R-trees:
external-memory data structures
for searching geometric objects

Herman Haverkort — University of Aarhus
Window queries
Window queries
Filtering approach: first search bounding boxes…
Window queries

Filtering approach: first search bounding boxes…
Window queries

Filtering approach: first search bounding boxes...
Window queries

Filtering approach: first search bounding boxes...
Filtering approach: first search bounding boxes... then check objects one by one.
Window queries

Filtering approach: first search bounding boxes... then check objects one by one.

Worst-case-efficient? In terms of query objects, not really...

Analysis on level of bounding boxes still useful. 
\( N \) rectangles, \( T \) answers, block size \( B \)

Goal: visit \( O(sublinear(N/B) + T/B) \) blocks.
Window queries with bounding volume hierarchies
Window queries with bounding volume hierarchies
Window queries with bounding volume hierarchies
Window queries with bounding volume hierarchies
Window queries with bounding volume hierarchies
Window queries with bounding volume hierarchies
R-tree: Group data into nodes of size $B$:
- Leaves contain $\Theta(B)$ objects’ bounding boxes (example: 2 each);
- Internal nodes have $\Theta(B)$ children (example: 4).
Window queries with bounding volume hierarchies

R-tree: Group data into nodes of size $B$:
- Leaves contain $\Theta(B)$ objects’ bounding boxes;
- Internal nodes have $\Theta(B)$ children.
Window queries with bounding volume hierarchies

R-tree: Group data into nodes of size $B$:

- Leaves contain $\Theta(B)$ objects’ bounding boxes;
- Internal nodes have $\Theta(B)$ children. Each node stores the bounding box of each of its children.
Lower bounds

Special case: objects are points.
R-trees use only linear space, so $\Omega(\sqrt{N/B} + T/B)$ worst-case lower bound.

Special case: queries are points.
Also $\Omega(\sqrt{N/B} + T/B)$.

General case: objects and queries are rectangles.
$\Omega(\sqrt{N/B} + T/B)$. 
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Order the rectangles by the position of their centers along the curve
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Order the rectangles by the position of their centers along the curve
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Order the rectangles by the position of their centers along the curve
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Order the rectangles by the position of their centers along the curve
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Order the rectangles by the position of their centers along the curve
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Order the rectangles by the position of their centers along the curve
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Order the rectangles by the position of their centers along the curve
Order the rectangles by the position of their centers along the curve.
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Order the rectangles by the position of their centers along the curve;
Put a B-tree on top.

Variant: order by points \((x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})\) along 4D curve.

Building in \(O\left(\frac{N}{B} \log \frac{M}{B} \frac{N}{B}\right)\) I/O’s
Inserts and deletes in \(O(\log_B \frac{N}{M})\) I/O’s.
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Query efficiency?
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Query efficiency?
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Query efficiency?
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Query efficiency?
R-tree heuristics: Hilbert R-tree [Kamel & Faloutsos]

Query efficiency?

$\Theta(N/B)$ nodes visited, while $T = 0$. 
R-tree heuristics: Top-down Greedy Split [García et al.]

Build R-tree bottom-down by recursively dividing input into two.

Make nodes of degree $B$ by combining binary cuts.

How to choose a binary cut?

- Consider the four orderings by $x_{\text{min}}, y_{\text{min}}, x_{\text{max}}$ and $y_{\text{max}}$.
- For each ordering, consider $B$ cutting positions.
- Choose the cut that minimizes the sum of the areas of the two resulting bounding boxes.
R-tree heuristics: Top-down Greedy Split [García et al.]
R-tree heuristics: Top-down Greedy Split [García et al.]

$x_{\text{min}}$-cuts
R-tree heuristics: Top-down Greedy Split [García et al.]
R-tree heuristics: Top-down Greedy Split [García et al.]

$x_{\text{min}}$-cuts

\[
\frac{N}{B} \quad \frac{N}{B} \quad \frac{N}{B}
\]
R-tree heuristics: Top-down Greedy Split [García et al.]

