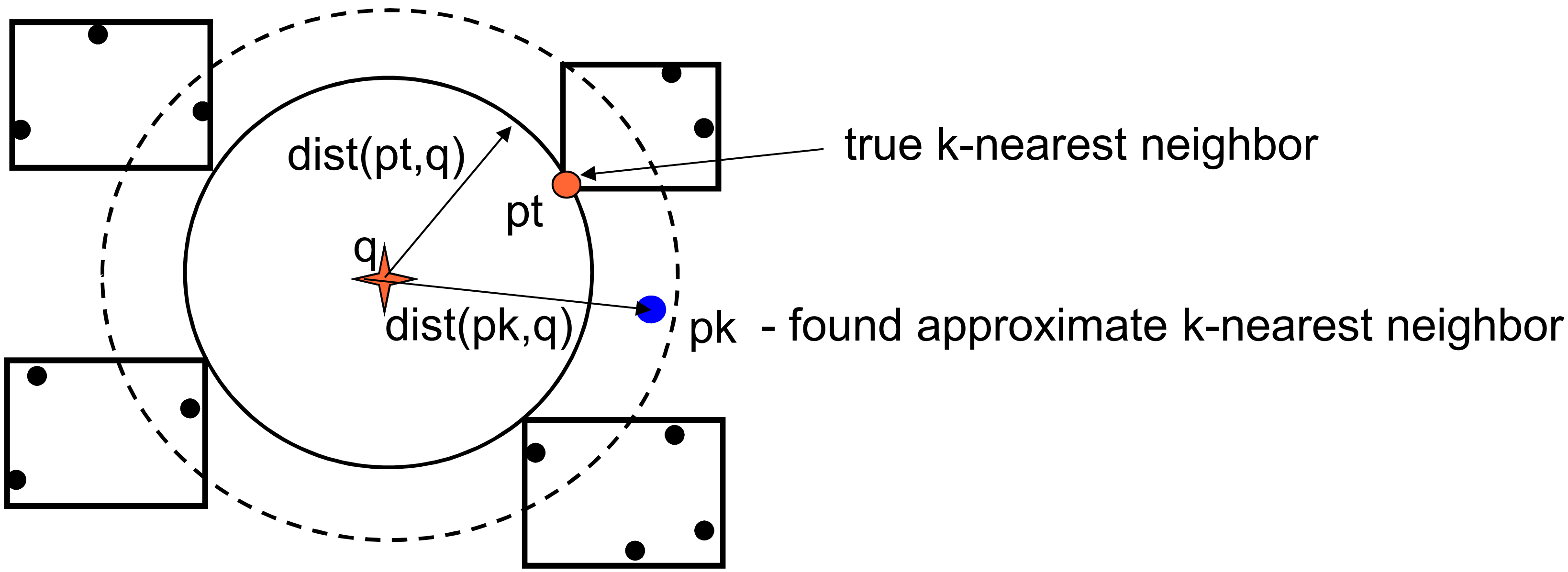
- Distance ‘d’ = infinity, approximate nearest neighbor ‘N’ = none
- Insert root node to PQ, dist 0
- DO

```
  ‘node’ = take closest one from PQ
  IF ‘node’ is leaf THEN
     Compute distance ‘dL’ to a point in leaf
     Record the nearest neighbor ‘N’ so far and update ‘d’ by ‘dL’      (if ‘dL’ < ‘d’)
  ELSE /* interior node */
     Insert the farther child to PQ (distance d1)
     Insert the closer child to PQ (distance d2)
  ENDIF
  ‘Dclosest’ = the closest node to query in PQ
UNTIL  (d / (1+eps) > Dclosest)
```

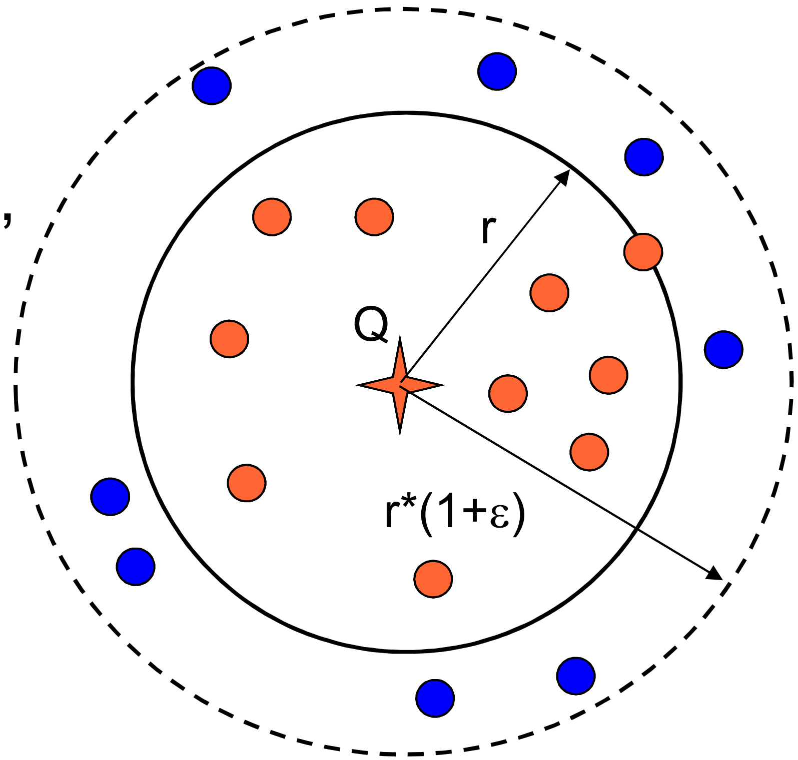
- Report the approximate nearest neighbor ‘N’

Approximate k-NN-search with BBD-tree

- The algorithm is similar to approximate NN-search.
- The sequence of k-nearest neighbors is stored in additional priority queue Q2 of fixed size k.
- We finish the search when the closest box in the priority queue is greater than *dist(pk,q)/(1+ε)*, where *dist(pk,q)* is the distance to the k nearest neighbor in the priority queue Q2.



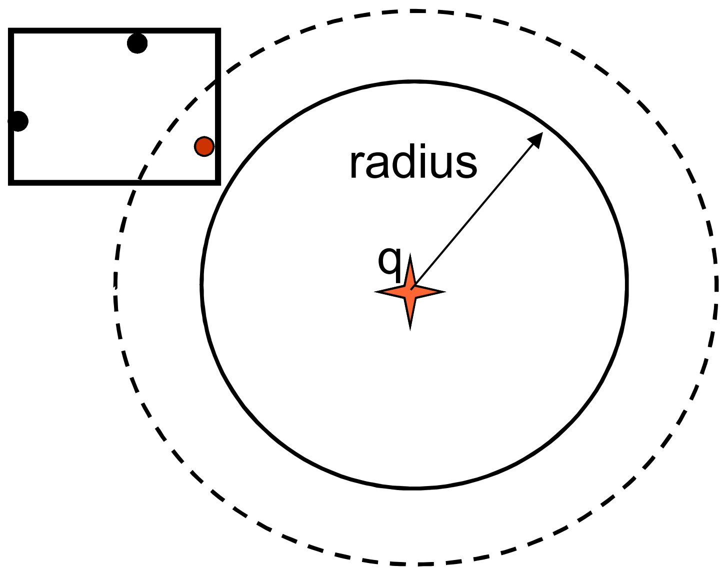
Approximate Circular Range Search with BBD-trees



- The range is extended by epsilon, so we have
  - inner radius "r"
  - outer radius $r*(1+\varepsilon)$
- We report all the cells, with maximum distance $r*(1+\varepsilon)$ from the query point
- Faster compared to the exact algorithm
- Maximum distance to all found points is smaller than $r*(1+\varepsilon)$

