An edge quadtree for external memory

Mark McGranaghan Laura Toma

Department of Computer Science
Bowdoin College

NCGIA, University of Maine, March 2011

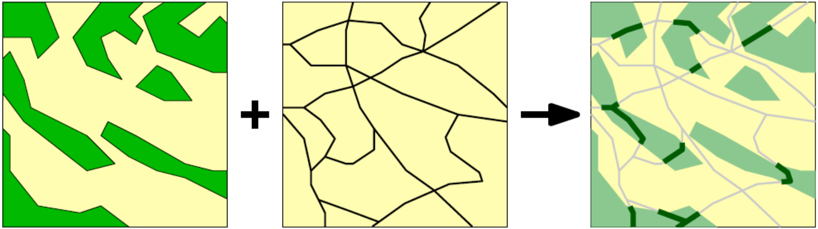
Outline

1. The problem and motivation
2. Quadtrees and Z-order
3. Our algorithm
4. Empirical evaluation

Outline

Map overlay

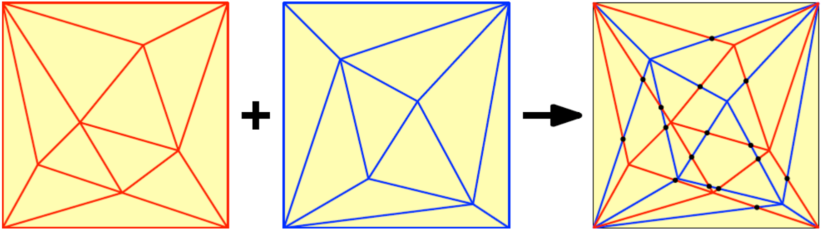
- Maps: planar subdivisions, sets of non-intersecting line segments, triangulations



