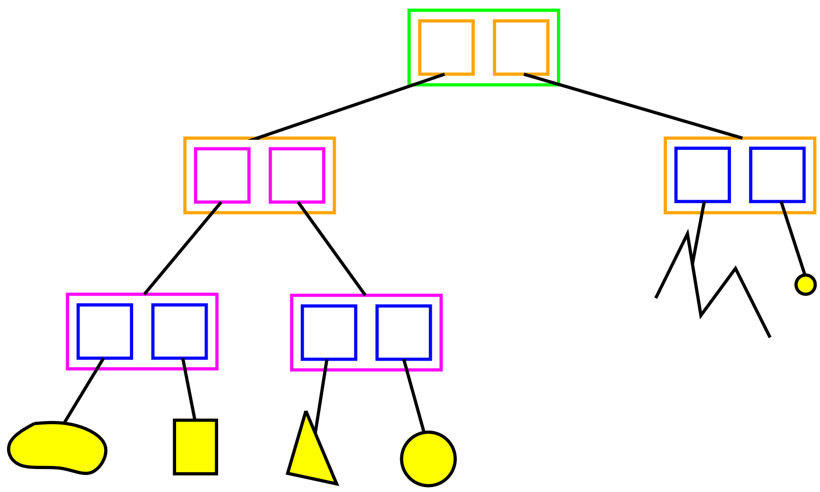
Box-Trees and R-Trees with Near-Optimal Query Time

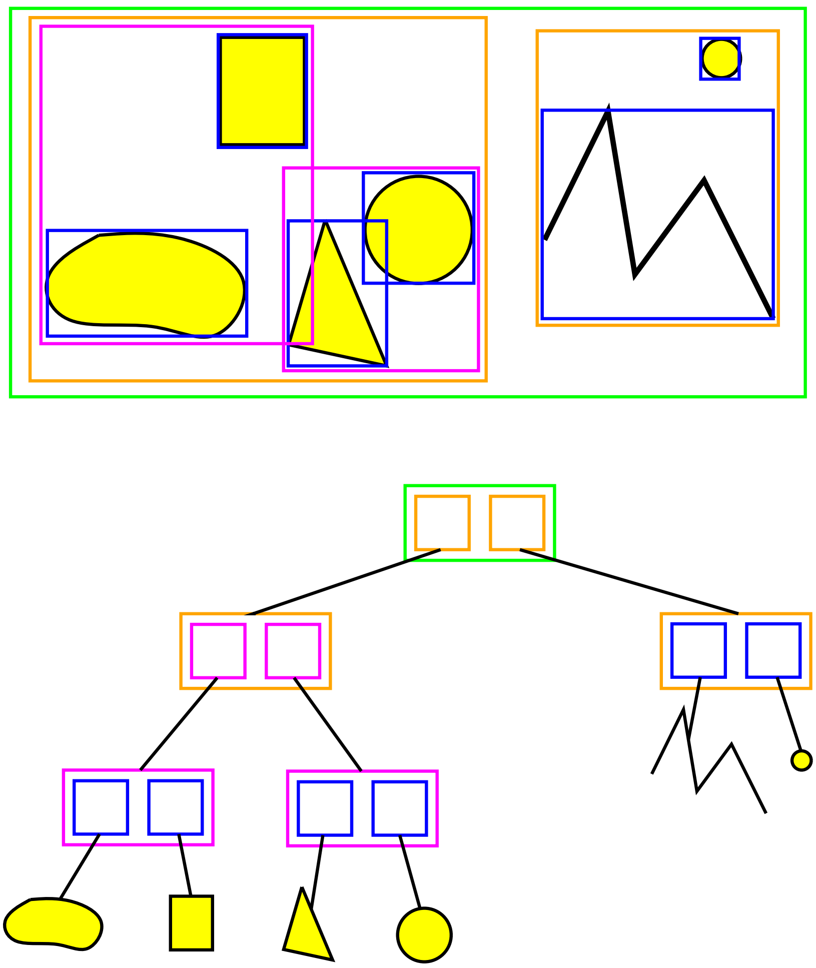
Pankaj Agarwal - Duke University
Mark de Berg - Utrecht University
Joachim Gudmundsson - Utrecht University
Mikael Hammar - Lund University
Herman Haverkort - Utrecht University