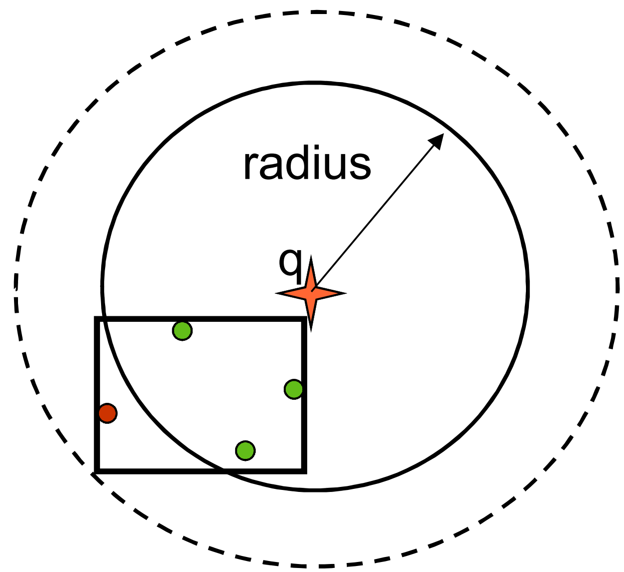
Approximate Circular Range Search with BBD-trees – 4 cases in total for a box

Case 1



Do not include

Case 2

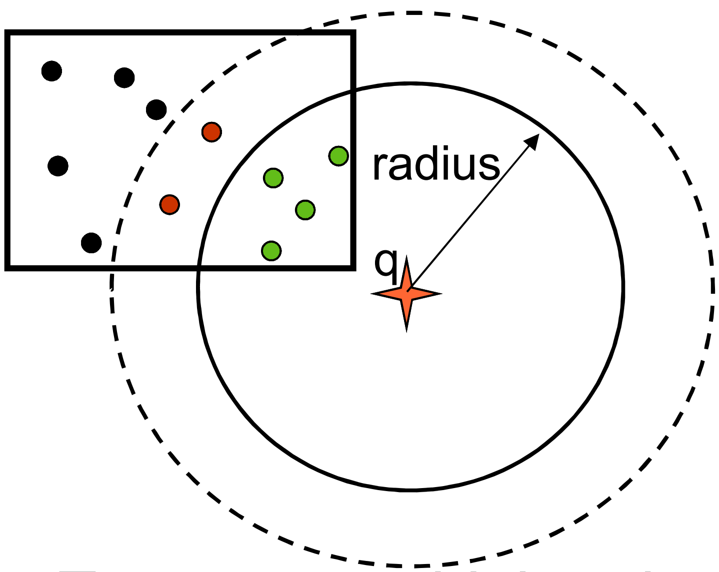


Fully include

(part of points can be approximate – red color)

Approximate Circular Range Search with BBD-trees – 4 cases in total for a box

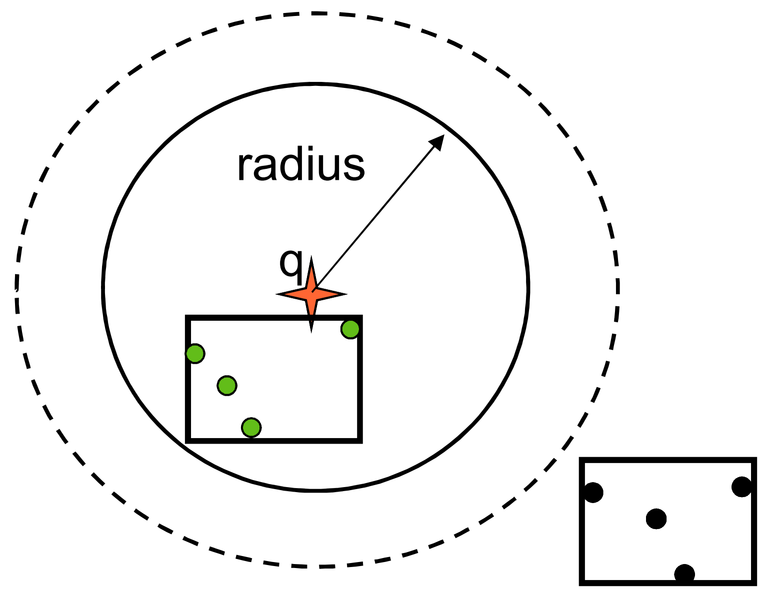
Case 3



Traverse to child nodes

or process the points in leaves

Case 4 – exactly resolved



Include or exclude

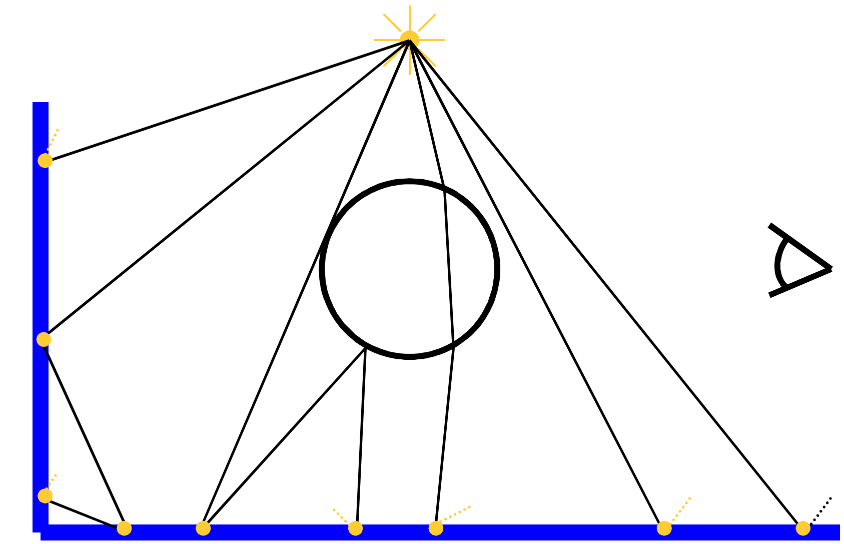
When counting – can terminate also for inner boxes

Applications of NN and k-NN search

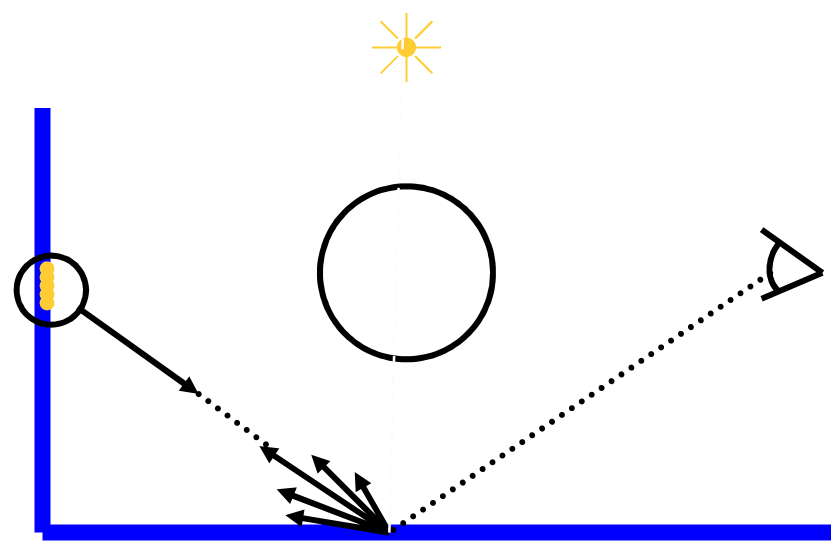
- Photon Mapping
  - equivalent to **density estimation**: we are given the hits made by some simulation and we want to recover the probability density function
- Data interpolation
  - Approximation of measured data (for example range scanner) for digitization. Each data point is: coordinate in R^d + value
  - Point-based models: re-sampling
  - Many applications outside computer graphics