figures thanks to H. Haverkort

Map overlay

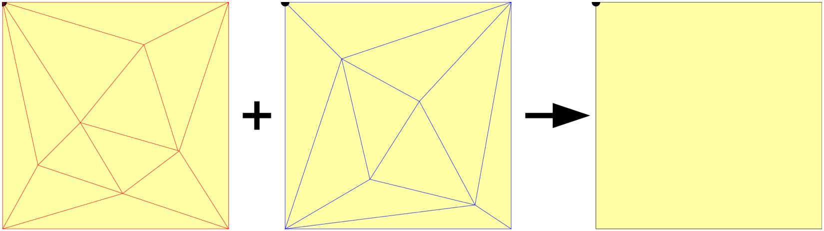
- Maps: triangulations



figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

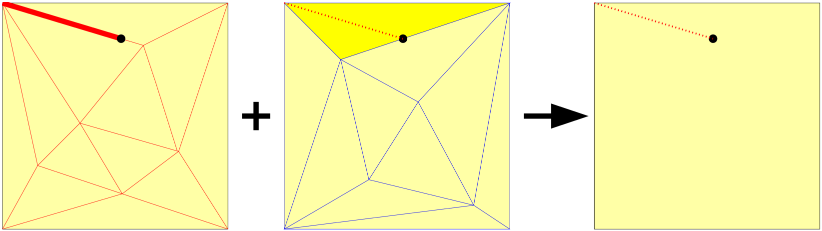


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

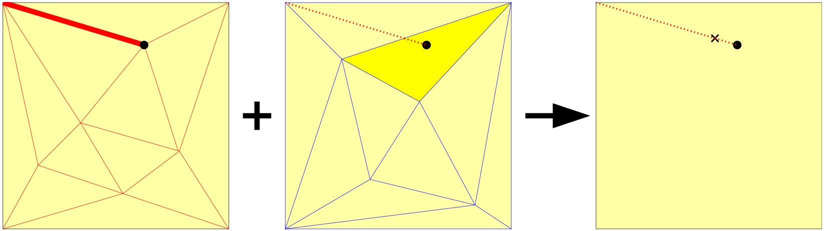


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

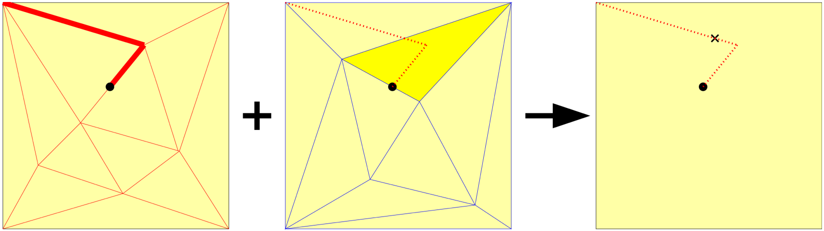


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

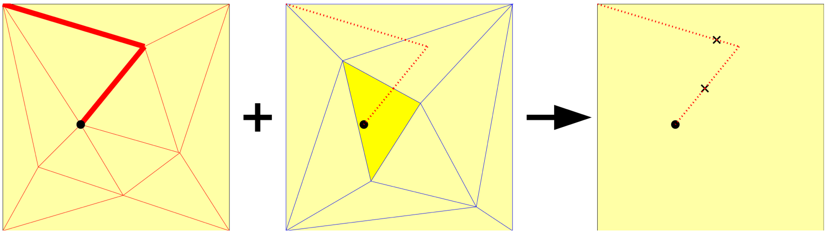


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

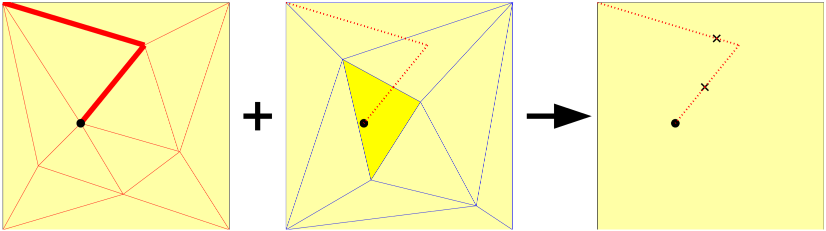


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

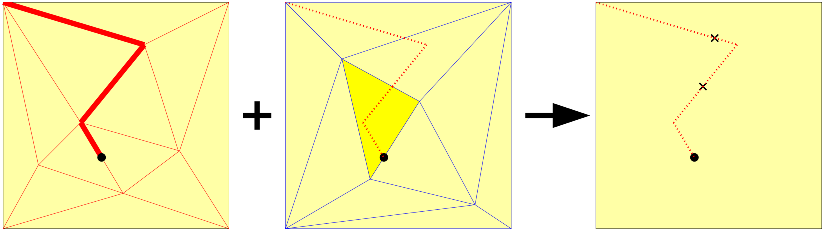


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

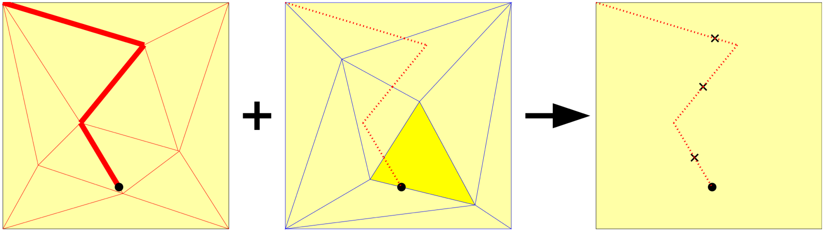


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

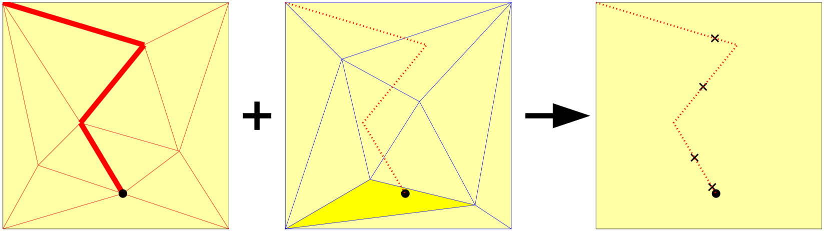


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

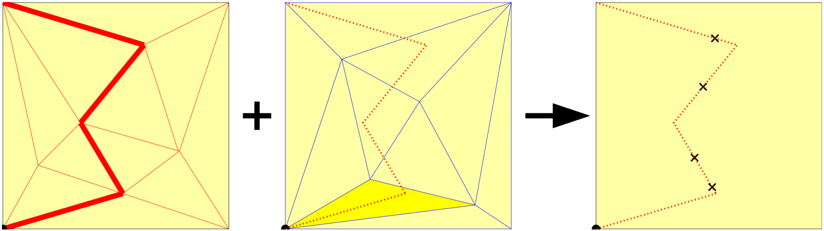


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

- Maps: triangulations

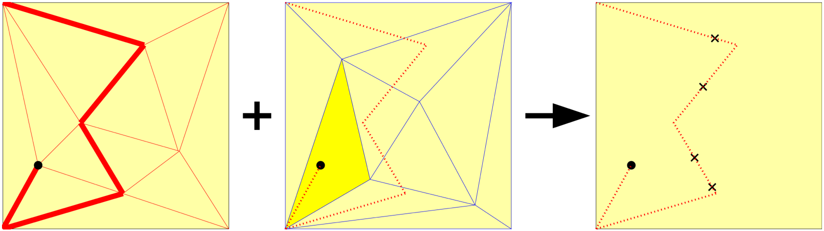


- DFS in one triangulation, traverse triangles in the other

figures thanks to H. Haverkort

Overlaying triangulations CPU-efficiently

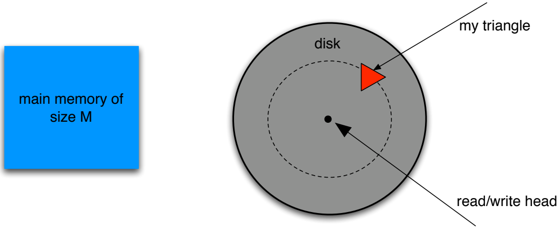
- Maps: triangulations



- DFS in one triangulation, traverse triangles in the other
  - $O(1)$ operations per edge, $O(1)$ operations per crossing
- Total: $O(n + k)$ CPU operations
  - for n triangles, k crossings

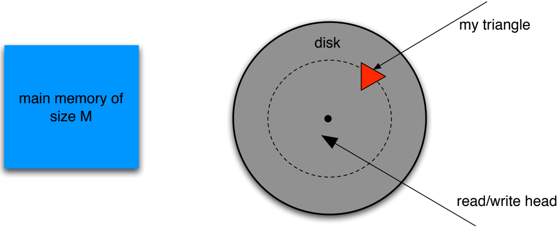
In external memory

- If main memory is too small to hold all data



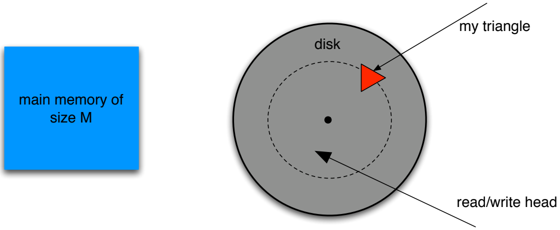
In external memory

- If main memory is too small to hold all data



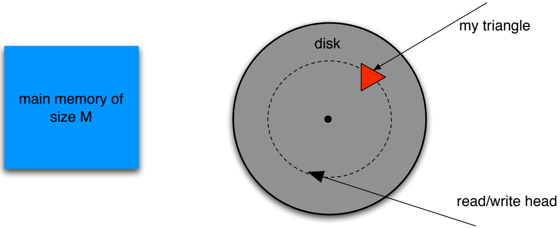
In external memory

- If main memory is too small to hold all data



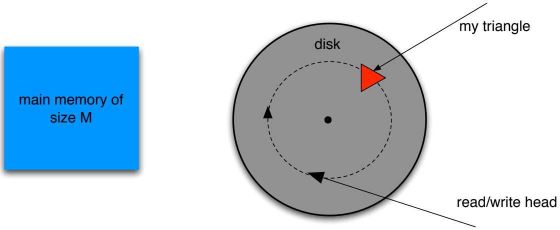
In external memory

- If main memory is too small to hold all data



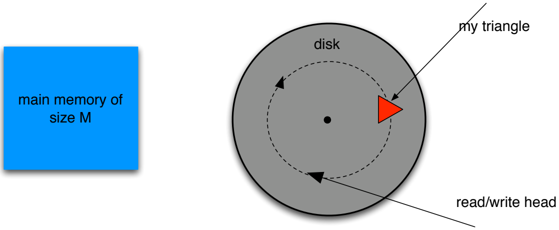
In external memory

- If main memory is too small to hold all data



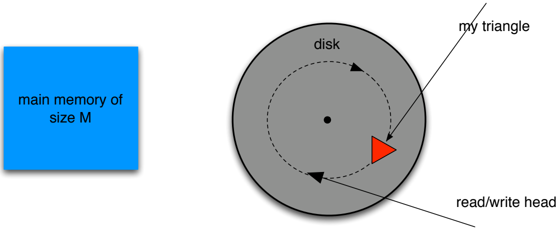
In external memory

- If main memory is too small to hold all data



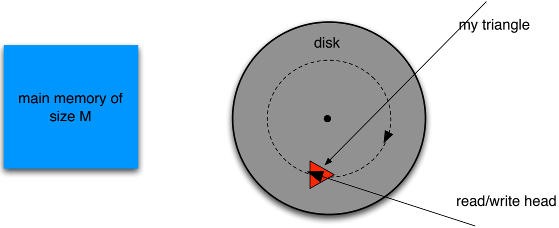
In external memory

- If main memory is too small to hold all data



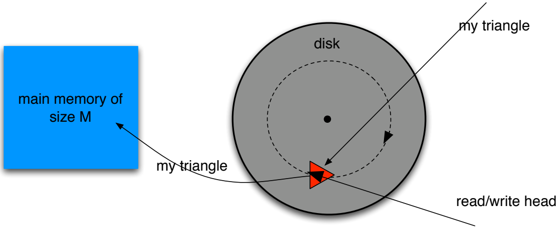
In external memory

- If main memory is too small to hold all data



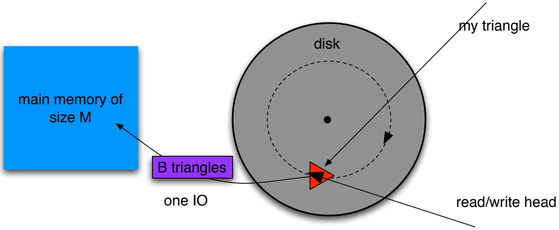
In external memory

- If main memory is too small to hold all data



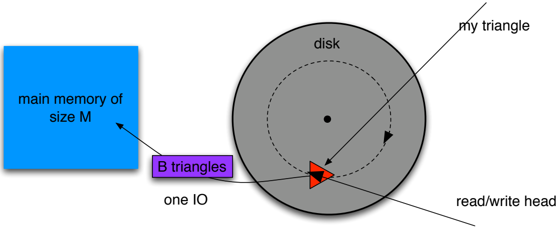
In external memory

- If main memory is too small to hold all data



In external memory

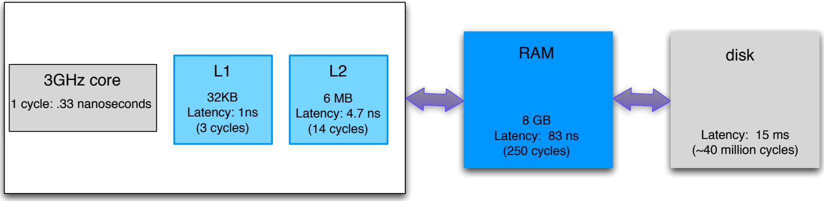
- If main memory is too small to hold all data



- The disk is 10^6 times slower than the memory
- B is big (8KB or more)
- With large data the bottleneck is usually the IO

What your computer does while you wait

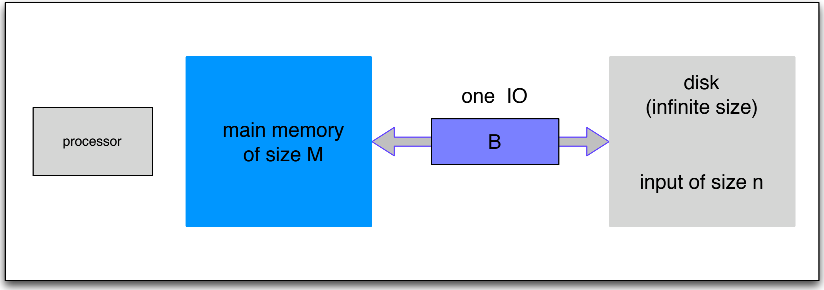
- Intel Core 2 Duo at 3.0GHz



`http://duartes.org/gustavo/blog`

...To put this into perspective, reading from L1 cache is like grabbing a piece of paper from your desk (3 seconds), L2 cache is picking up a book from a nearby shelf (14 seconds), and main memory is taking a 4-minute walk down the hall to buy a Twix bar. Waiting for a hard drive seek is like leaving the building to roam the earth for one year and three months [...]