- $y_{\text{max}}$-cuts
- $x_{\text{max}}$-cuts
- $y_{\text{min}}$-cuts
- $x_{\text{min}}$-cuts
R-tree heuristics: Top-down Greedy Split [García et al.]

Continue until all sets have size $N/B$

size $N/B$ already:
no recursion
Continue until all sets have size $N/B$

size $N/B$ already:
no recursion
R-tree heuristics: Top-down Greedy Split [García et al.]

Continue until all sets have size $N/B$

one node:

Build the subtrees for the children recursively
R-tree heuristics: Top-down Greedy Split [García et al.]

Continue until all sets have size $N/B$

Build the subtrees for the children recursively

Worst-case building time: $O\left(\frac{N}{B} \log_2 N\right)$
Worst-case query time: $\Theta\left(N/B\right)$
Update time: use logarithmic method?
R-trees with worst-case-optimal query time

PR-tree and pseudo-PR-tree.
Building P-trees: step 1
Building P-trees: step 1

Make leftmost rectangle a child of the root
Building P-trees: step 1

For each direction, make most extreme rectangle a child (priority box) of the root.
Building P-trees: step 1

For each direction, make most extreme rectangle a child (priority box) of the root.
Building P-trees: step 1

For each direction, make most extreme rectangle a child (priority box) of the root.
Building P-trees: step 2

Divide remaining rectangles by the $x_{\text{min}}$-coordinates...
Building P-trees: step 2

Build subtrees recursively, dividing by $x_{\text{min}}, y_{\text{min}}, x_{\text{max}}$ and $y_{\text{max}}$-coordinates in a round-robin fashion, like in a kd-tree).
A window query visits two types of nodes.

**Type A:** children of nodes that have (at least) one priority box reported as an answer

At most $T$ priority boxes reported (by definition) $\rightarrow O(T)$ such children.

**Type B:** children of nodes that have no priority box reported as an answer

A bit more difficult...
Analysing case B (children of nodes that have no priority box reported as an answer)

A query window $Q$, and the four P-boxes of a node $\nu$:

Each P-box separated from $Q$ by line through edge of $Q$, e.g.:
$A$ is separated from $Q$ since $x_{\text{max}}(A) < x_{\text{min}}(Q)$. 

Dividing by $x_{\text{min}}, y_{\text{min}}, x_{\text{max}}$ and $y_{\text{max}}$-coordinates in a round-robin fashion is like building a kd-tree on 4D points $(x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})$:

$A$ is separated from $Q$ since $x_{\text{max}}(A) < x_{\text{min}}(Q)$, so in 4D: the point for $A$ lies on the low side of the hyperplane $x_{\text{max}} = x_{\text{min}}(Q)$. 

Analysing case B (children of nodes that have no priority box reported as an answer)
Analysing case B (children of nodes that have no priority box reported as an answer)

A lies on the low side of the hyperplane $x_{\text{max}} = x_{\text{min}}(Q)$.

Node $\nu$ must also have a P-box $B$ with $x_{\text{max}}(B) > x_{\text{min}}(Q)$, otherwise we would not visit this node.

Thus: $\nu$’s cell in the kd-tree, is cut by 3D-hyperplane $x_{\text{max}} = x_{\text{min}}(Q)$. 
Analysing case B (children of nodes that have no priority box reported as an answer)

$B$ is also separated from $Q$, by another line through an edge of $Q$:

Because of $A$, $\nu$'s kd-cell is cut by hyperplane $x_{\text{max}} = x_{\text{min}}(Q)$.
Because of $B$, $\nu$'s kd-cell is cut by hyperplane $y_{\text{min}} = y_{\text{max}}(Q)$.

Fact: in a 4D kd-tree, only $O(\sqrt{N})$ cells can simultaneously intersect two orthogonal 3D-hyperplanes.
Querying P-trees

A window query visits two types of nodes.