Photon Mapping Algorithm Review

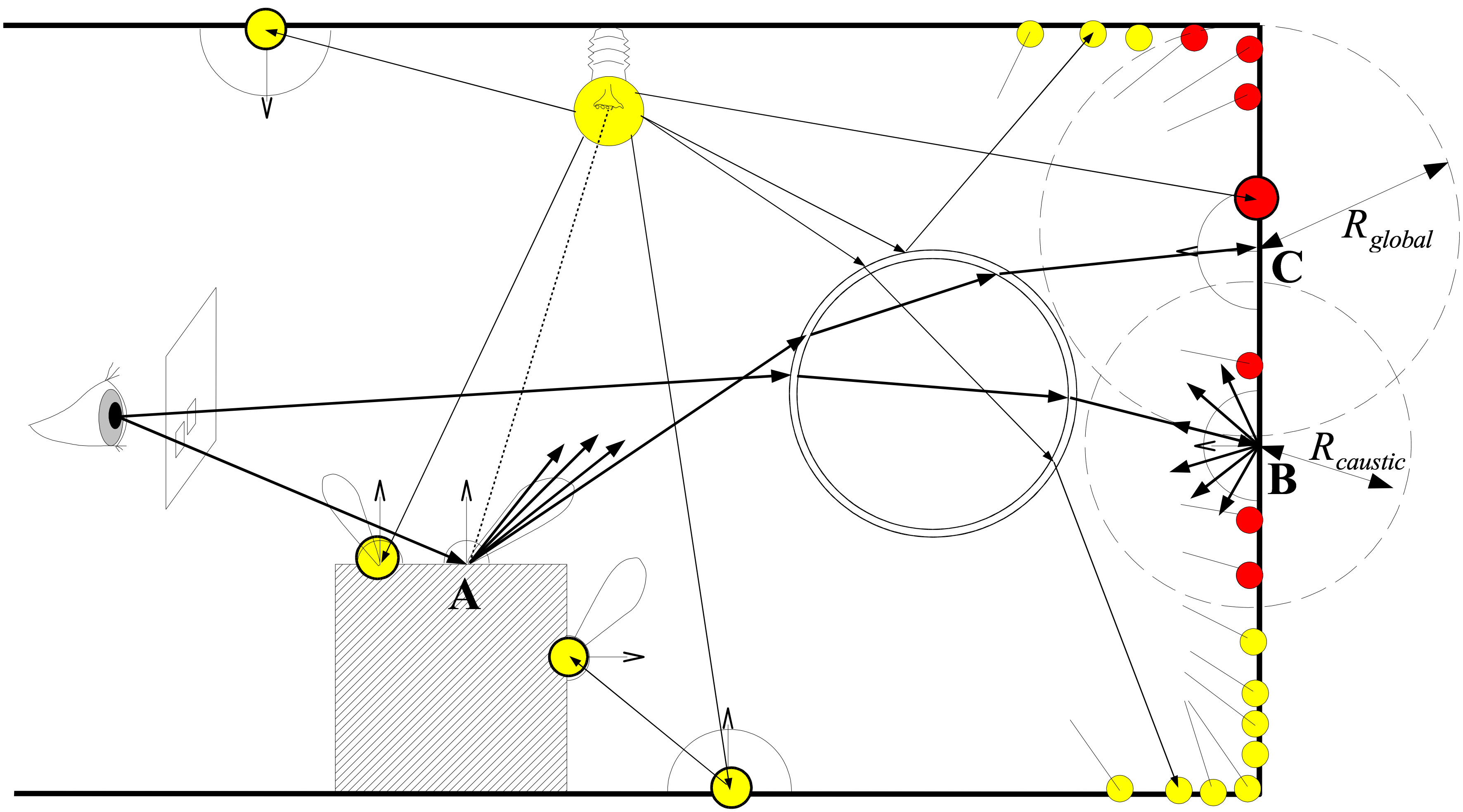
- Photon shooting
  - Emission, scattering, storing into data structure
  - Similar to ray tracing



- Gathering
  - Ray tracing for direct illumination
  - Photon map visualization
    - ➔ Indirect bounce



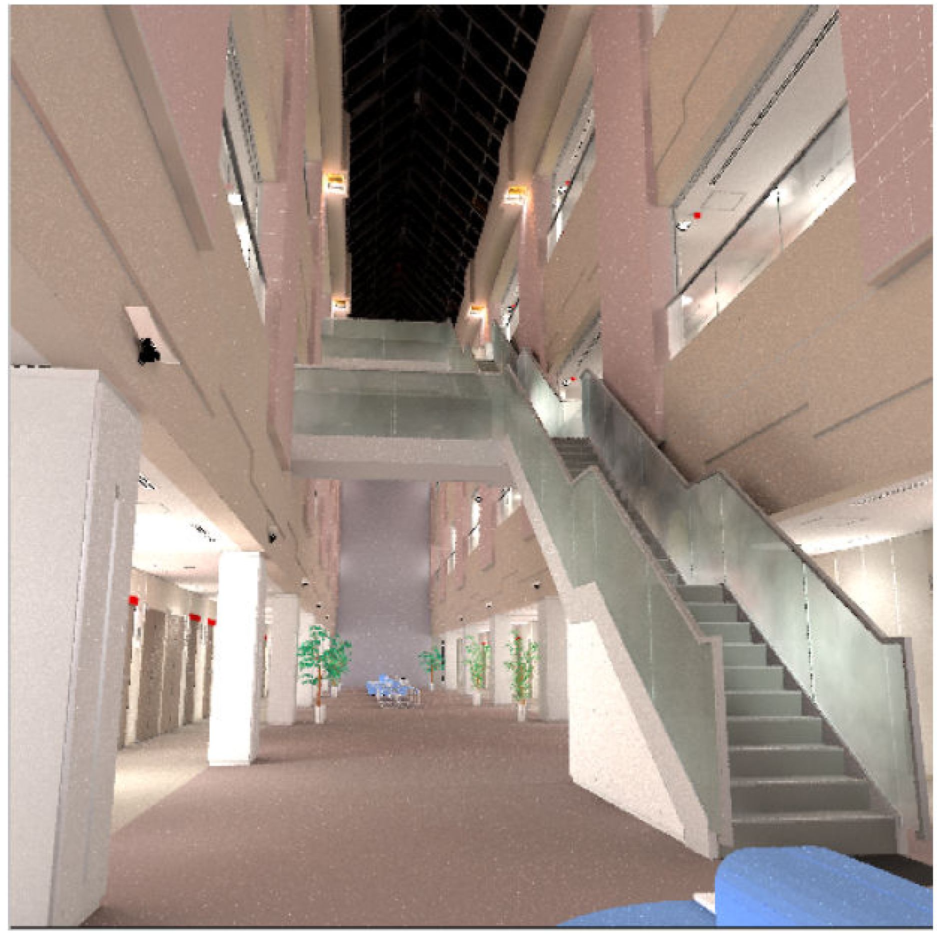
Photon Mapping – Two Phases: Shooting and Gathering