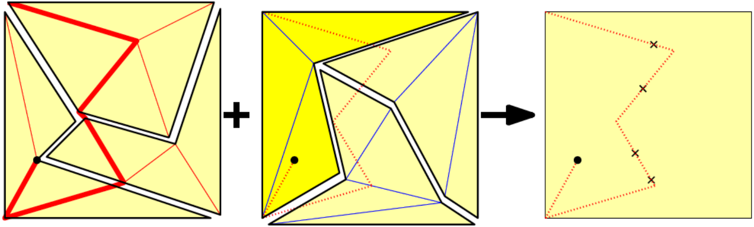
The IO-Model [Agarwal & Vitter, 1988]



- one IO $\approx 40,000,000$ CPU operations
- IO-complexity: number of IOs
- Fundamental bounds
  - scanning: $\texttt{scan}(n) = \frac{n}{B}$ IOs
  - sorting: $\texttt{sort}(n) = \Theta(\frac{n}{B} \log_{M/B} \frac{n}{M})$ IOs

Triangulation overlay IO-efficiently?

- The triangulation is on disk, arranged in blocks



- DFS in one triangulation, traverse triangles in the other
- CPU: $\Theta(n+k)$ operations
- IO:
  - one IO per edge and triangle
  - total: $O(n)$ IOs

figures thanks to H. Haverkort

IO-efficient map overlay: Related work

n = input size, M = memory size, B = disk block size

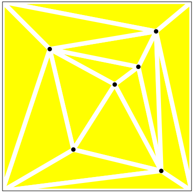
- Arge et al 1995: $O(\text{sort}(n) + k/B)$ IOs
  - complicated, super-linear space
- Crauser et al 2001: $O(\text{sort}(n) + k/B)$ IOs
  - randomized
- De Berg et al 2007: in $O(\text{sort}(\lambda n))$ IOs can build a data structure that supports map overlay in $O(\text{scan}(\lambda n))$ IOs
  - λ is the density of the set of segments (for any circle C, intersecting segments $> diam(C)$ is $O(\lambda)$).

Outline

Ingredients: quadtrees ...

Quadtree: divide unit square into quadrants, refine until amount of data per cell is small.

(for example: until every cell has at most one vertex)

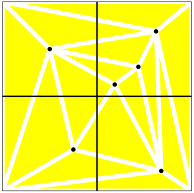


figures thanks to H. Haverkort

Ingredients: quadtrees ...

Quadtree: divide unit square into quadrants, refine until amount of data per cell is small.

(for example: until every cell has at most one vertex)

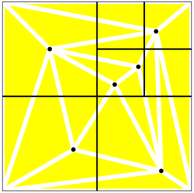


figures thanks to H. Haverkort

Ingredients: quadtrees ...

Quadtree: divide unit square into quadrants, refine until amount of data per cell is small.

(for example: until every cell has at most one vertex)

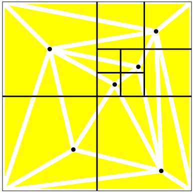


figures thanks to H. Haverkort

Ingredients: quadtrees ...

Quadtree: divide unit square into quadrants, refine until amount of data per cell is small.

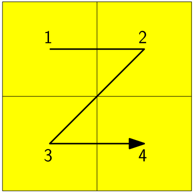
(for example: until every cell has at most one vertex)



figures thanks to H. Haverkort

Ingredients: ... and Z-order

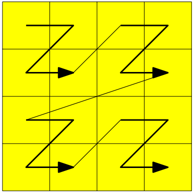
Z-order space-filling curve: visit quadrants recursively in order NW, NE, SW, SE



figures thanks to H. Haverkort

Ingredients: ... and Z-order

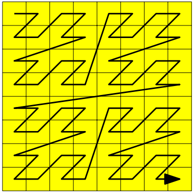
Z-order space-filling curve: visit quadrants recursively in order NW, NE, SW, SE



figures thanks to H. Haverkort

Ingredients: ... and Z-order

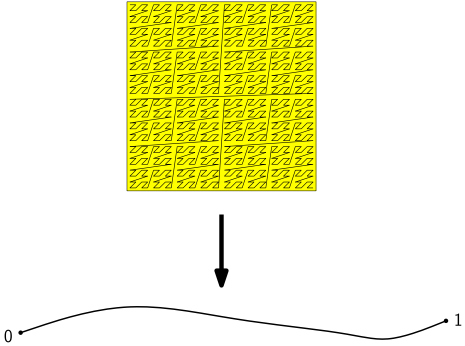
Z-order space-filling curve: visit quadrants recursively in order NW, NE, SW, SE



figures thanks to H. Haverkort

Ingredients: ... and Z-order

Z-order space-filling curve: visit quadrants recursively in order NW, NE, SW, SE

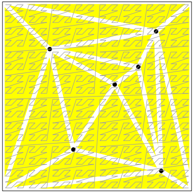


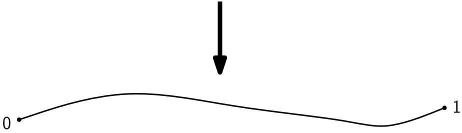
figures thanks to H. Haverkort

Ingredients: quadtrees and Z-order

Quadtree cell ≡ interval on Z-order curve

Quadtree subdivision ≡ subdivision of Z-order curve



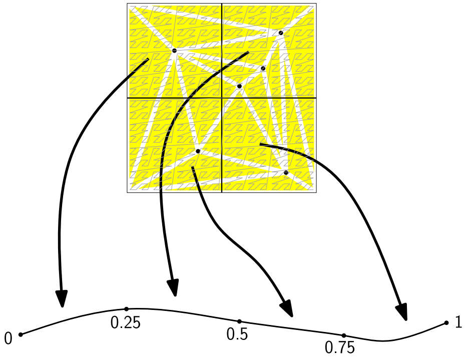


figures thanks to H. Haverkort

Ingredients: quadtrees and Z-order

Quadtree cell ≡ interval on Z-order curve

Quadtree subdivision ≡ subdivision of Z-order curve

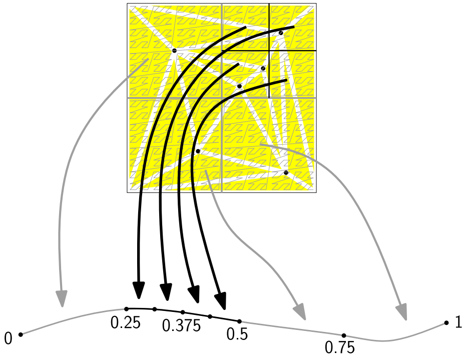


figures thanks to H. Haverkort

Ingredients: quadtrees and Z-order

Quadtree cell ≡ interval on Z-order curve

Quadtree subdivision ≡ subdivision of Z-order curve

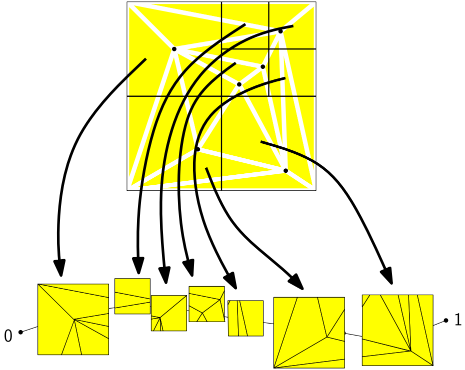


figures thanks to H. Haverkort

Ingredients: quadtrees and Z-order

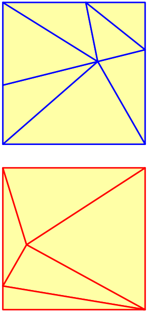
Quadtree cell ≡ interval on Z-order curve

Quadtree subdivision ≡ subdivision of Z-order curve



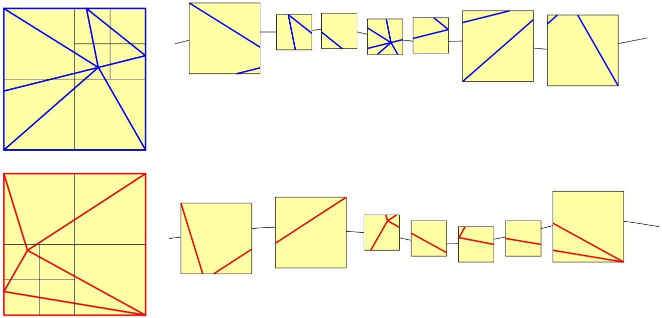
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



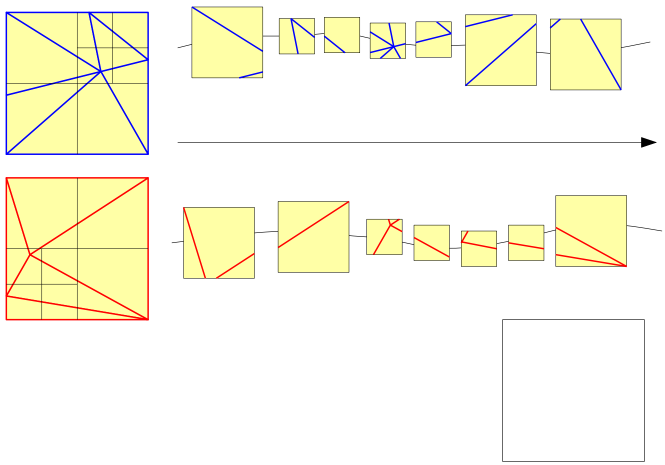
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



