Nice Triangulations

Given a point set S in the plane produce a *nice triangulation* $\mathcal{T}$ of S, i.e. each triangle Δ of $\mathcal{T}$ is "nicely shaped."

Nice Triangulations

Given a point set S in the plane produce a *nice triangulation* $\mathcal{T}$ of S, i.e. each triangle Δ of $\mathcal{T}$ is "nicely shaped."

Different definitions of "nicely shaped" triangle Δ:

- smallest angle is large
- the largest angle is small
- all angles are acute
- ratio of radius of circumcirle and radius of incircle is small
- ratio of longest edge and shortest edge is small
- ratio of longest edge and corresponding altitude is small
- ...

Nice Triangulations

Given a point set S in the plane produce a *nice triangulation* $\mathcal{T}$ of S, i.e. each triangle Δ of $\mathcal{T}$ is "nicely shaped."

Different definitions of "nicely shaped" triangle Δ:

- smallest angle is large
- the largest angle is small
- all angles are acute
- ratio of radius of circumcirle and radius of incircle is small
- ratio of longest edge and shortest edge is small
- ratio of longest edge and corresponding altitude is small
- ...

All this notions are closely related and the respective "small" and "large" can be appropritely parameterized and related to each other.

Nice Triangulations

- $\theta(\Delta)$: smallest angle of Δ
- $R(\Delta)$: ratio of longest edge and shortest edge
- $A(\Delta)$: ratio of longest edge and corresponding altitude ("aspect ratio")

Relations:

$$\frac{1}{\sin\theta(\Delta)} \leq A(\Delta) \leq \frac{2}{\sin\theta(\Delta)} \quad \text{and} \quad R(\Delta) < A(\Delta)$$

Nice Triangulations

- $\theta(\Delta)$: smallest angle of Δ
- $R(\Delta)$: ratio of longest edge and shortest edge
- $A(\Delta)$: ratio of longest edge and corresponding altitude ("aspect ratio")

Relations:

$$\frac{1}{\sin\theta(\Delta)} \leq A(\Delta) \leq \frac{2}{\sin\theta(\Delta)} \quad \text{and} \quad R(\Delta) < A(\Delta)$$

$\mathcal{T}$ triangulation

- $\theta(\mathcal{T}) = \min_{\Delta\in\mathcal{T}} \theta(\Delta)$
- $R(\mathcal{T}) = \max_{\Delta\in\mathcal{T}} R(\Delta)$
- $A(\mathcal{T}) = \max_{\Delta\in\mathcal{T}} A(\Delta)$

Looking for triangulations $\mathcal{T}$ of S so that min-angle $\theta(\mathcal{T})$ is large, of $R(\mathcal{T})$ is small, of $A(\mathcal{T})$ is small.

Nice Triangulations

Theorem:
Among all triangulations $\mathcal{T}$ of S the Delaunay triangulation maximizes the minimum angle, i.e. $\theta(\mathcal{DT}(S)) = \max\{\theta(\mathcal{T}) | \mathcal{T} \text{ triangulation of } S\}$.