Box-Trees

- each leaf nodes stores a geometric object
- each internal node:
  - has two children (or: $O(1)$ children)
  - stores for each child the bounding box of all objects in the child's subtree

2D Example:



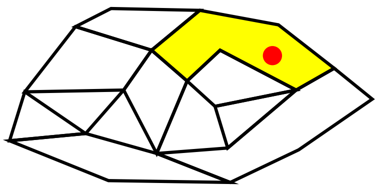
Applications

Box-trees store geometric data (2D, 3D, higher-D): maps, CAD-models, etc.

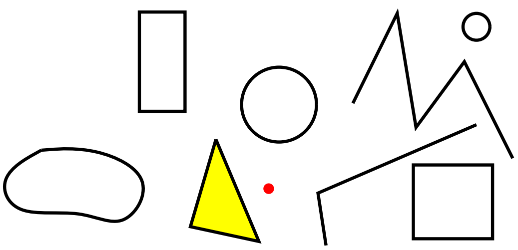
Applications in:

- geographic information systems (e.g. point location)
- computer graphics (e.g. visibility queries)
- virtual reality (e.g. collision detection)
- robotics
- motion planning

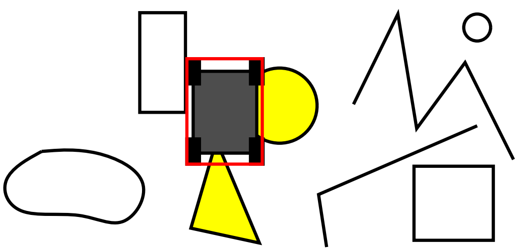
Examples:



Point location



Nearest neighbour



Collision detection or
Range searching

Pros and cons

Advantages:

- *low storage costs*
  GIS-databases and CAD-models can be very large – storage efficiency is critical; constants matter
- *simple to implement*
- *flexible*
  In many applications, many different types of objects must be stored and different types of queries are done
- *usually good performance in practice*

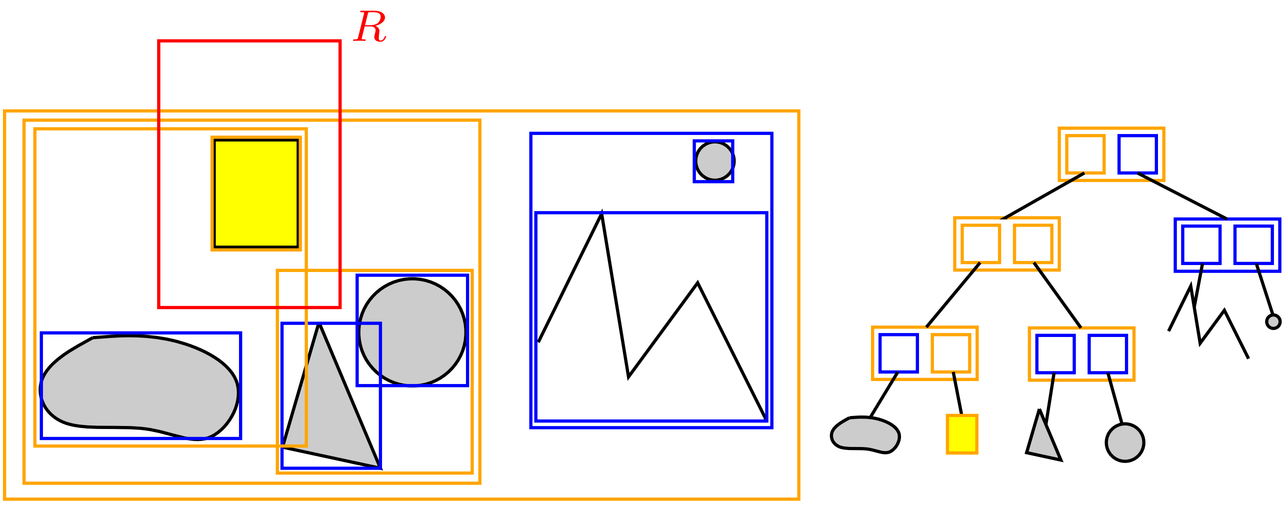
Disadvantages:

- *no guarantee on performance*
  query time depends on the way the tree is built – little theoretical work has been done about efficient constructions

Rectangle-Intersection Queries

Report all objects intersecting query rectangle R:

1. Check the bounding boxes stored at the root to see if they intersect R;
2. For each bounding box that intersects R, recursively visit the corresponding subtree – if that is a leaf, check the corresponding object and report if it intersects R.



Running time:

$\approx$ number of nodes visited

$=$ number of bounding boxes intersecting R.

Known results

$n =$ total number of input rectangles
(object bounding boxes) in box-tree

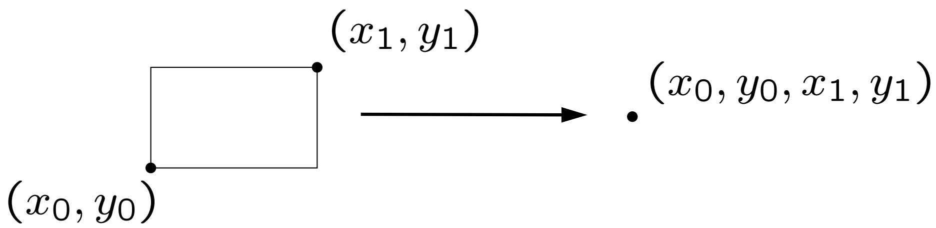
$k =$ number of input rectangles intersected by R

Lower bounds:

- De Berg et al. (2000):
  - input: disjoint unit cubes in d dimensions
  - query ranges: very thin/flat rectangles
  - bound: $\Omega(n^{1-1/d} + k)$

Upper bounds:

- A box in d dimensions can be represented by a point in $2d$-dimensional 'configuration space'.



  Determine which boxes are grouped together by partitioning the representative points using a kd-tree. Result: $O(n^{1-1/(2d)} + k)$

- De Berg et al. (2000):
  - input: rectangles in 2D
  - query range: rectangle with relative width w
  - bound: $O(\log^2 n + (w + k) \log n)$
    ($\Theta(n)$ in the worst case)

Our contribution

Lower bounds:

- $\Omega(n^{1-1/d} + k)$ also in the following case:
  - input: intersecting almost-unit-almost-cubes in $d \geq 2$ dimensions
  - query ranges: points
- $\Omega(n^{1-1/d} + k)$ also in the following case:
  - input: disjoint almost-unit-almost-cubes in $d \geq 3$ dimensions
  - query ranges: cubes

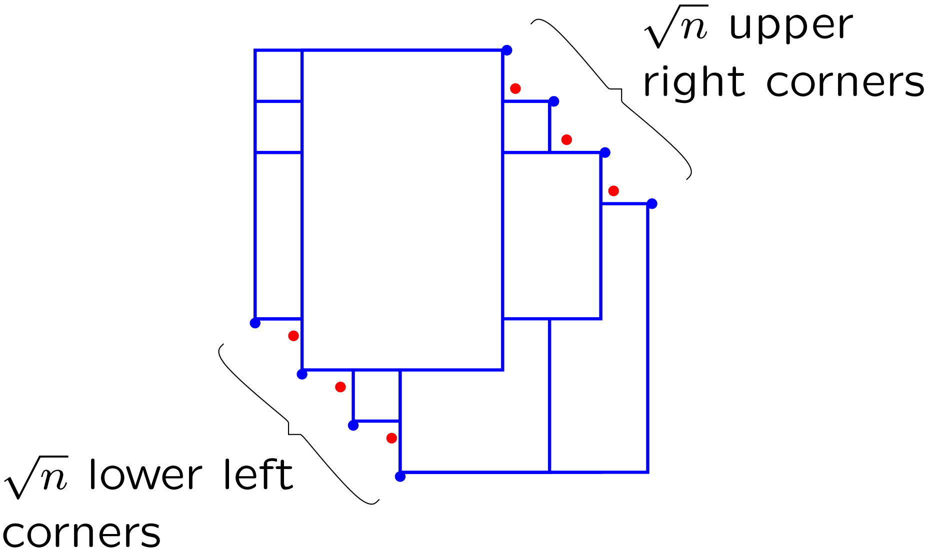
Upper bounds:

- Better analysis of configuration space approach: $O(n^{1-1/d} + k \log n)$ for point and rectangle queries
- After small modification of the construction: $\Theta(n^{1-1/d} + k) =$ optimal
- New construction for (almost) disjoint input in 2D:
  $O(\sqrt{n} \log n + k)$ for rectangle queries
  $O(\log^2 n)$ for point queries
- Variant of this construction:
  $O(\log^2 n + k)$ for queries with rectangles of bounded aspect ratio

Lower bound intersecting input

Theorem: for all n, there is a set of almost-unit-squares in 2D such that in any box-tree on this set, a point query with result $\emptyset$ takes $\Omega(\sqrt{n})$ time in the worst case.

Proof:



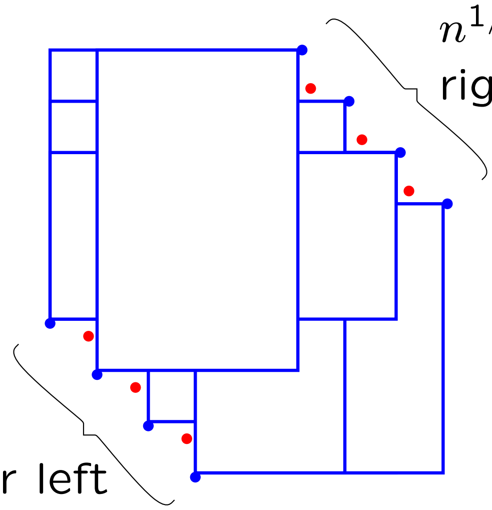
input boxes are all combinations of lower left corner with upper right corner (n boxes)

- any box-tree has $\Theta(n)$ bounding boxes of pairs
- each intersects one of $O(\sqrt{n})$ query points
- at least one query point gets $\Omega(\sqrt{n})$ intersections

Generalises to higher dimensions: $\Omega(n^{1-1/d})$

Lower bound disjoint input

Lower bound holds also for disjoint input in 3D:
start with 2D-construction on $n^{2/3}$ almost-squares.



use 3rd dimension to make disjoint almost-cubes
(query points become edges of large cubes),
line up $n^{1/3}$ such sets with query points in between

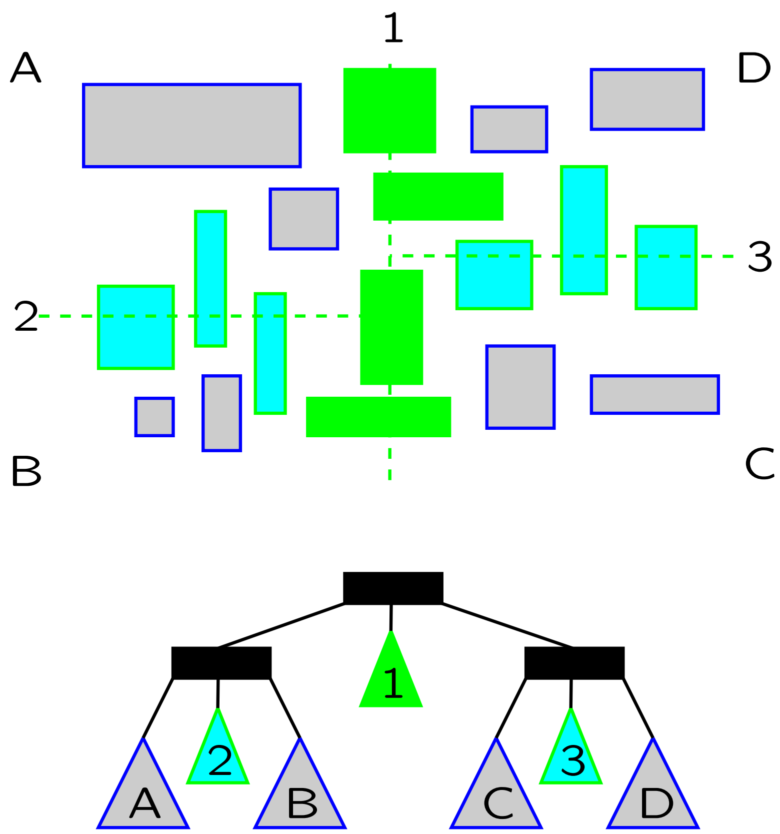
$\Longrightarrow$ each of $\Theta(n)$ internal boxes intersects one of $O(n^{1/3})$ query points/cubes $\Longrightarrow$ $\Omega(n^{2/3})$ query time