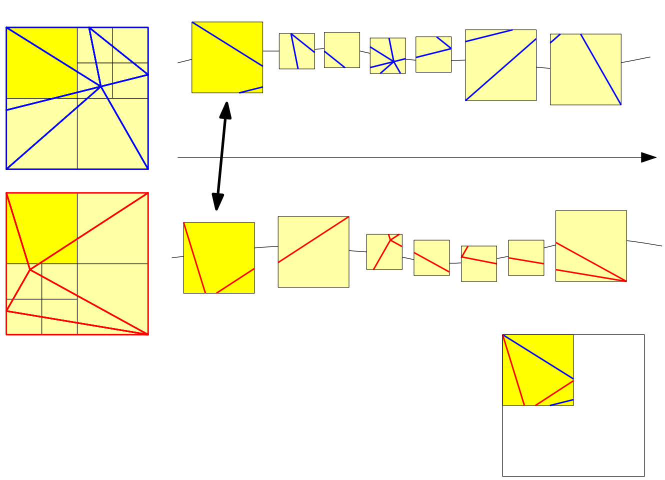
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



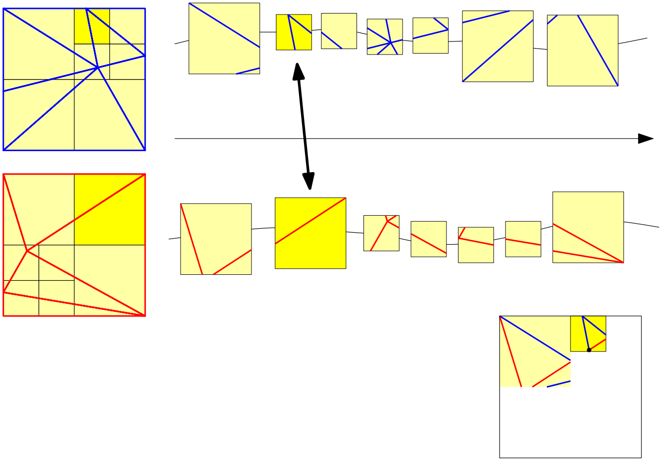
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



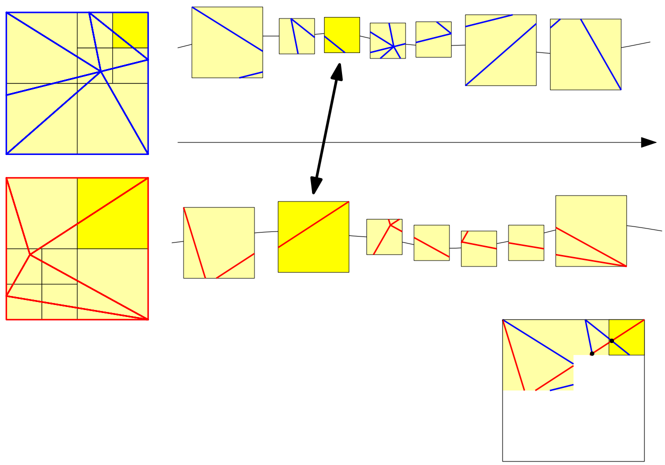
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



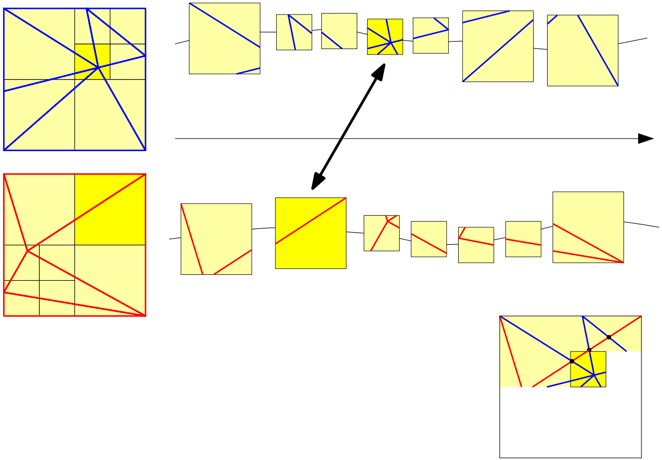
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



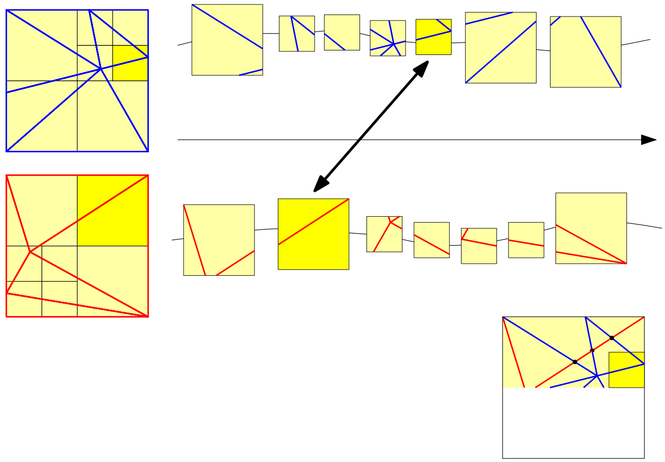
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



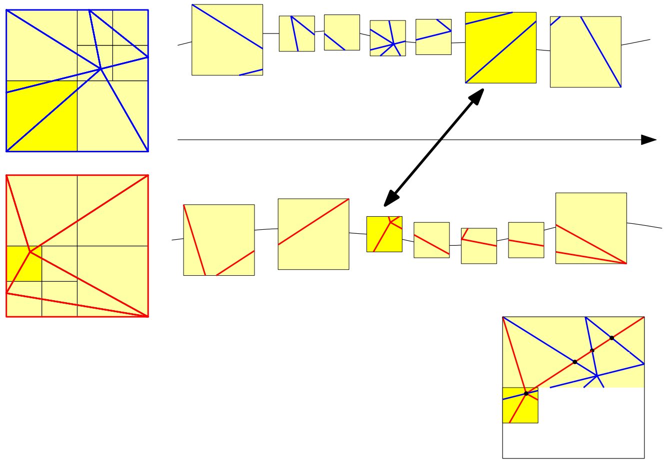
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



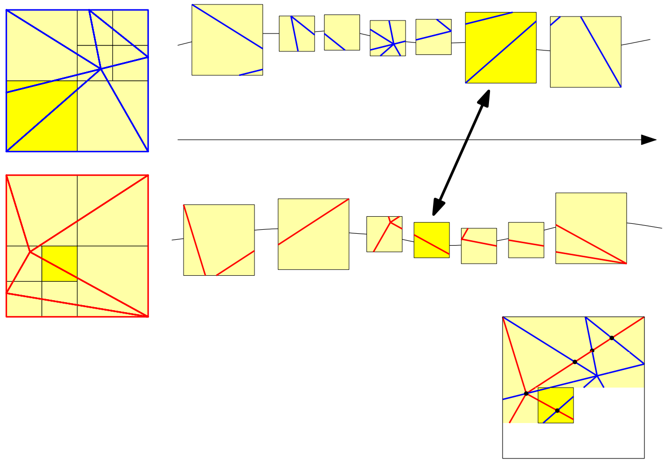
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



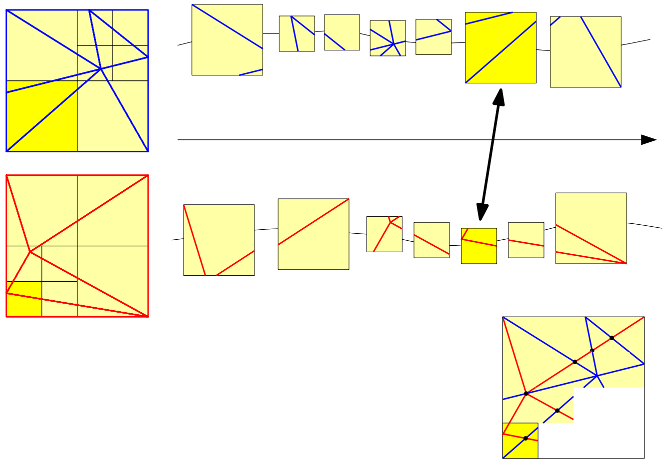
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