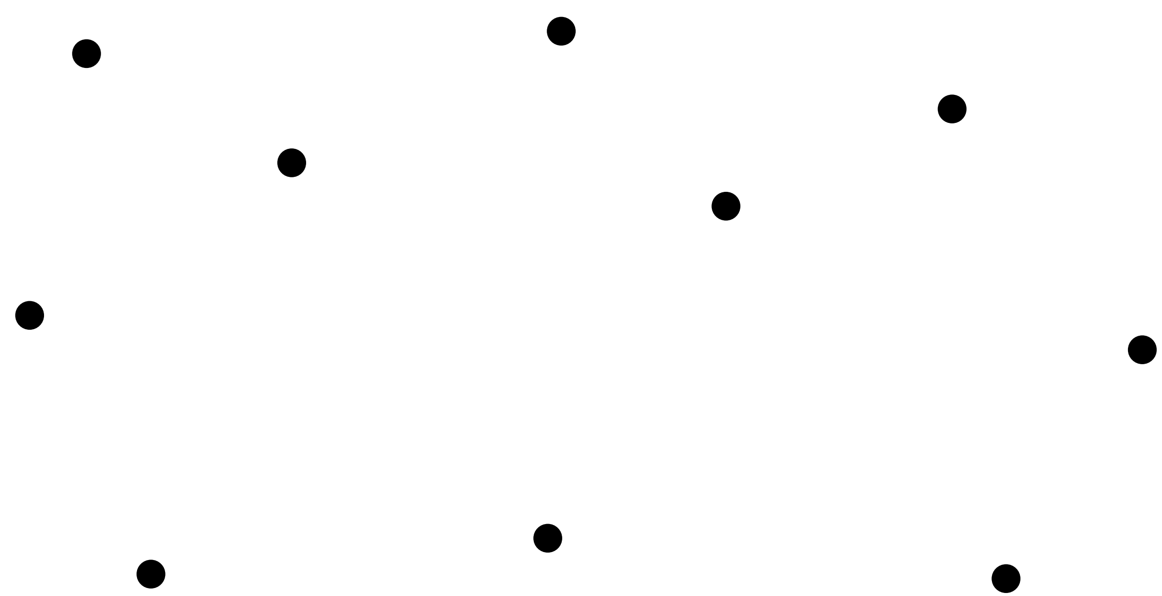
Nice Triangulations

Theorem:
Among all triangulations $\mathcal{T}$ of S the Delaunay triangulation maximizes the minimum angle, i.e. $\theta(\mathcal{DT}(S)) = \max\{\theta(\mathcal{T}) | \mathcal{T} \text{ triangulation of } S\}$.

Proof idea:
Flip algorithm makes sorted vector of all triangle angles in the triangulation lexicographically increase.

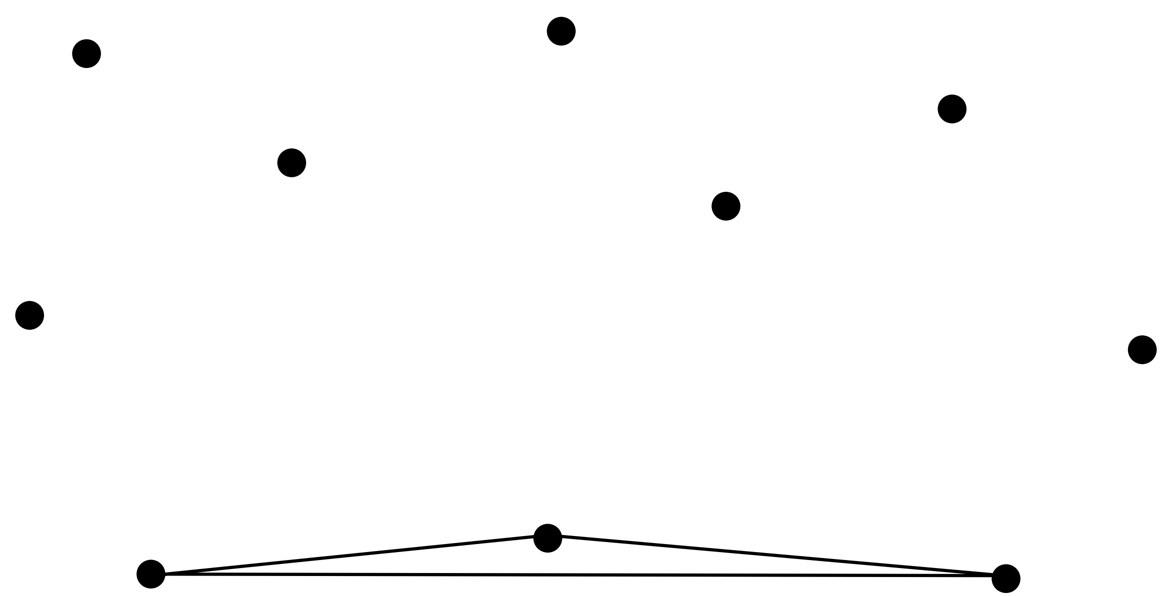
Nice Triangulations

Problem:
For some point sets S, there are no nice triangulations, i.e. $\theta(\mathcal{T})$ is small for all triangulations $\mathcal{T}$ of S (and $A(\mathcal{T})$ and $R(\mathcal{T})$ are large).