Result:

- shows polylogarithmic point-query times are impossible without near-linear range-query time
- generalises to higher dimensions: $\Omega(n^{1-1/d})$
- does not work in 2D

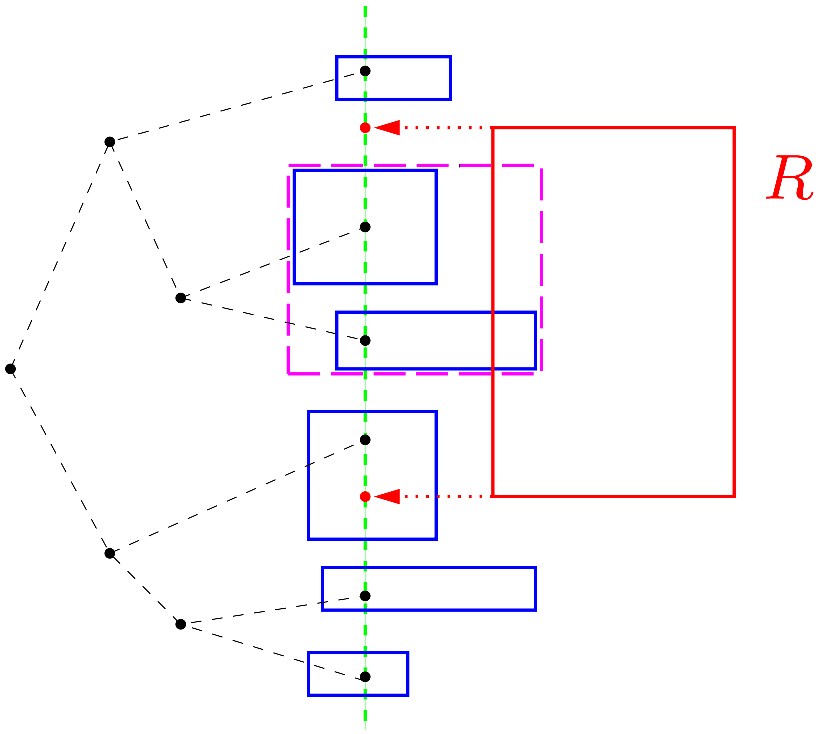
Kd-Interval-Trees

- on each level, cut such that at most half of the input lies to one side, at most half lies to the other side
- store each side recursively
- store intersected boxes in separate substructures
- cut vertical on every odd level, horizontal on every even level



Kd-Interval-Trees: substructures

Substructures for boxes intersected by a cutting line: a binary tree on the order along the line



Analysis for search with query rectangle R:

- $O(\log n)$ bounding boxes may contain an endpoint of R's projection on the cutting line
- A bounding box in between the endpoints only intersects R if there is a leaf node to be reported in its subtree: $O(k \log n)$ bounding boxes