Results of Photon Mapping

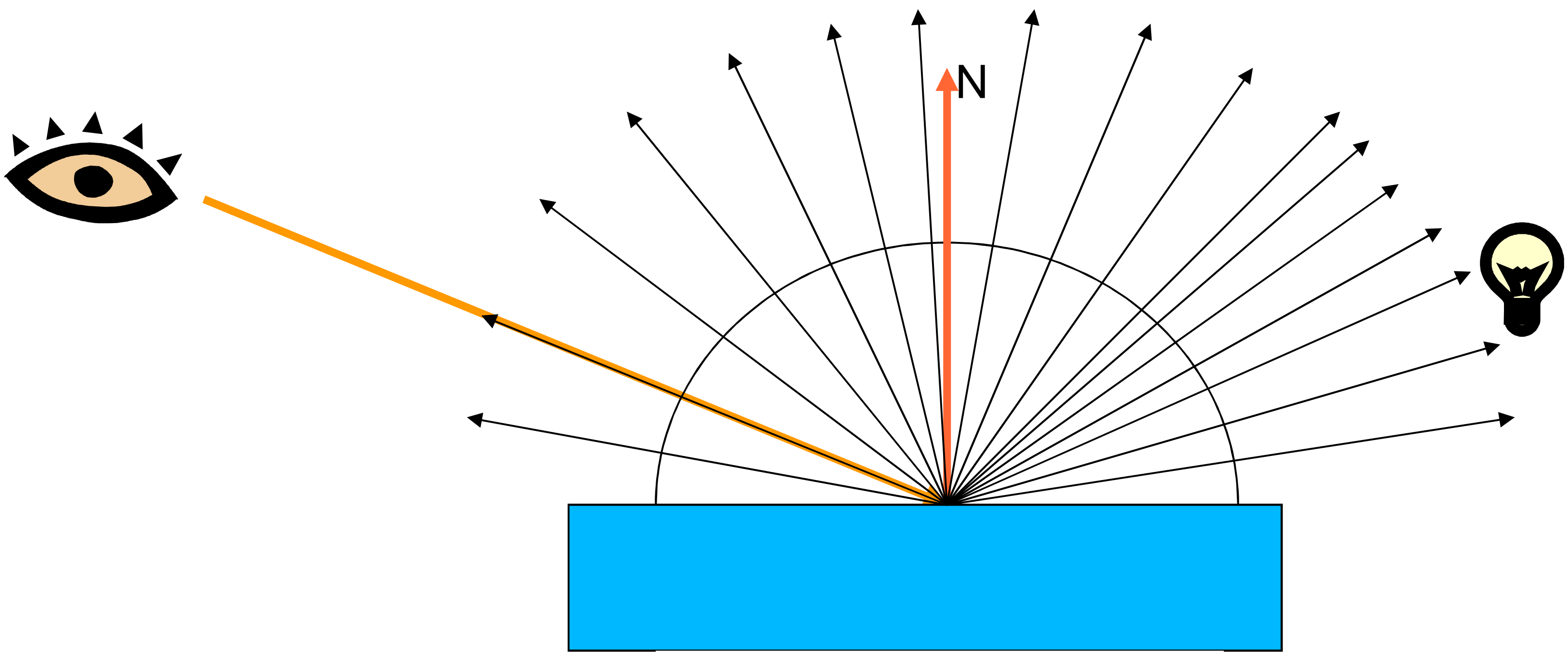
Courtesy of Henrik Wann Jensen, 2001

Other example images by photon mapping



Final Gathering

- Shooting many secondary rays (possibly according to BRDF), gathering radiances from the rays
- The radiance along gather ray is computed via density estimation that requires kNN search or range search.
- Integrating the radiances properly to render image



- Used for indirect diffuse illumination

Final Gathering in Numbers

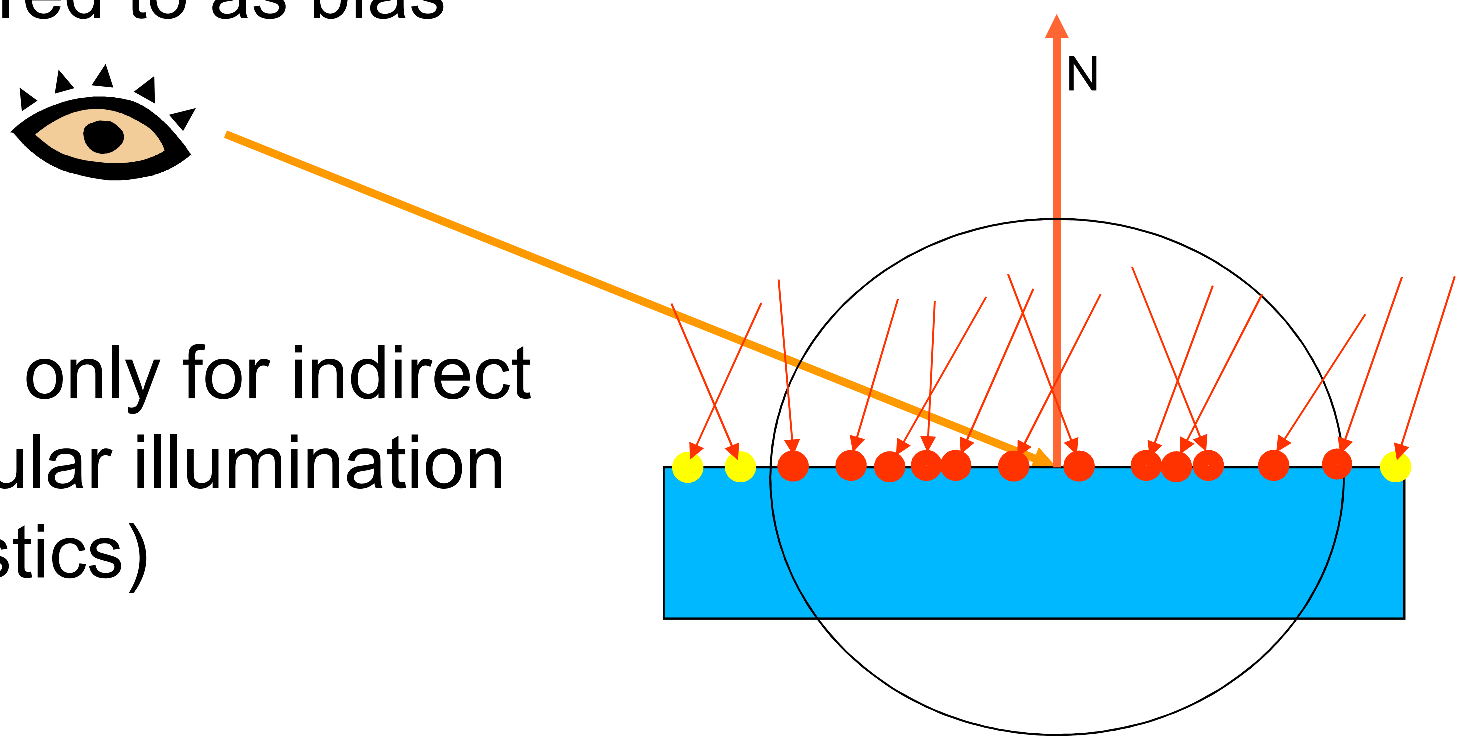
- Each final gather ray requires density estimation that uses kNN search or range search
- The number of final gather rays is computed as:

#pixels * #pixel samples * #rays per pixel sample

- 1000 x 1000 pixels * 5 samples per pixel * 1000 final gather rays per pixel sample = 5×10^9 uses of kNN or range searches
- Each search operation requires to find 20-50 nearest neighbors
- Irradiance/radiance caching – reuse some results!

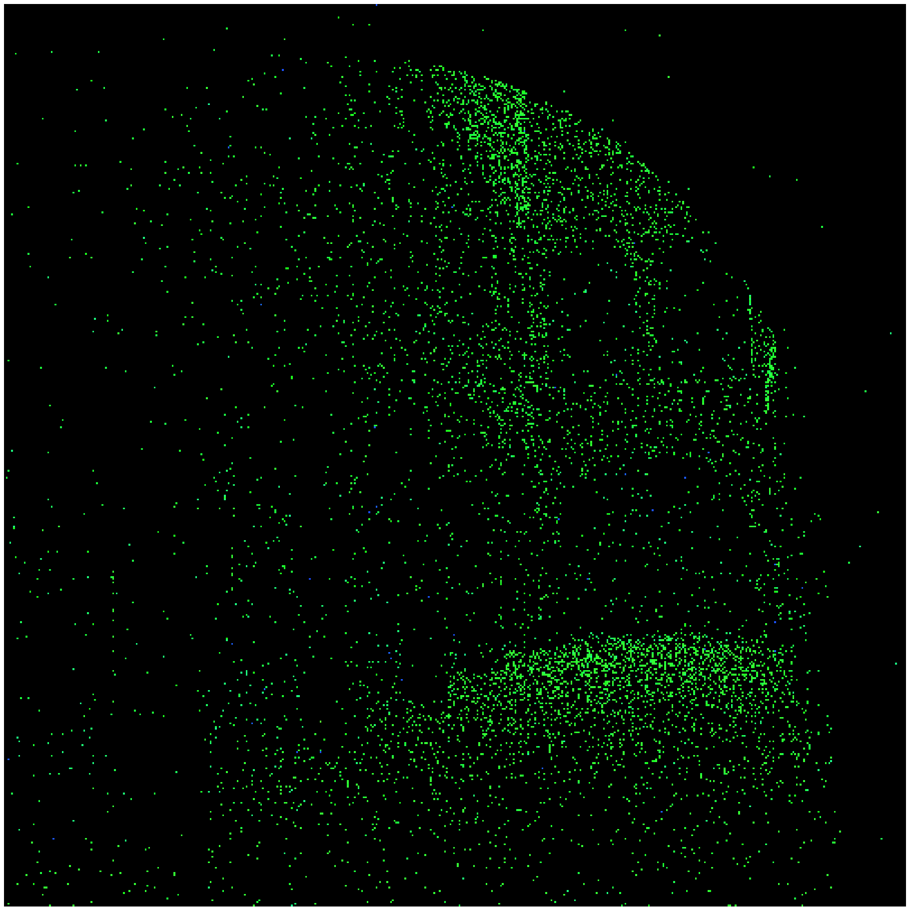
Direct Visualization of Photon Maps

- Do not shoot final gather rays, use directly visible photons from camera (primary rays)
- The radiance is computed by density estimation
- It is prone to artifacts on object boundaries referred to as bias