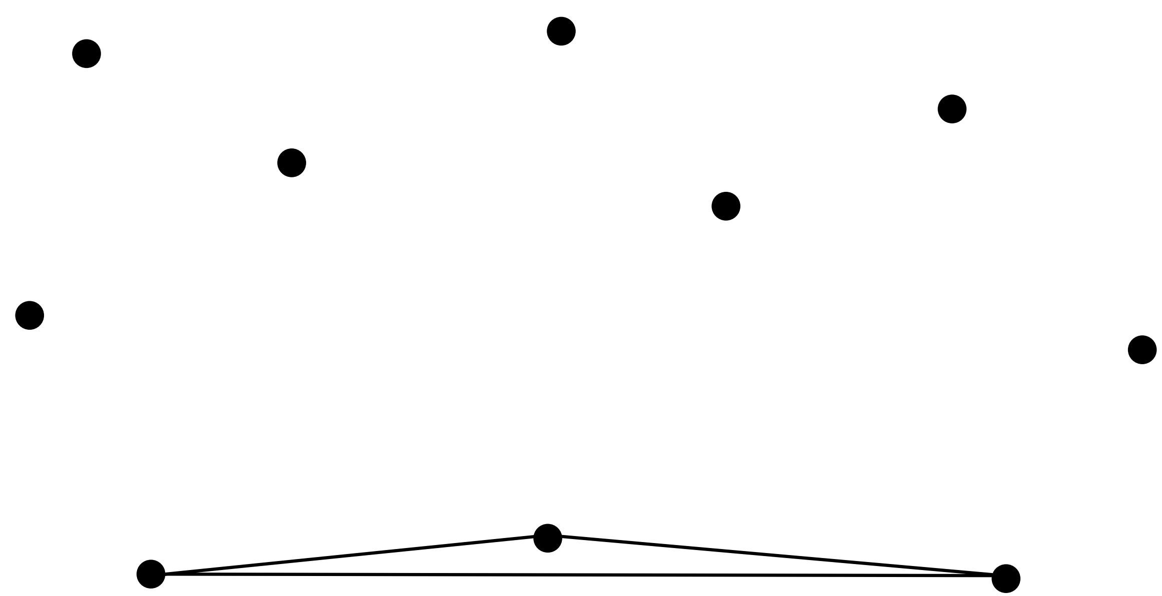
Nice Triangulations

Problem:
For some point sets S, there are no nice triangulations, i.e. $\theta(\mathcal{T})$ is small for all triangulations $\mathcal{T}$ of S (and $A(\mathcal{T})$ and $R(\mathcal{T})$ are large).



Nice Triangulations

Problem:
For some point sets S, there are no nice triangulations, i.e. $\theta(\mathcal{T})$ is small for all triangulations $\mathcal{T}$ of S (and $A(\mathcal{T})$ and $R(\mathcal{T})$ are large).

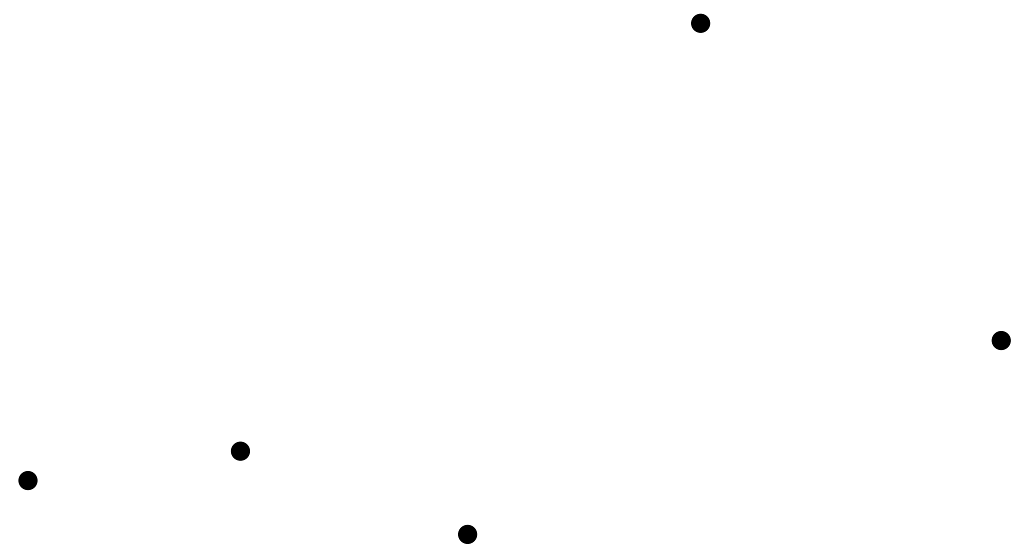


Idea: Add points to S (so-called "Steiner points") so that a nice triangulation on the larger set is possible

Nice Triangulations

Idea: Add points to S (so-called “Steiner points”) so that a nice triangulation on the larger set is possible.