Total: $O(\log n + k \log n)$

Kd-Interval-Trees: query time

Analysis for the complete structure (rectangle query):

- known about kd-trees: only $O(\sqrt{n})$ kd-tree cells may intersect the boundary of a rectangle
- for each of them, spend $O(\log n + k' \log n)$ in the associated "intersected substructure", where $\sum k' = k$

Total: $O(\sqrt{n} \log n + k \log n)$

Analysis for the complete structure (point query):

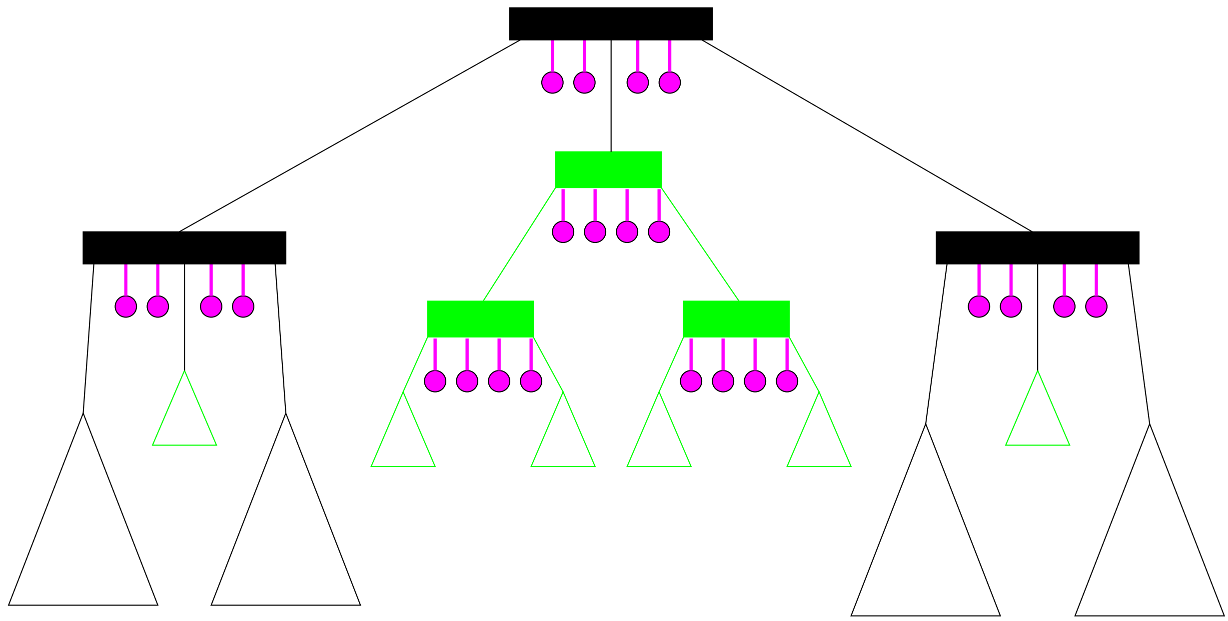
- $O(\log n)$ cells may be visited
- spend $O(\log n)$ in each "intersected substructure"

Total: $O(\log^2 n)$

Priority nodes

(like a priority search tree)

In each subtree, store the leftmost, rightmost, topmost and bottommost input objects as priority leaves directly under the root.



Effect for rectangle queries:

- search time substructures improves to $O(\log n + k \log n)$
- total search time improves to $O(\sqrt{n} \log n + k \log n)$

Conclusions

Results:

- lower bounds that hold with “normal” query ranges
- an easy construction which achieves optimal query time for range searching in box-trees on overlapping input in any number of dimensions
- an easy construction which achieves near-optimal query time for range and point searching in box-trees on disjoint input in 2D
- generalisations of the bounds and efficient conversions to R-trees

Open problems:

- Why do bounding volume hierarchies seem to work well in practice, despite bad bounds?
  - analysis under realistic constraints on input?
  - analysis for approximate range searching?
- How do our box-tree constructions compare to known heuristic approaches?
- How to deal with insertions and deletions?