- Used only for indirect specular illumination (caustics)

Example of Direct Visualization of Photon Map: why we need final gathering

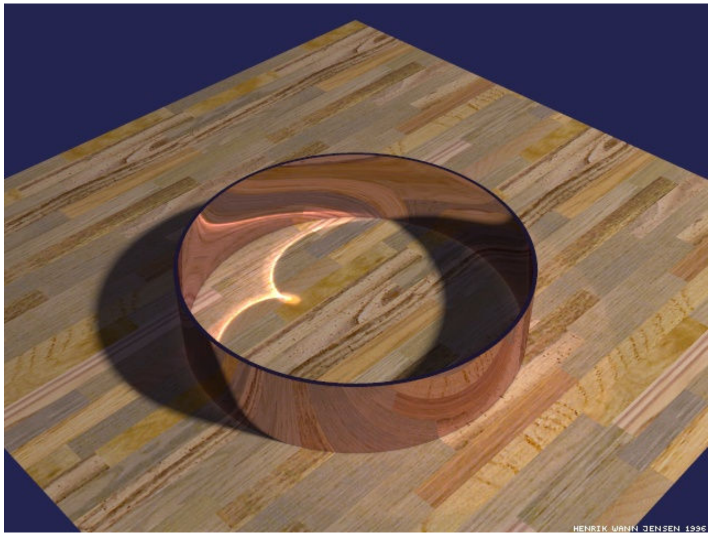


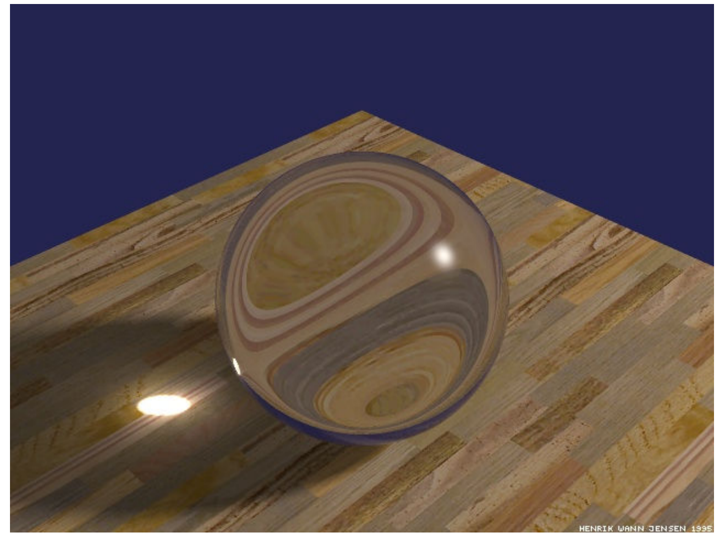
Photon Hits

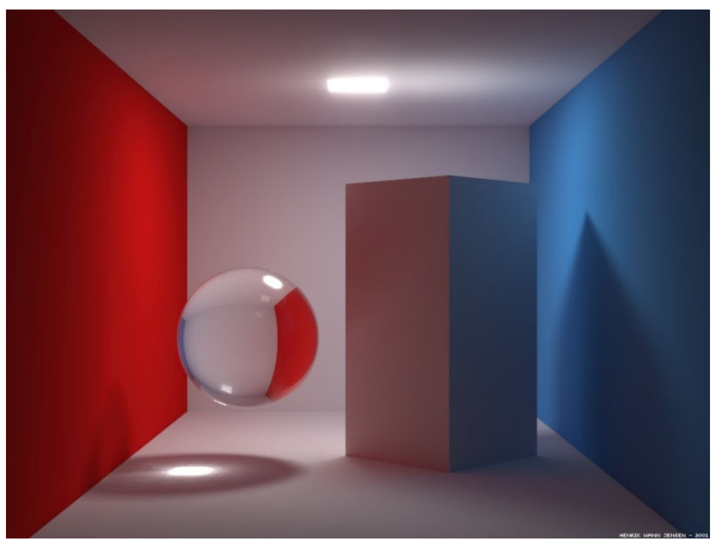
Direct Visualization

An Image with Final Gathering

Examples of Caustics



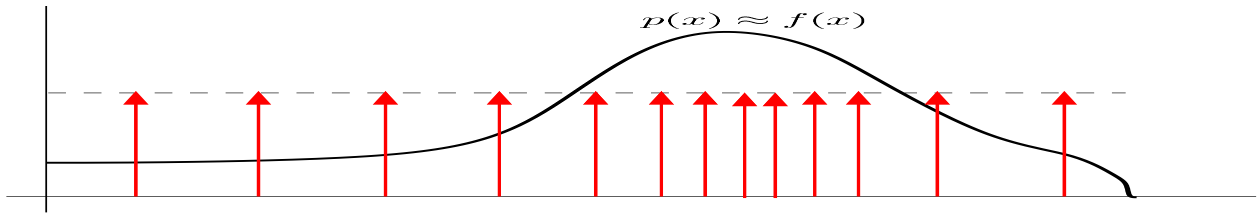




Estimating Radiance along Final Gather Ray

- Using the density estimation, from the photon hits estimating *PDF*
- It requires K nearest neighbor search for each final gather ray
- The number of final gather rays (the number of searches) is enormous (200-4000/ pixel)

Density Estimation Basics



Density Estimation: from samples we estimate probability density function *p(x)* … more complicated

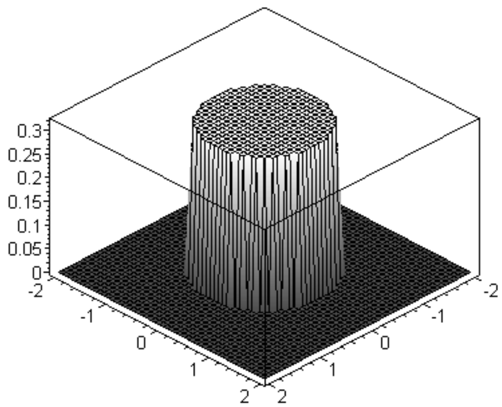
Note: **Importance Sampling**: from given probability density function *p(x)* generate samples

Intro to Density Estimation

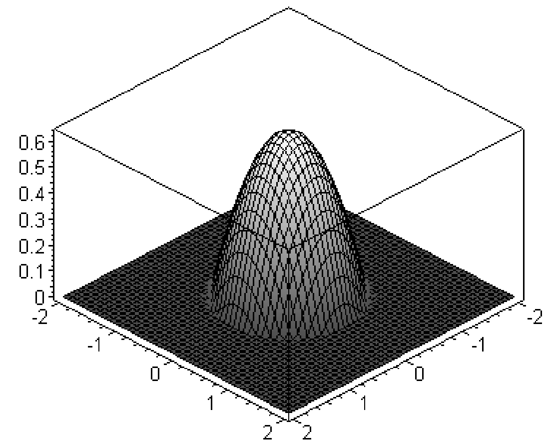
- Histogram method – record hits into buckets
- Kernel density estimation
- K-Nearest neighbors estimator
- Variable kernel density estimator
- Multiple pass methods
  - First pass – pilot estimate
  - Second pass – final estimate
- We use some kernel function to weight the importance of samples for the estimate (the weight of samples decreases with the distance of a hit from the point to be estimated)