**Type A: children of nodes that have (at least) one priority box reported as an answer**

At most $T$ priority boxes reported (by definition) $\rightarrow O(T)$ such children.

**Type B: children of nodes that have no priority box reported as an answer**

Dividing by $x_{\min}$, $y_{\min}$, $x_{\max}$ and $y_{\max}$-coordinates in a round-robin fashion is like building a kd-tree on 4D points $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$. The kd-cell of a type-B node simultaneously intersects two orthogonal 3D-hyperplanes in 4D space (out of four hyperplanes defined by $Q$). Fact: in a 4D kd-tree, there can be only $O(\sqrt{N})$ such cells.
Querying P-trees

A window query visits two types of nodes.

**Type A:** children of nodes that have (at least) one priority box reported as an answer

At most $T$ priority boxes reported (by definition) $\rightarrow O(T)$ such children.

**Type B:** children of nodes that have no priority box reported as an answer

By analysing the kd-tree in 4D: $O(\sqrt{N})$ such children.

**Total:** $O(\sqrt{N} + T)$
Making P-trees I/O-efficient: step 1
Making P-trees I/O-efficient: step 1

Make leftmost rectangle a child of the root
Making P-trees I/O-efficient: step 1

Make leftmost rectangle $B$ leftmost rectangles a child of the root...
Making P-trees I/O-efficient: step 1

Make leftmost rectangle $B$ leftmost rectangles a child of the root and store bounding box in the root.
Making P-trees I/O-efficient: step 1

For each direction, put $B$ most extreme rectangles in a priority leaf.
For each direction, put $B$ most extreme rectangles in a priority leaf.
Making P-trees I/O-efficient: step 1

For each direction, put $B$ most extreme rectangles in a priority leaf.
Making P-trees I/O-efficient: step 2

Divide remaining rectangles by the $x_{\text{min}}$-coordinates...
Making P-trees I/O-efficient: step 2

Build subtrees recursively, dividing by $x_{\text{min}}$, $y_{\text{min}}$, $x_{\text{max}}$ and $y_{\text{max}}$-coordinates in a round-robin fashion, like in a kd-tree).
Making P-trees I/O-efficient: analysis

A window query visits two types of nodes.

**Type A:** children of nodes that have (at least) one priority box all rectangles in (at least) one priority leaf reported as answers  

$T$ answers fill at most $T/B$ priority leaves $\rightarrow O(T/B)$ such children.

**Type B:** children of nodes that have no priority box leaf completely reported as answers

By analysing the kd-tree in 4D (which has only $N/B$ leaves): $O(\sqrt{N/B})$ such children.

**Total:** $O(\sqrt{N/B} + T/B)$
Analysing case B (children of nodes that have no priority box reported as an answer)

A query window $Q$, and the four P-boxes of a node $\nu$. From each P-leaf a box which is not an answer:

Each such box separated from $Q$ by line through edge of $Q$, e.g.: $A$ is separated from $Q$ since $x_{\text{max}}(A) < x_{\text{min}}(Q)$. 

$x_{\text{max}}(A)$ and $x_{\text{min}}(Q)$ refer to the maximum and minimum x-coordinates of the boxes $A$ and $Q$, respectively.
A window query visits two types of nodes.

Type A: children of nodes that have (at least) one priority box all rectangles in (at least) one priority leaf reported as answers

$T$ answers fill at most $T/B$ priority leaves $\rightarrow O(T/B)$ such children.

Type B: children of nodes that have no priority box leaf completely reported as answers

By analysing the kd-tree in 4D (which has only $N/B$ leaves): $O(\sqrt{N/B})$ such children.

Total: $O(\sqrt{N/B} + T/B)$
Building PR-trees efficiently

Like a 4-dimensional kd-tree.

Sort into four lists, one for each coordinate.

Build $\Theta(M^{1/4})$ nodes = $\Theta(\log M)$ levels at a time.