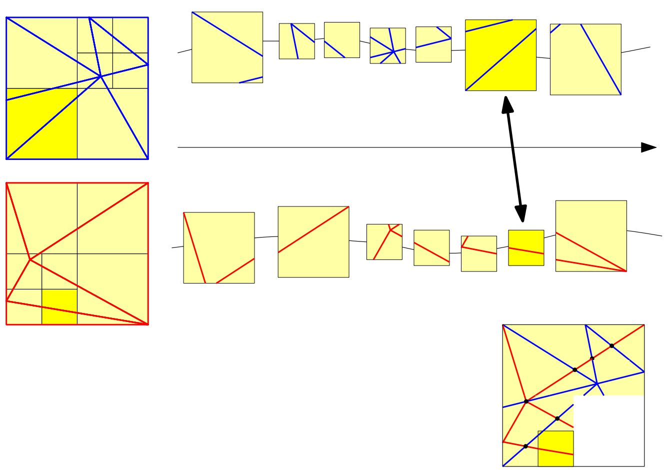
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



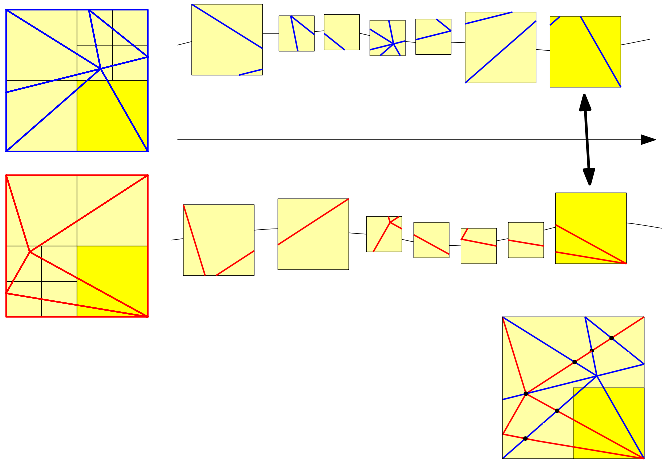
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



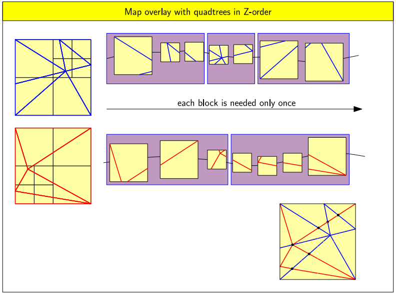
figures thanks to H. Haverkort

Map overlay with quadtrees in Z-order



figures thanks to H. Haverkort

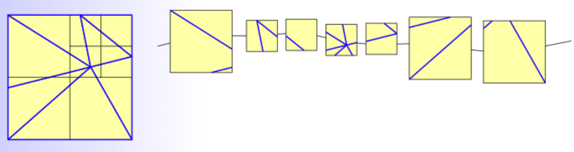
Map overlay with quadtrees



- IO-complexity
  - $\text{scan}(n_1 + n_2 + k)$ IOs
  - assuming a cell fits in memory

figures thanks to H. Haverkort

Building edge quadtrees: Related work



- Build quadtree induced by endpoints and distribute edges
  - $O(n)$ cells, ≤ 1 point per cell, $I = O(n^2)$
- Split a region until it intersects a single edge
  - unbounded size
- Formulate specific stopping criteria
  - PM quadtree (PM1, PM2, PM3)
  - segment quadtree
  - PMR quadtree
  - Samet 85, 86, 87, 89, 92, 97, 99, 02..

Contributions

Given a set of n segments in the plane

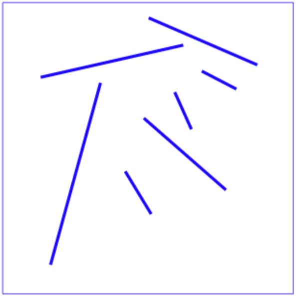
- New algorithm to construct a linear edge-quadtree with $O(n)$ cells in $O(\mathrm{sort}(n+I))$ IOs
  - I is the nb. edge-cell intersections, $I = O(n^2)$
  - same IO bound as [De Berg et al], but much simpler
- k-quadtree
  - $O(k)$ vertices per cell, $O(n/k)$ cells
  - can be constructed in $O(\mathrm{sort}(n+I))$ IOs
- Empirical evaluation
  - triangulated terrains, TIGER data

Outline

Our algorithm

Input: A file with n segments in the plane

Algorithm:

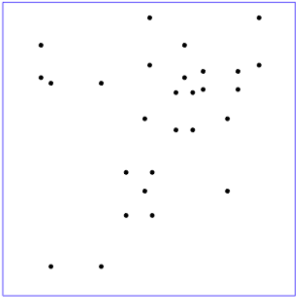


Our algorithm

Input: A file with n segments in the plane

Algorithm:

1. Find the endpoints of the segments.

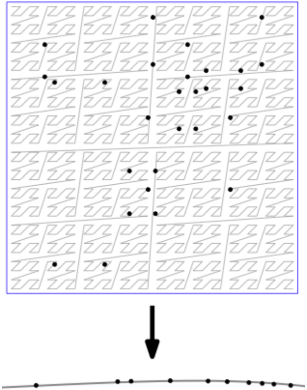


Our algorithm

Input: A file with n segments in the plane

Algorithm:

1. Find the endpoints of the segments.
2. Sort them in Z-order.

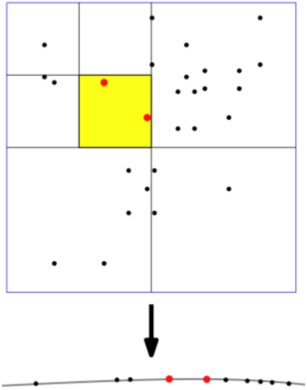


Our algorithm

Input: A file with n segments in the plane

Algorithm:

1. Find the endpoints of the segments.
2. Sort them in Z-order.
3. For every two consecutive points p_i and p_{i+1} in order:
    - find smallest cell Q that contains p_i and p_{i+1}

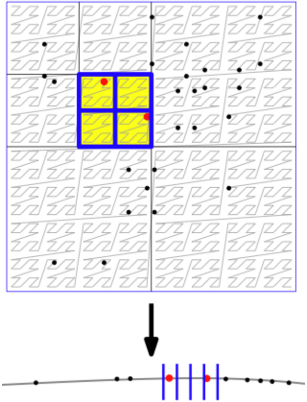


Our algorithm

Input: A file with n segments in the plane

Algorithm:

1. Find the endpoints of the segments.
2. Sort them in Z-order.
3. For every two consecutive points p_i and p_{i+1} in order:
   - find smallest cell Q that contains p_i and p_{i+1}
   - output cell boundaries of Q and its quadrants

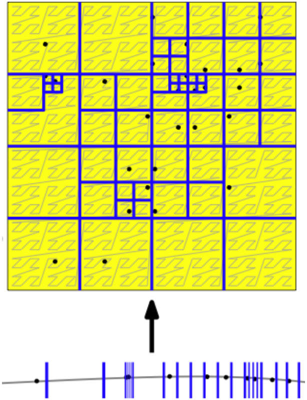


Our algorithm

Input: A file with n segments in the plane

Algorithm:

1. Find the endpoints of the segments.
2. Sort them in Z-order.
3. For every two consecutive points p_i and p_{i+1} in order:
   - find smallest cell Q that contains p_i and p_{i+1}
   - output cell boundaries of Q and its quadrants



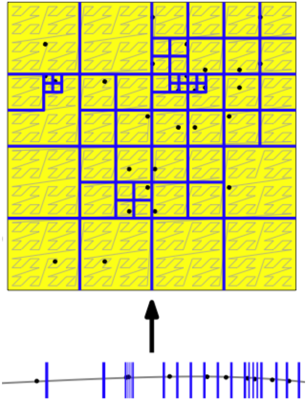
Our algorithm

Input: A file with n segments in the plane

Algorithm:

1. Find the endpoints of the segments.
2. Sort them in Z-order.
3. For every two consecutive points p_i and p_{i+1} in order:
   - find smallest cell Q that contains p_i and p_{i+1}
   - output cell boundaries of Q and its quadrants

$\Longrightarrow$ compressed quadtree subdivision with $O(n)$ cells and ≤ 1 point per cell.