Kernel Types

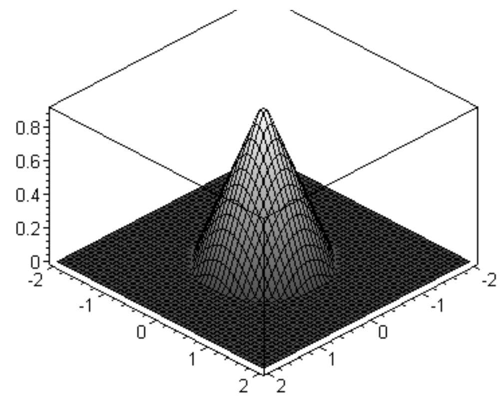
Uniform



Epanechnikov (optimal kernel)

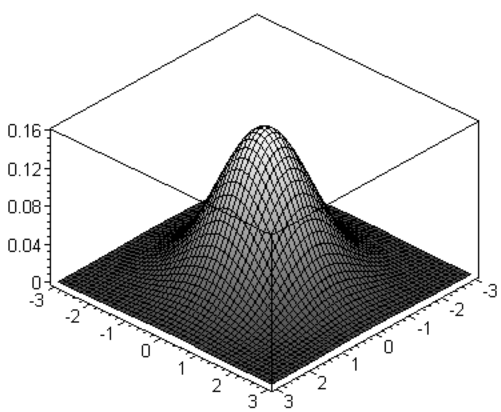


Hat

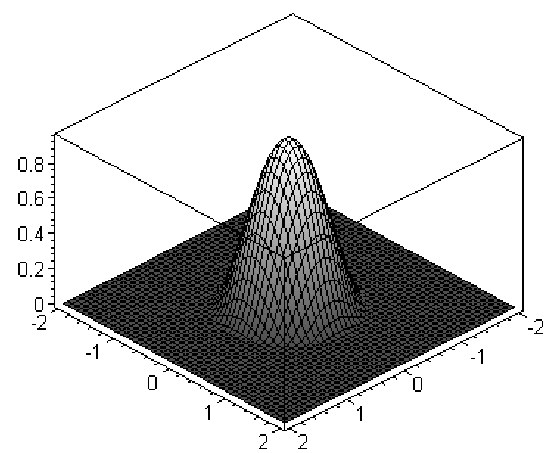


- High efficiency
- Simple formula

Gaussian



Biweight



Kernel Formulae for 2D problems

- Uniform Kernel: $K_U(t) = 1/2$ if $|t|<1$, else 0
- Hat(Cone) Kernel: $K_H(t) = 3/4.(1-|t|)$ if $|t|<1$, else 0
- Gaussian Kernel: $K_G(t) = 1/sqrt(2\pi).exp(-t^2/2)$
- Epanechnikov Kernel: $K_E(t) = 3/4.(1-t^2)$ if $|t|<1$, else 0
- Biweight Kernel: $K_B(t) = 3/\pi.(1-t^2)^2$ if $|t|<1$, else 0

Note: it is necessary to normalize the kernel, so the integration of the kernel over the input domain you get 1. Formulas above normalized for 2D domain.

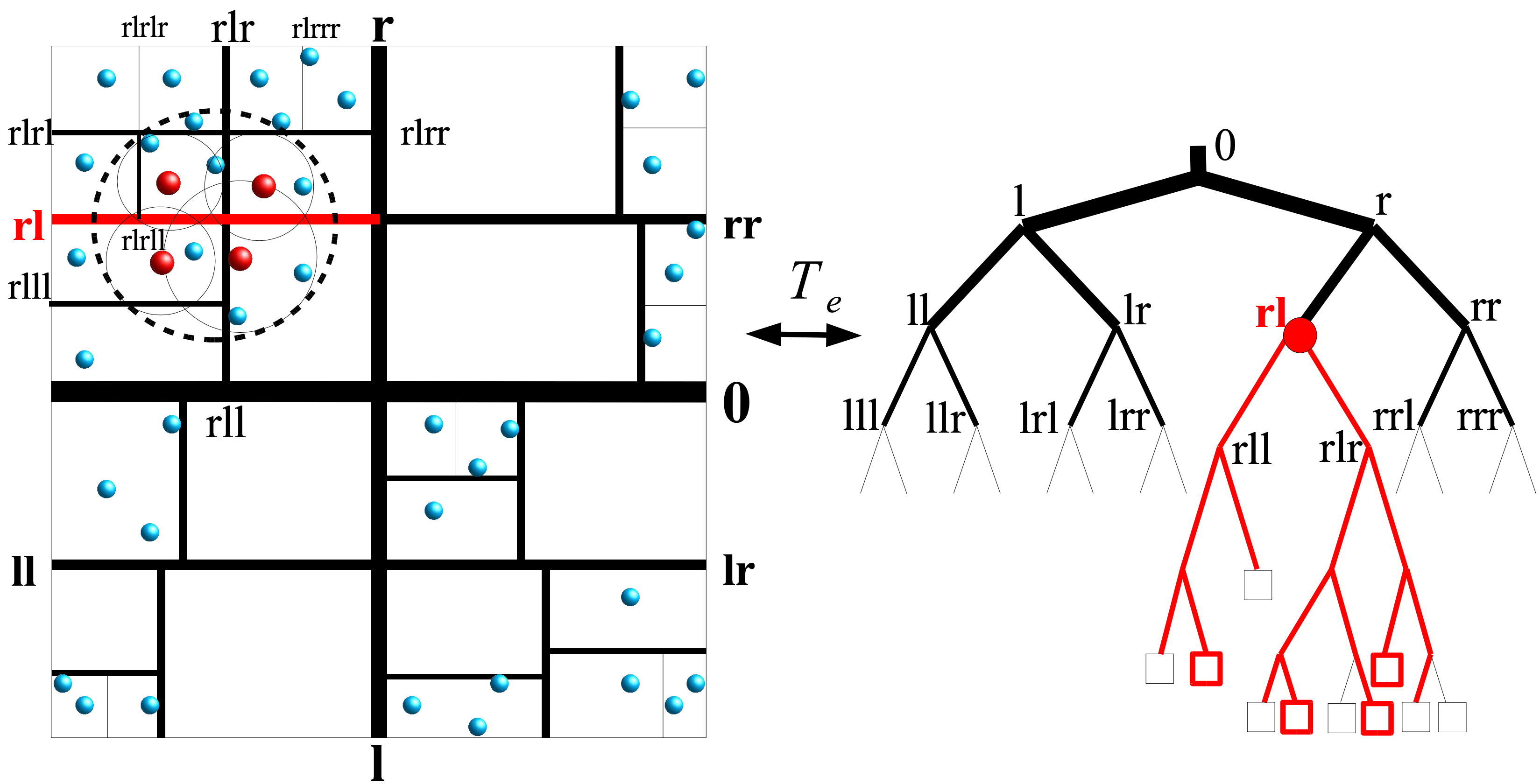
Relation to Searching

- *Range search* – given a fixed range query (sphere, ellipsoid), find all the photons in the range
- *K nearest neighbor search* – given a center of the expanding shape X (sphere, ellipsoid), find K nearest photons
  - Without considering the direction of incoming photons
  - With considering only valid photons with respect to the normal at point X

Lecture Content Below

- Offline search for many queries with two trees
- Special searching – ray maps etc.
- Data interpolation

Even Faster: Aggregate Searching with Two Trees (several queries at once)



- splitting planes of the spatial kd-tree
- first intersected plane <=> start node
- density estimation point (ep)
- photon hit point
- leaf node containing ep
- searched leaf
- start node for individual queries
- **l** left child , **r** right child

Idea behind Aggregate Searching

- Idea: put similar queries together into one larger query
- Evaluate the big query by traversing the tree
- Limitations:

  you have to know all the queries in advance or the subsequent queries have to be similar (=coherent)
- Properties: depending on the implementation and size of the problem you can reach the speedup in practice between 4 to 8.