In each round, add a filtering step:

Each node gets four priority leaves of size $O(B)$; filter boxes through the tree—the extreme boxes will be ”caught” by the priority leaves.
Experiments

Implemented with Duke’s TPIE (Transparent Parallel I/O-Environment):

- 2D Hilbert R-tree
- 4D Hilbert R-tree
- Top-down Greedy Split (TGS)
- our Priority R-tree

Tested with:

- Real-life data: TIGER road map data:
  up to 17 mln bounding boxes of road line segments;
- Synthetic data: 10 mln random rectangles,
  uniformly distributed in unit square,
  with varying parameters
Cartographic data

Data: road segments
Queries: squares with areas from 0.25% to 2%

Shown: Number of I/Os spent in answering a query $T/B$
Random rectangles of variable size

Data: sides of rectangles uniformly distributed in $[0, max\_side]$
Queries: squares with area 1%
Random boxes of variable shape

Data: randomly oriented rectangles with area $10^{-6}$ and aspect ratio $a$

Queries: squares with area 1%
Random boxes with variable distribution

Data: uniformly distributed points, then transform \((x, y)\) to \((x, y^c)\)
Queries: squares with area 1\%, transformed in the same way
“Worst-case” random boxes

Data: 10 000 clusters of height $10^{-5}$ on a horizontal line
Queries: rectangles of width 1, height $10^{-7}$, through all clusters

<table>
<thead>
<tr>
<th>tree:</th>
<th>Hilbert 2D</th>
<th>Hilbert 4D</th>
<th>PR-tree</th>
<th>TGS-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td># I/Os:</td>
<td>32 920</td>
<td>83 389</td>
<td>1 060</td>
<td>22 158</td>
</tr>
<tr>
<td>% of the R-tree visited:</td>
<td>37%</td>
<td>94%</td>
<td>1%</td>
<td>25%</td>
</tr>
</tbody>
</table>
In theory:

The PR-tree is the first R-tree variant that answers a window query in $O(\sqrt{N/B} + T/B)$ I/Os in the worst case.

In practice:

Window queries in real-life and relatively nicely distributed data: roughly the same as best previous R-trees.

Window queries in more extreme data: PR-trees outperforms best previous R-trees significantly.
What about...

...construction time?

tree : theory : on 10 mln boxes

Hilbert : $O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os : 5 min.

PR : $O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os : 20 min.

TGS : $O\left(\frac{N}{B} \log_2 \frac{N}{B}\right)$ I/Os (typically) : 60-240 min.

...update time?

Hilbert : like a B-tree: $O\left(\log_B \frac{N}{M}\right)$ I/Os

PR : similar to Hilbert, amortized

...higher dimensions?

PR-tree answers window query in $O\left((\frac{N}{B})^{1-1/d} + \frac{T}{B}\right)$ I/Os
## Still to do

<table>
<thead>
<tr>
<th>Problem</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gap ( \Omega((\frac{dN}{B})^{1-1/d} + \frac{dT}{B}) ) — ( O((\frac{dN}{B})^{1-1/d} + \frac{d^2T}{B}) )</td>
<td>other priorities</td>
</tr>
<tr>
<td>Make it cache-oblivous</td>
<td>getting close!</td>
</tr>
<tr>
<td>Speed up point queries theoretically</td>
<td>( \log_2 N \log_B N ) in disjoint data</td>
</tr>
<tr>
<td>Speed up point queries in practice</td>
<td>we’re puzzled</td>
</tr>
<tr>
<td>Experiment with updates</td>
<td>maybe later</td>
</tr>
<tr>
<td>Experiment with 3D</td>
<td>maybe later</td>
</tr>
<tr>
<td>Generalize to more flexible bounding volumes</td>
<td>almost there!</td>
</tr>
</tbody>
</table>
THAT’S ALL FOLKS

http://haverkort.net/herman/cs/