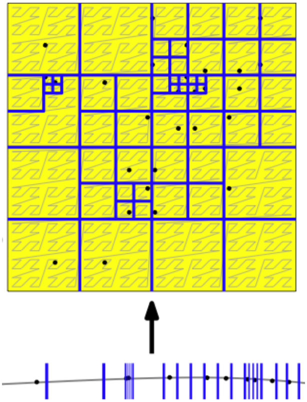


Our algorithm

Input: A file with n segments in the plane

Algorithm:

1. Find the endpoints of the segments.
2. Sort them in Z-order.
3. For every two consecutive points p_i and p_{i+1} in order:
   - find smallest cell Q that contains p_i and p_{i+1}
   - output cell boundaries of Q and its quadrants
4. Distribute edges to cells.



Edge distribution

Input

- E: a set of edges in the plane.
- $Q = \{z_0, z_1, z_2, ...\}$: a quadtree subdivision of $[0, 1]$

Output

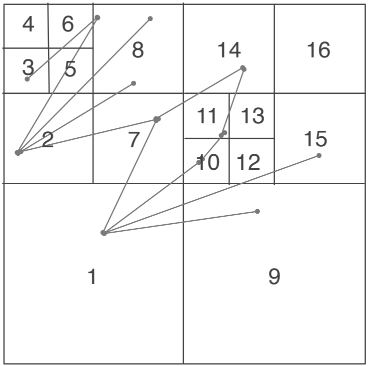
- For each interval $I_k = [z_k, z_{k+1}]$, the set of edges that intersect σ_k

- Let E^+ be the edges of positive slope
- Let E^- be the edges of negative slope

We'll process E^+ and E^- separately.

Distributing E^+

Input: $Q = \{z_0, z_1, ...\}, E^+$

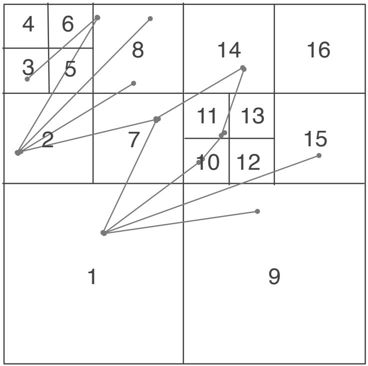


Distributing E^+

Input: $Q = \{z_0, z_1, ...\}, E^+$

Algorithm:

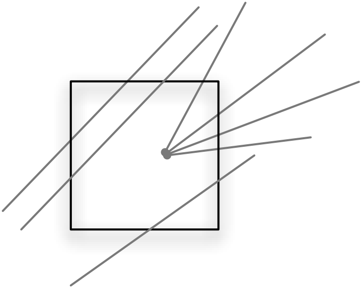
- Sort E^+ by the z-index of the first endpoint.
- For each interval $I_k = [z_{k-1}, z_k]$ in Q:
    - Find all edges in E^+ that intersect I_k



Distributing E^+

Algorithm:

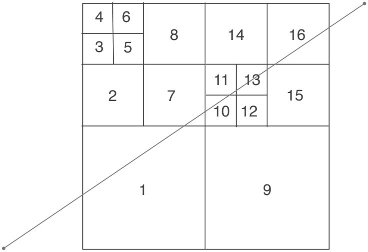
- Sort E^+ by the z-index of the first endpoint.
- For each interval $I_k = [z_{k-1}, z_k]$ in Q:
  - Find all edges in E^+ that intersect I_k



Distributing E^+

Lemma

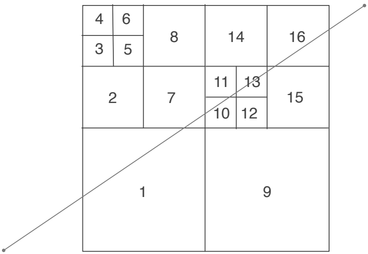
An edge of positive slope intersects the cells in Q in z-order.



Distributing E^+

Lemma

An edge of positive slope intersects the cells in Q in z-order.



The edges that intersect I_k either:

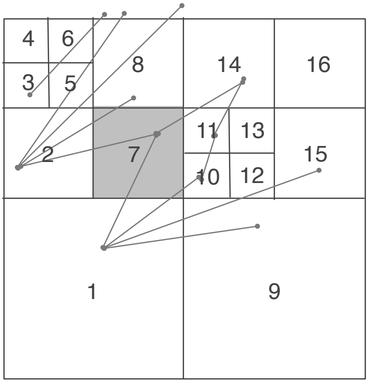
- start in I_k, or,
- start in an interval before I_k

Distributing E^+

Input: $Q = \{z_0, z_1, ...\}, E^+$

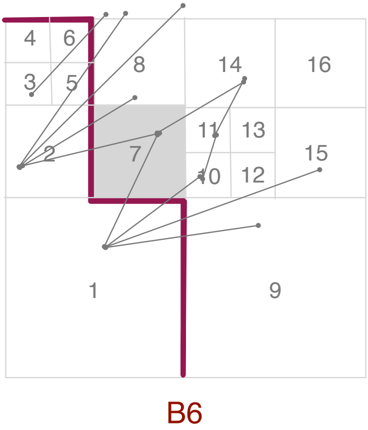
Algorithm:

- Sort E^+ by the z-index of the first endpoint.
- For each interval $I_k = [z_{k-1}, z_k]$ in Q:
  - Find all edges in E^+ that start in I_k



Distributing E^+

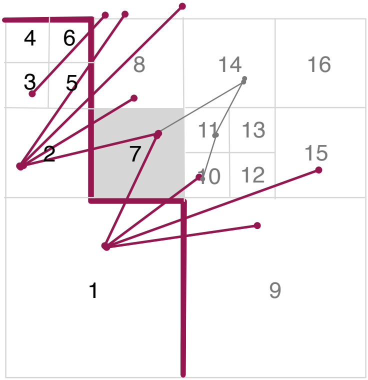
B_k: the boundary between $\sigma_1 \cup \sigma_2 \cup \ldots \cup \sigma_k$ and $\sigma_{k+1} \cup \sigma_{k+2} \ldots$



Distributing E^+

B_k: the boundary between $\sigma_1 \cup \sigma_2 \ldots \cup \sigma_k$ and $\sigma_{k+1} \cup \sigma_{k+2} \ldots$

BL_k: the edges that intersect B_k, in order.



BL6

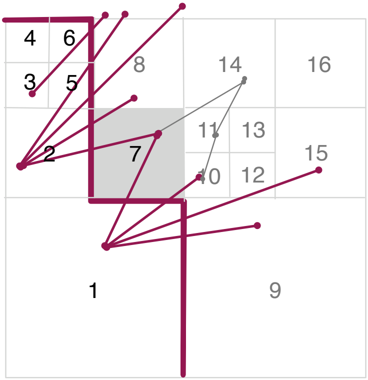
Distributing E^+

B_k: the boundary between $\sigma_1 \cup \sigma_2 \ldots \cup \sigma_k$ and $\sigma_{k+1} \cup \sigma_{k+2} \ldots$

BL_k: the edges that intersect B_k, in order.

Lemma

B_k is a monotone staircase and the intersection of σ_k and B_{k-1} covers a connected part of B_{k-1}.

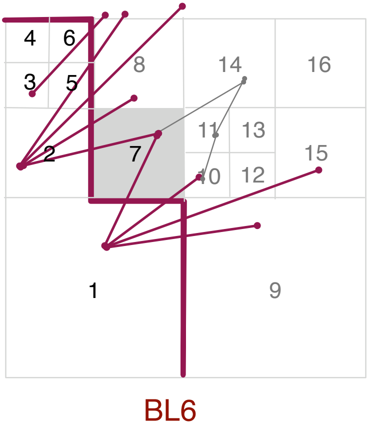


BL6

Distributing E^+

Algorithm:

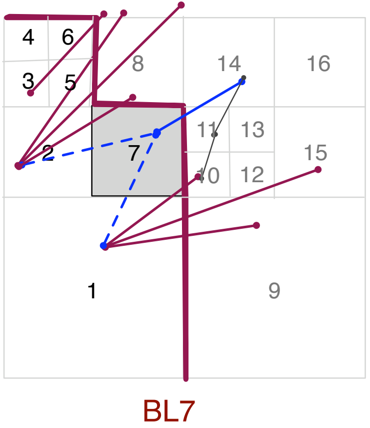
- Sort E^+ by the z-index of the first endpoint.
- For each interval $I_k = [z_{k-1}, z_k]$ in Q:
  - Find all edges in E^+ that start in I_k
  - Use BL_{k-1} to find the edges that start before σ_k and intersect σ_k



Distributing E^+

Algorithm:

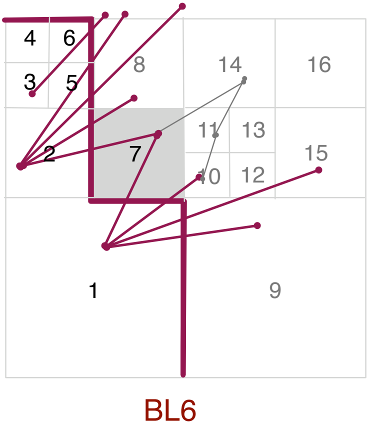
- Sort E^+ by the z-index of the first endpoint.
- For each interval $I_k = [z_{k-1}, z_k]$ in Q:
  - Find all edges in E^+ that start in I_k
  - Use BL_{k-1} to find the edges that start before σ_k and intersect σ_k
  - Update BL_{k-1} to BL_k



Distributing E^+

How to find an edge in BL_{k-1} that intersects σ_k?