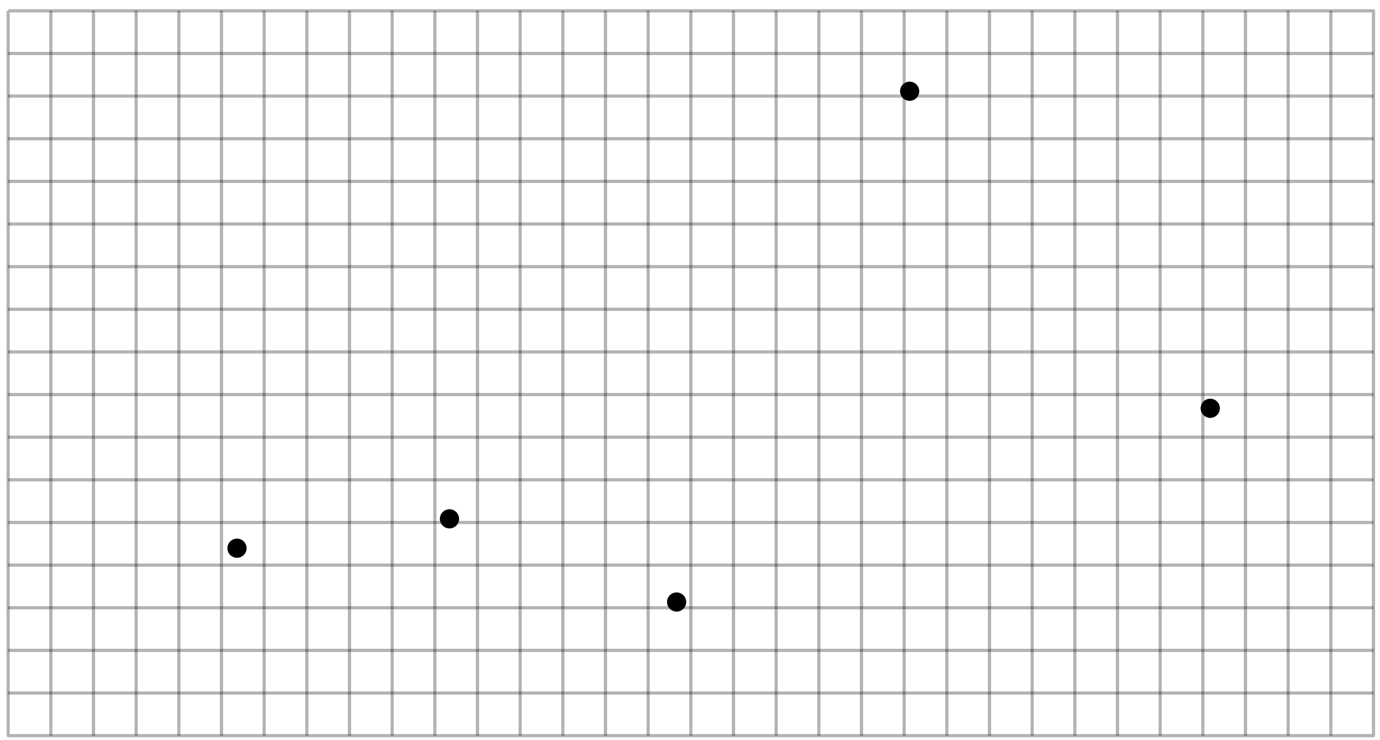
This is always possible!!!



Nice Triangulations

Idea: Add points to S (so-called "Steiner points") so that a nice triangulation on the larger set is possible.

This is always possible!!!