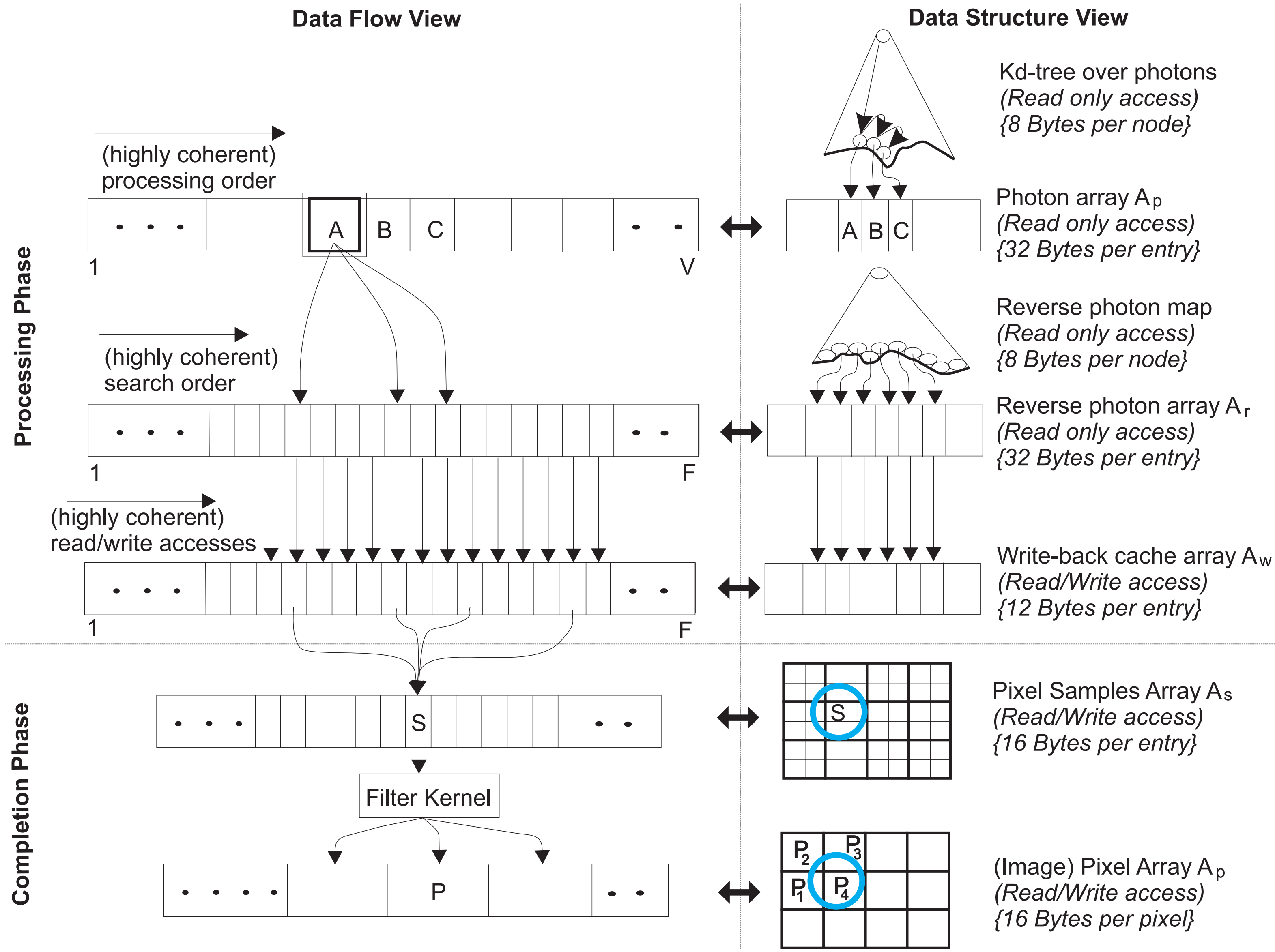
The Method of Two Trees: Offline Search

- Only if we know all the queries in advance before the first query is asked (**=offline searching**). We have to compute some incidence operations: NN-search or range-search.
- We need two trees:
  – Tree over the data (to be queried)
  – Tree over the queries (the second tree)
- We traverse the tree over the queries and compute the results in the tree over the data
  – The second tree provides a coherence for the data access and it can be significantly faster
  – It requires to store all the data: higher memory consumption
  – If the number of queries is larger than the number of data, then if it is possible, exchange the role of the data and queries

Algorithm Overview

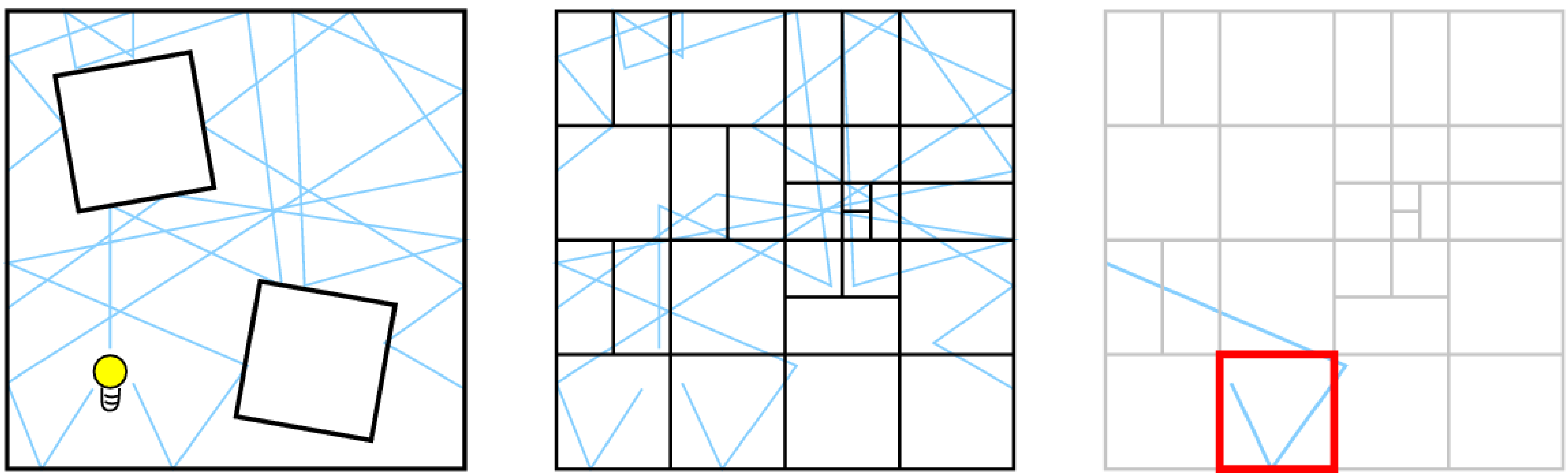
- Construct tree TD over the data
- Construct tree TQ over the queries
- Form an aggregate query AG by adding still unprocessed queries until the size of the aggregate query reaches a limit (size, number of points) – requires DFS trough TD
- Process AG by traversing TQ – if both children of an interior node should be processed:
  - (A) Subdivide AG into two smaller queries
  - (B) Process the individual queries

Two Trees Searching for Photon Mapping



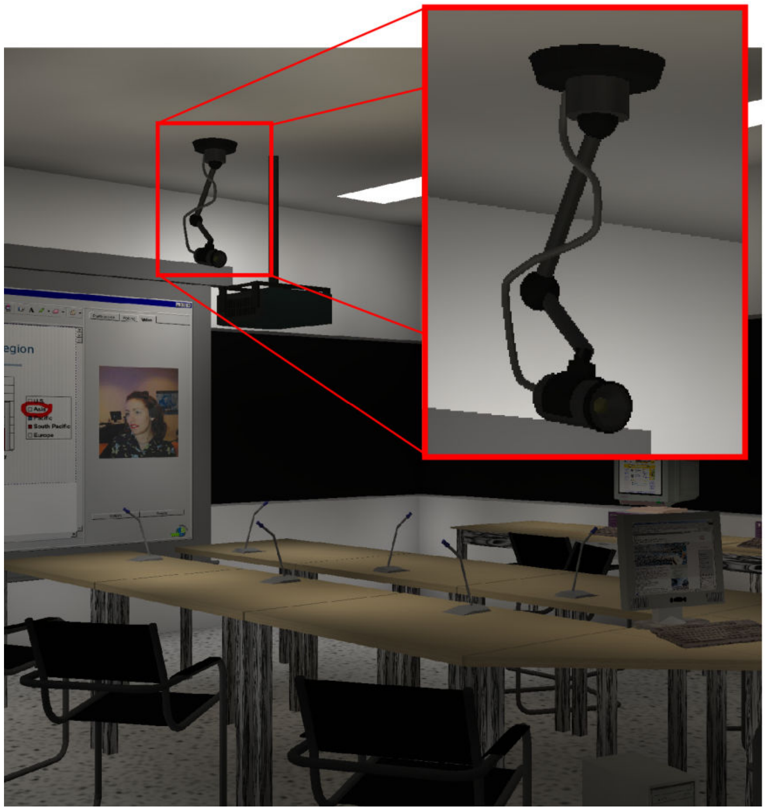
Raymaps – Data Structure for Rays

- We organize whole line segments (rays) in the kd-tree instead of storing points (photons)
- It requires lazily update/reconstruction of the kd-tree based on the coherent(=similar) queries
- It needs more memory than photon maps