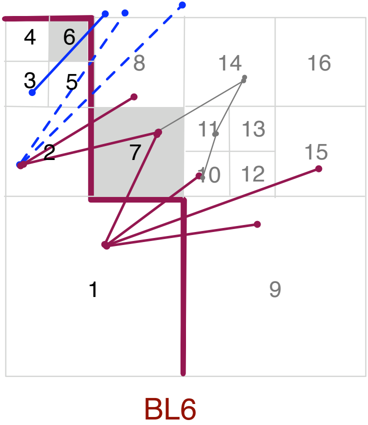
- avoid searching



Distributing E^+

How to find an edge in BL_{k-1} that intersects σ_k?

Start from the first edge in σ_{k-1} that intersects BL_{k-1}



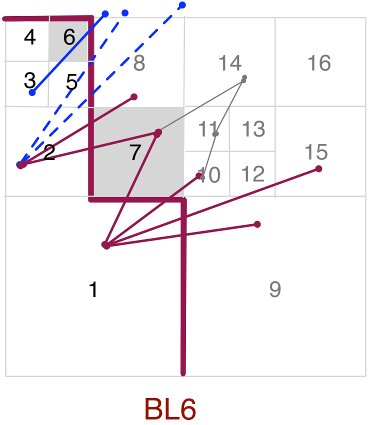
Distributing E^+

How to find an edge in BL_{k-1} that intersects σ_k?

Start from the first edge in σ_{k-1} that intersects BL_{k-1}

Lemma

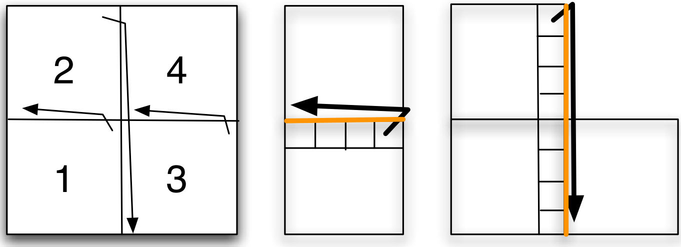
The number of edges traversed and skipped is $O(l)$.



Distributing E^+

Lemma

The number of edges traversed and skipped is $O(I)$, where I is the number of edge-cell intersections.



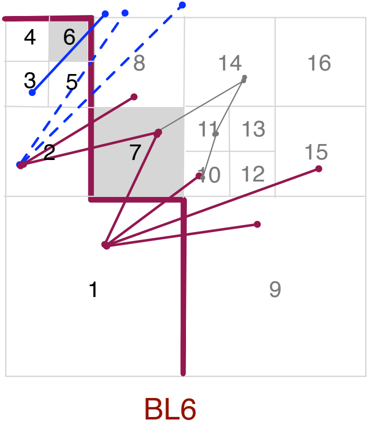
Distributing E^+

How to find an edge in BL_{k-1} that intersects σ_k?

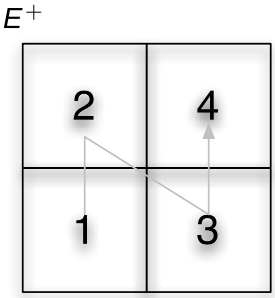
Start from the first edge in σ_{k-1} that intersects BL_{k-1}

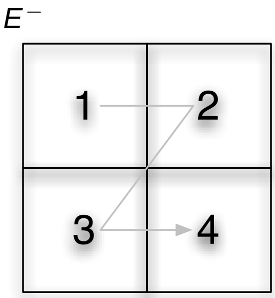
Lemma

The intersections of E^+ and Q can be found in $O(scan(n+l))$ IOs, once Q and E^+ are sorted.



Distributing E^-





Our algorithm

Theorem

Given a set of n edges in the plane, a compressed quadtree subdivision with $O(n)$ cells and $O(1)$ points per cell can be computed in $O(\mathrm{sort}(n + I))$ IOs, where I is the number of edge-cell intersections.

The k-quadtree

The algorithm can be extended to get a quadtree with $O(k)$ vertices per cell.

1. Find the endpoints of the segments.
2. Sort them in Z-order.
3. For every two consecutive points in $p_0, p_k, p_{2k}, \ldots$:
   - find smallest cell Q that contains them
   - output cell boundaries of Q and its quadrants
4. Distribute edges to cells.
   - interleave the edges in σ_k with the edges in BL_{k-1}, etc

Theorem

A quadtree subdivision with $O(n/k)$ cells, each cell with $O(k)$ vertices can be computed in $O(\mathrm{sort}(n+I))$ IOs, where $I = O(n^2/k)$ is the number of edge-cell intersections.

The k-quadtree

The algorithm can be extended to get a quadtree with $O(k)$ vertices per cell.

1. Find the endpoints of the segments.
2. Sort them in Z-order.
3. For every two consecutive points in $p_0, p_k, p_{2k}, \ldots$:
   - find smallest cell Q that contains them
   - output cell boundaries of Q and its quadrants
4. Distribute edges to cells.
   - interleave the edges in σ_k with the edges in BL_{k-1}, etc

Theorem

A quadtree subdivision with $O(n/k)$ cells, each cell with $O(k)$ vertices can be computed in $O(\mathrm{sort}(n+I))$ IOs, where $I = O(n^2/k)$ is the number of edge-cell intersections.

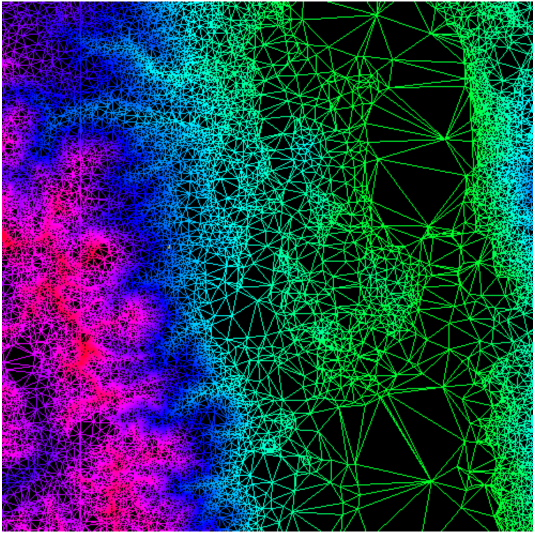
Outline

Datasets: Triangulated terrains

- We ignored the elevation.
- Delaunay triangulation
- Lots of small angles on the boundary

Dataset	e	Max inc.	Min $\angle$
Kaweah	$1.2 \cdot 10^6$	31	.0704
Puerto Rico	$4.1 \cdot 10^6$	291	.0010
Cumberlands	$5.1 \cdot 10^6$	44	.0016
Sierra	$7.9 \cdot 10^6$	75	.0137
Central App.	$10.1 \cdot 10^6$	62	.0013
Hawaii	$19.7 \cdot 10^6$	356	.0007
Haldem	$37.1 \cdot 10^6$	78	.0097
Lower NE	$53.9 \cdot 10^6$	168	.0021