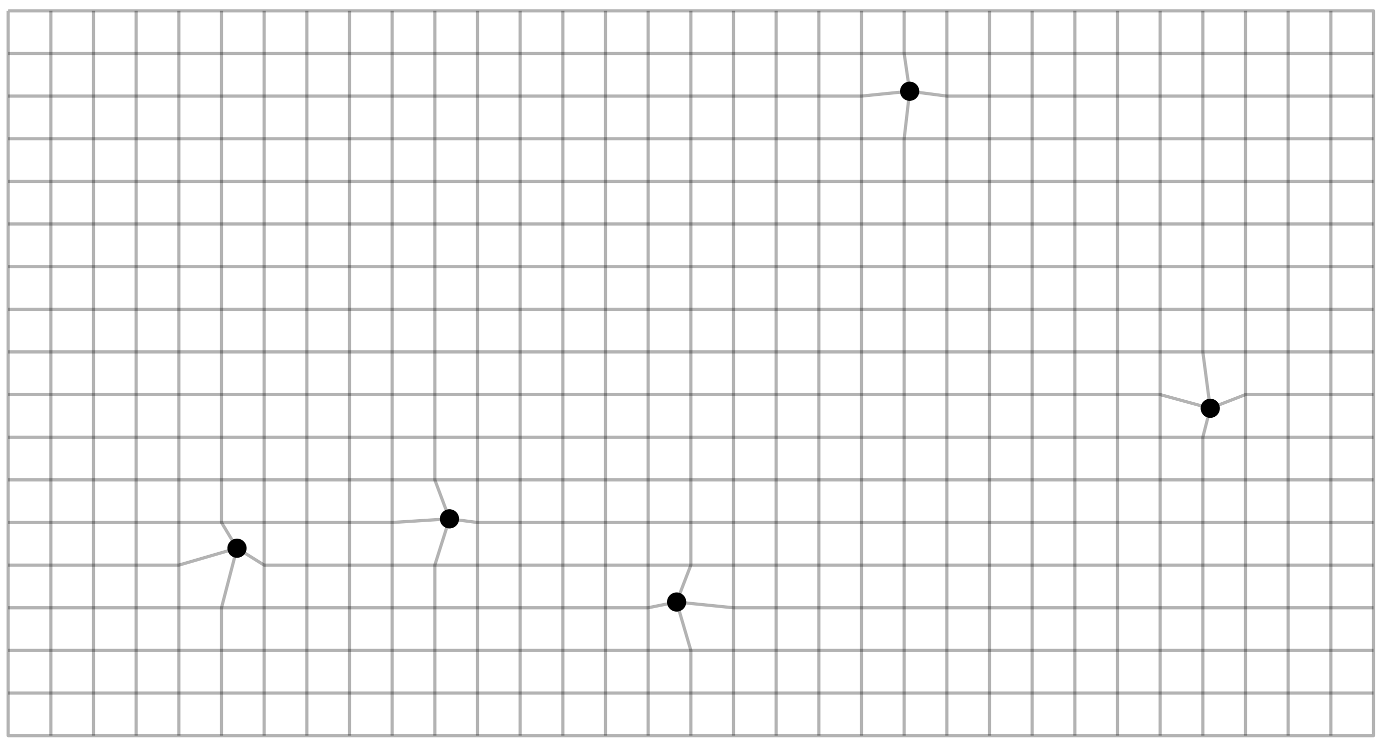


1.Draw sufficiently fine grid, so that points in S are separated by two layers of boxes.

Nice Triangulations

Idea: Add points to S (so-called "Steiner points") so that a nice triangulation on the larger set is possible.

This is always possible!!!