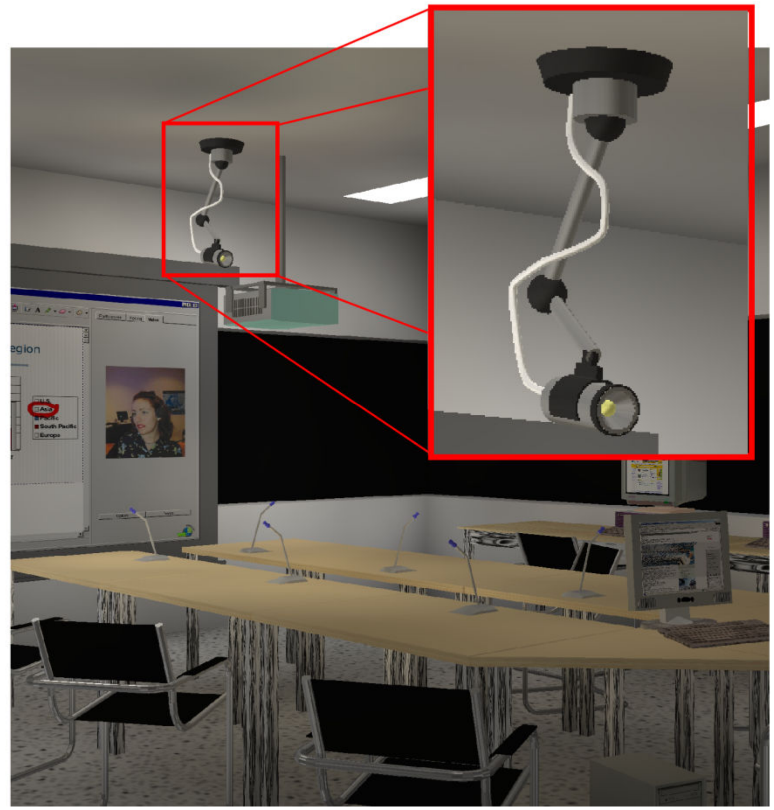


Raymaps: Results for Direct Visualization

Photon Maps



Ray Maps



Data Interpolation

- Input: a set of vectors in R^n
- Output: interpolated vector in R^n
- There exist many interpolation methods based on different principles
- The implementation via searching:
  - Shepard's method + Renka's
  - Locally Supported Radial Basis Functions (RBFs)

Shepard’s Algorithms for 2D and 3D

- We have M points (x_r, y_r, z_r), r in <1, M>
- Interpolation in 3D is provided by formula:

$$Q(x,y,z) =$$

$$[Sum_{i=1,m}\ w_r (x,y,z) * q_r\ ] / [Sum_{i=1,m}\ w_r (x,y,z)],$$

where

$$w_r (x,y,z) = (1/d_r(x,y,z))^2,$$

$$d_r(x,y,z)^2 = (x - x_r)^2 + (y - y_r)^2 + (z - z_r)^2$$

Note: we take all data for each point, even if they are far away from (x,y,z)

Renka's method (1988)

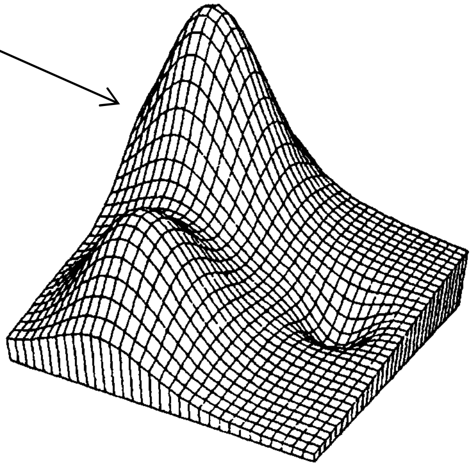
- Localization of Shepard's method
- Interpolation properties are better
- We take circular range search with the radius R_w

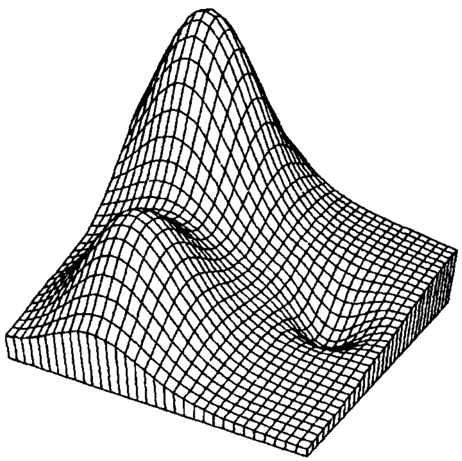
$$w_r(x,y,z) = [(R_w - d_r(x,y,z))_+ / (R_w . d_r(x,y,z))]^2,$$

$$(R_w - d_{r,k}(x,y,z))_+ = \begin{cases} R_w - d_r(x,y,z) & \text{if } d_r(x,y,z) < R_w \\ 0 & \text{if } d_r(x,y,z) > R_w \end{cases}$$

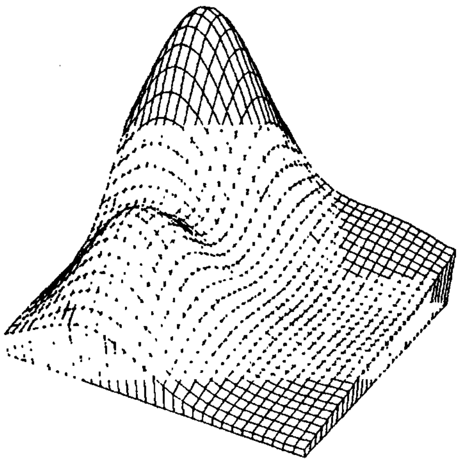
Example and Testing of Interpolation

- **Golden Standard method**: define the function F, select some points, and try to recover the function – you can compare the result with ground truth = the function F
- Compute maximum error, root mean square error, etc. between interpolated results and function F to evaluate interpolation quality

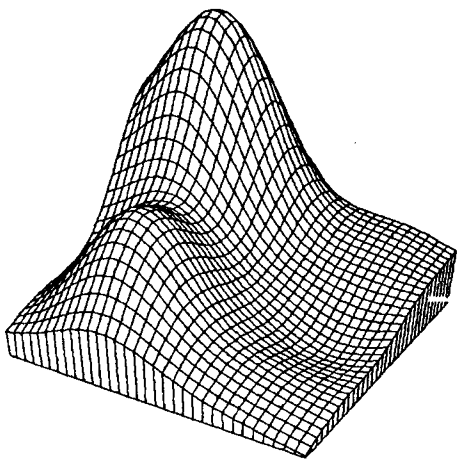




100 point samples



33 point samples



25 point samples

Example for 2D – Franke's function for 2D functions used to test interpolation

- $F1(x,y) = 0.75 * \exp(-((9x-2)^2+(9y-2)^2)/4)$

  $+ 0.75 * \exp(-(9x+2)^2/49-(9y+1)/10)$

  $+ 0.50 * \exp(-((9x-7)^2 + (9y-3)^2/4)$

  $- 0.20 * \exp(-(9x-4)^2 – (9y-7)^2)$

Note: it can be used to generate the data for homework: ray tracing height fields.

More in the paper: Renka+Brown, 1999, Algorithm 792: Accuracy Tests of ACM….

Thank you for your attention!