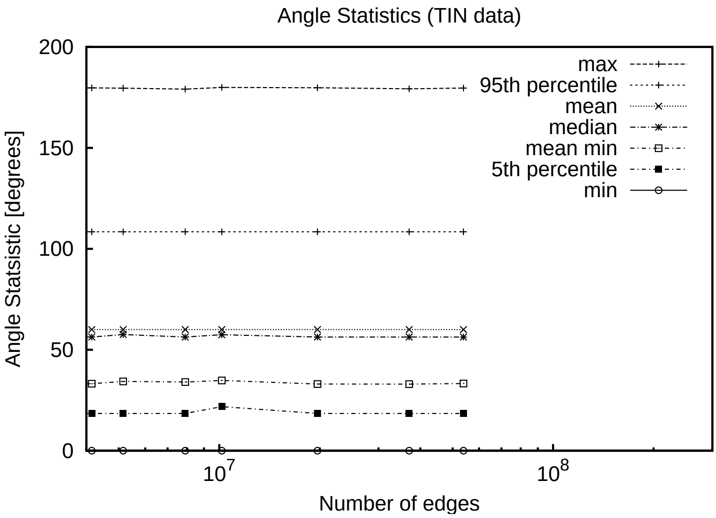
Datasets: Triangulated terrains



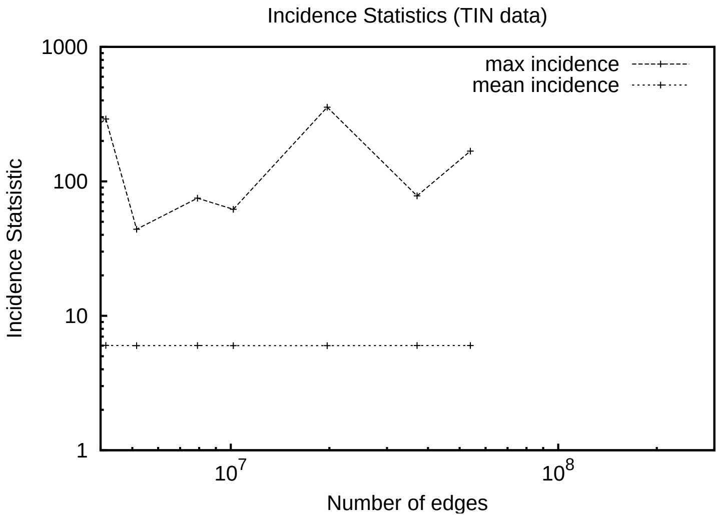
Datasets: Triangulated terrains

- min angle around .001°
- max angle close to 180°
- 5% below 18°
- 5% above 108°
- median angle 57°
- max degree varies widely across all datasets, ranging between 31 and 356
- average degree across all datasets is approx. 6.

Datasets: Triangulated terrains



Datasets: Triangulated terrains



Datasets: TIGER data

- Available at `http://www.census.gov/geo/www/tiger/`
- 50 sets, one set for each state, containing roads, hydrography, railways and boundaries
- Largest set: TX ($e = 40.4 \cdot 10^6$)
- We assembled larger bundles.

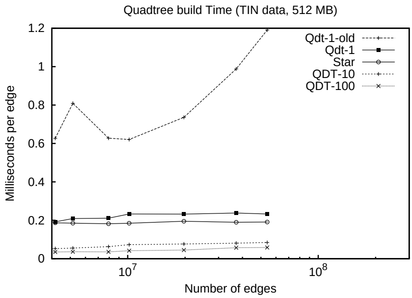
Dataset	e
`New England`	$25.8 \cdot 10^6$
`East Coast`	$113.0 \cdot 10^6$
`Eastern Half`	$208.3 \cdot 10^6$
`All USA`	$427.7 \cdot 10^6$

Platform

- C
- `g++ 4.1.2 -O3`
- HP 220 blade servers
- Intel 2.83 GHz
- 5400 rpm SATA drive
- 512 MB RAM

Results: TIN data

Quadtree build time

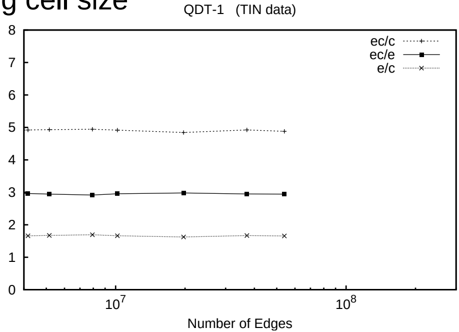


- Our algorithms: QDT-k (k=1, 10, 100, ..)
- Previous work [De Berg et al]: Qdt-1-old, Star
- QDT-k gets faster up to $k = 100$ and then levels

Results: TIN data

QDT-1 size

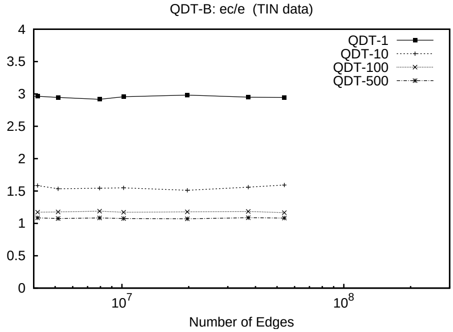
- ec: nb edge-cell intersections
- c: nb cells
- ec/c: avg cell size



- Across all datasets: $c \approx .6e$, $ec \approx 3e$, $ec \approx 5c$

Results: TIN data

QDT-k total size



- As k increases
  - c decreases, ec/c increases, ec/e decreases
  - fewer cells $\longrightarrow$ fewer edge-cell intersections

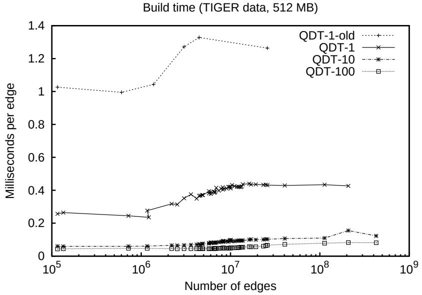
Results: TIN data

Sizes and build time on LowerNE ($e = 53.9 \cdot 10^6$)

	c	ec	ec/c	build (min)
QDT-1	$32.5 \cdot 10^6$	$158.8 \cdot 10^6$	4.8	210
QDT-100	$.24 \cdot 10^6$	$62.8 \cdot 10^6$	257.4	57
QDT-500	$.06 \cdot 10^6$	$58.4 \cdot 10^6$	957.4	53

Results: TIGER data

Quadtree build time

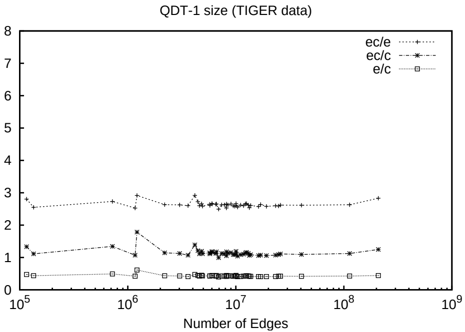


- Our algorithms: Qdt-K (k=1, 10, 100, ..)
- Previous work [De Berg et al]: Qdt-1-old
- QDT-k gets faster up to $k = 100$ and then levels
- QDT-100 on AllUSA in 9.7 hours

Results: TIGER data

QDT-1 size

- ec: nb edge-cell intersections
- c: nb cells
- ec/c: avg cell size

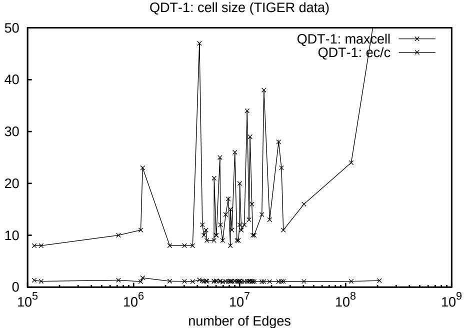


Sizes relatively consistent across all data sets (!).

$c \approx 2.5e$, $ec \approx 3e$, $ec \approx c$

Results: TIGER data

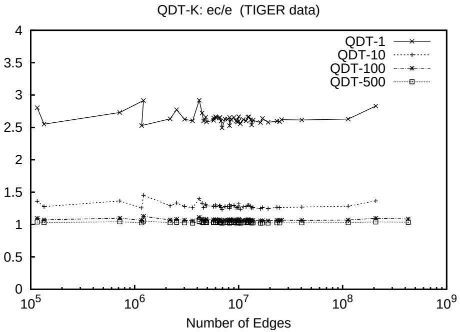
QDT-1: maximum cell size



- Varies widely from state to state
  - E.g.: Easthalf: max cell intersects 58 edges
  - ME: max cell intersects 8 edges

Results: TIGER data

QDT-k total size: ec/e



- As k increases
  - c decreases, ec/c increases, ec/e decreases

Results: TIGER data

Sizes and build time on EastHalf ($e = 208 \cdot 10^6$)

	c	ec	ec/c	build (min)
QDT-1	$472.5 \cdot 10^6$	$589.7 \cdot 10^6$	1.3	1,482
QDT-10	$36.8 \cdot 10^6$	$284.4 \cdot 10^6$	7.7	539
QDT-100	$3.2 \cdot 10^6$	$228.4 \cdot 10^6$	71.4	287

Results: Map overlay

- We overlayed a TIN stored as QDT-1 with TIGER data stored as QDT-k
  - All data scaled to unit square
- Fast and scalable
- Optimal k: $k \in [100, 500]$
  - cell-cell intersection time increases with k
  - nb. of cells decreases with k

Summary

- A simple and IO-efficient algorithm to build k-quadtrees
- Fast and scalable in practice
  - tested up to $e = 427 \cdot 10^6$ with 512MB RAM
- k-quadtrees are a viable solution for two classes of data widely used in practice, TIN and TIGER

- Outlook
  - Comparison with PMR quadtree
  - Other applications?

Thank you!