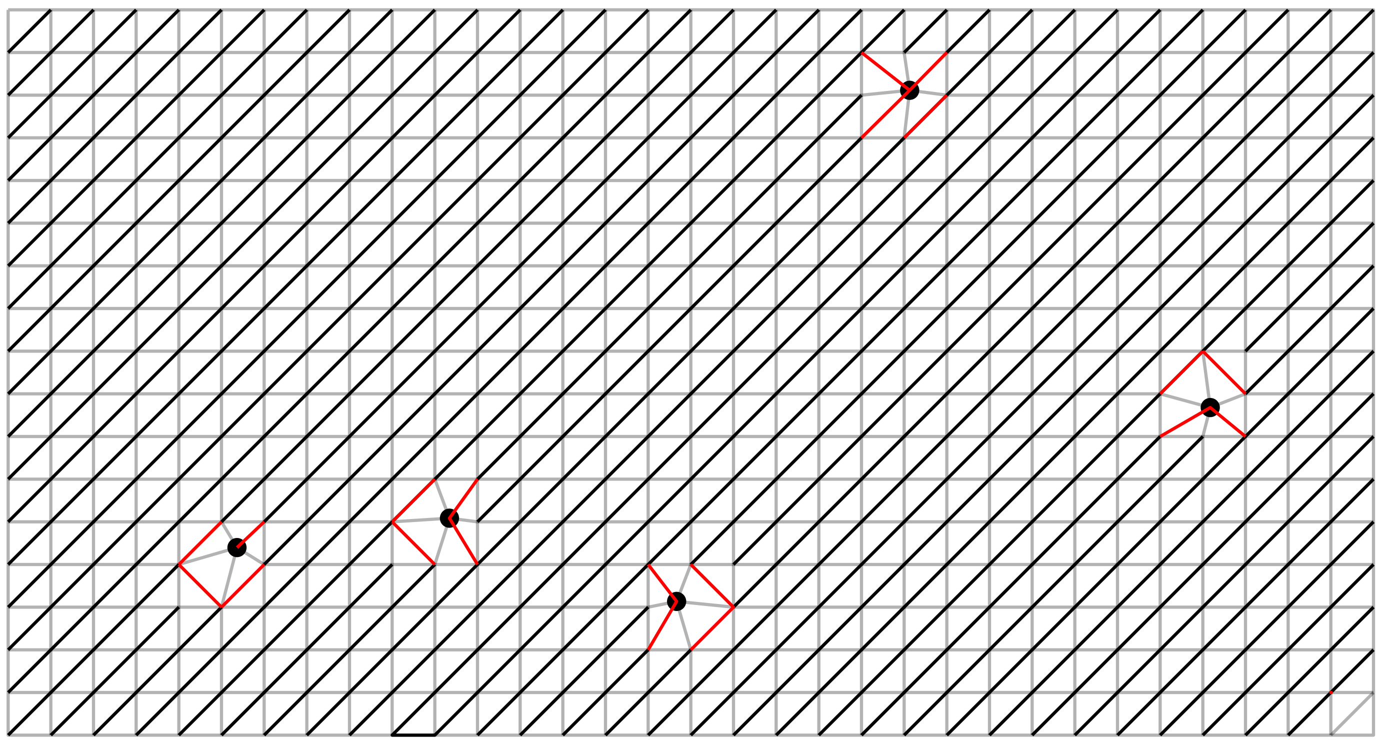


1.Draw sufficiently fine grid, so that points in S are separated by two layers of boxes.
2. For each point in S warp the closest grid point to its position.

Nice Triangulations

Idea: Add points to S (so-called "Steiner points") so that a nice triangulation on the larger set is possible.

This is always possible!!!



1.Draw sufficiently fine grid, so that points in S are separated by two layers of boxes.

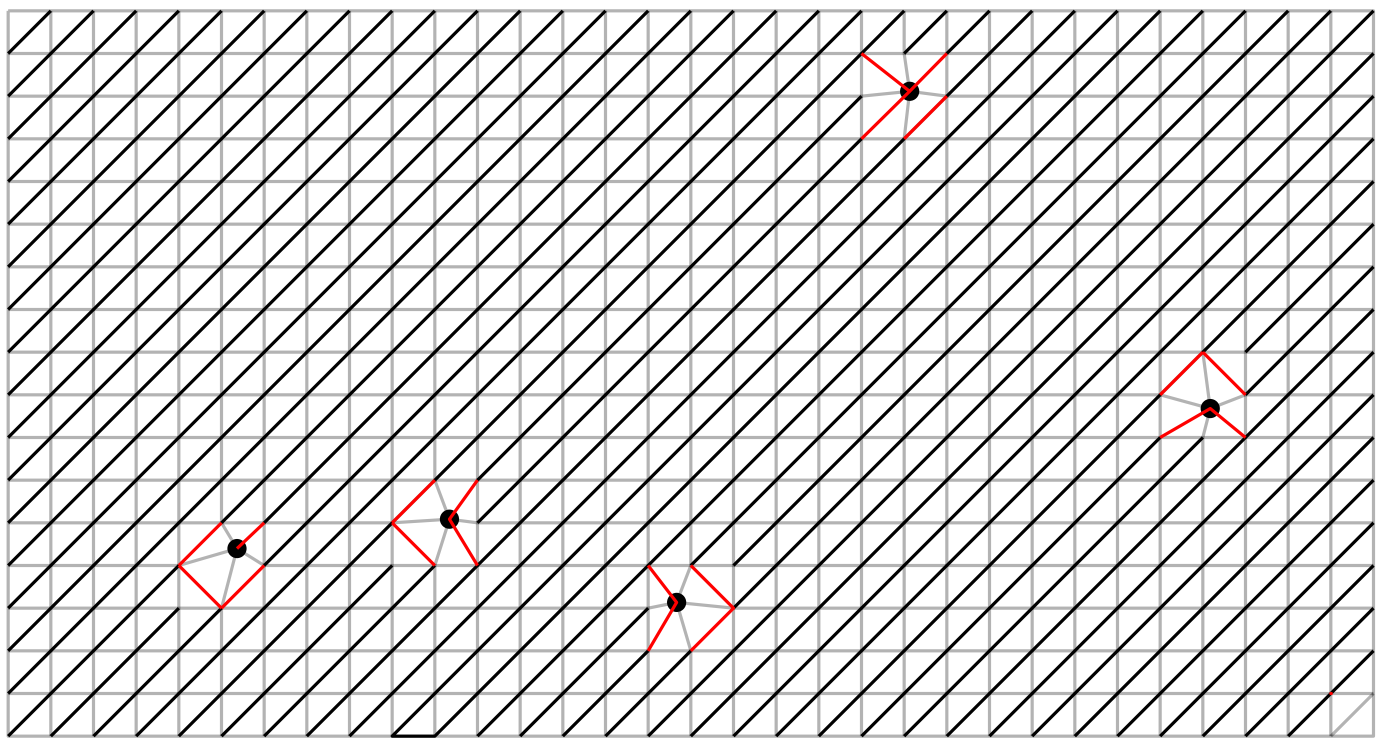
2. For each point in S warp the closest grid point to its position.

3. Triangulate each quadrilateral.

Nice Triangulations

Idea: Add points to S (so-called "Steiner points") so that a nice triangulation on the larger set is possible.

This is always possible!!!

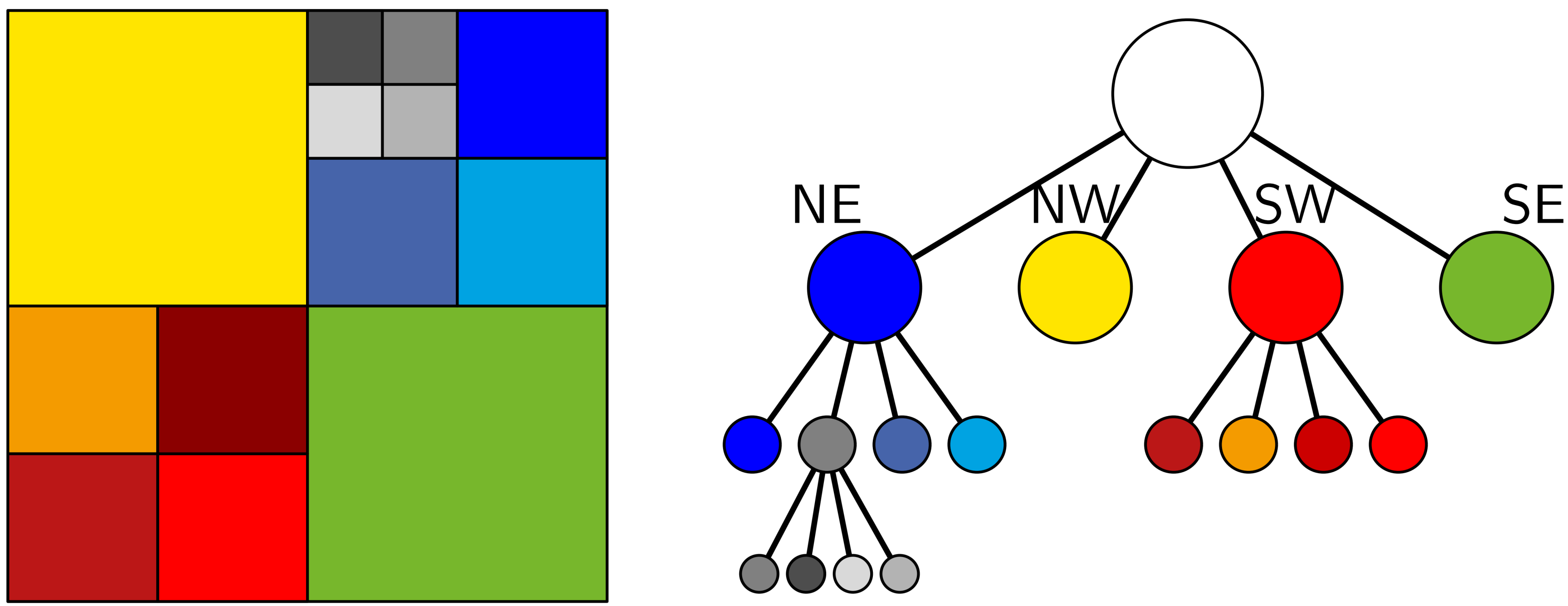


1.Draw sufficiently fine grid, so that points in S are separated by two layers of boxes.
2. For each point in S warp to closest grid point to its position.
3. Triangulate each quadrilateral.

Way too many new vertices!!!

Quadtrees

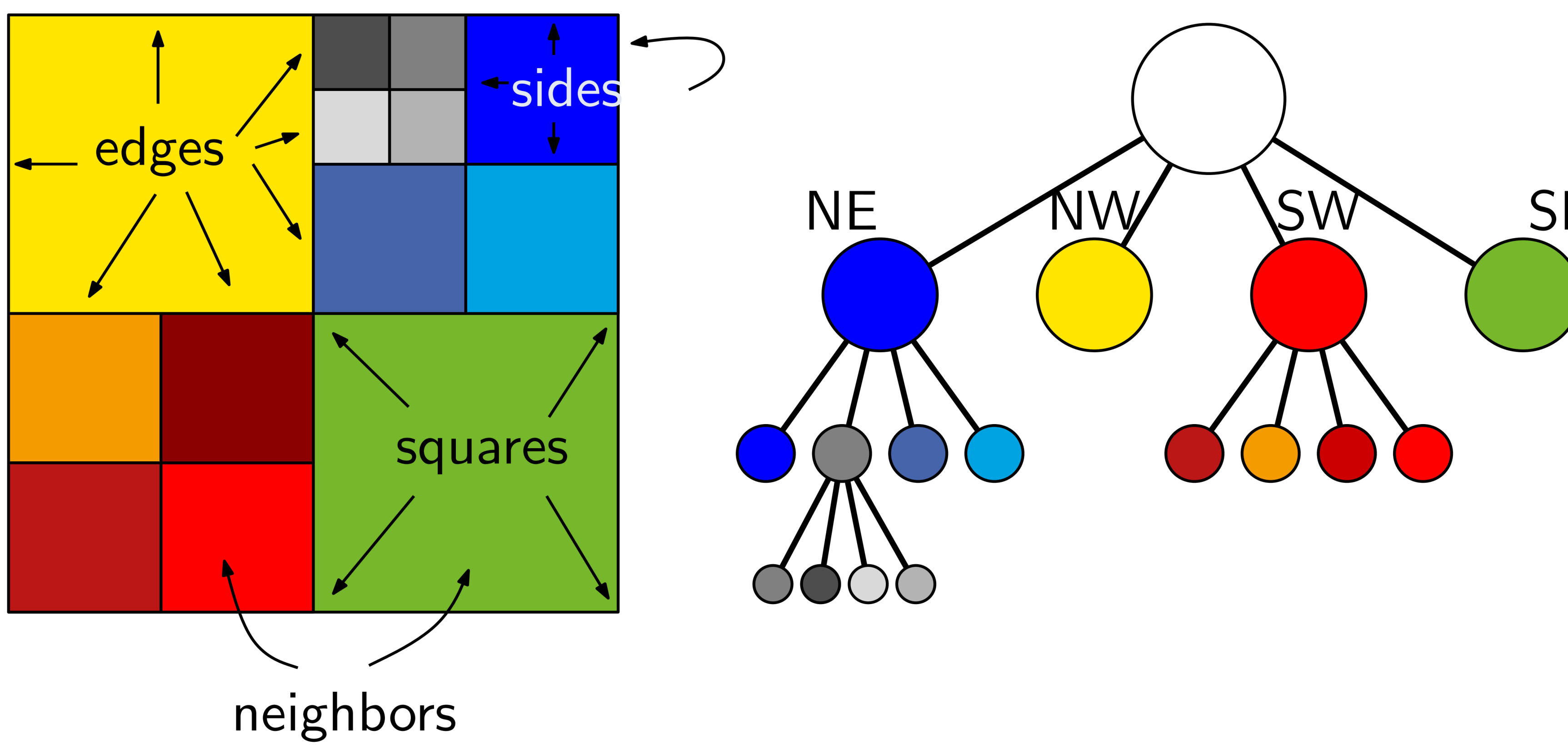
Def.: A **quadtree** is a rooted tree, where each internal node has 4 children. Each node corresponds to a square, and the squares of the leaves form a partition of the root square.



From T. Mchedlidze, KIT

Quadtrees

Def.: A **quadtree** is a rooted tree, where each internal node has 4 children. Each node corresponds to a square, and the squares of the leaves form a partition of the root square.



From T. Mchedlidze, KIT

Quadtrees

Def.: A **quadtree** is a rooted tree, where each internal node has 4 children. Each node corresponds to a square, and the squares of the leaves form a partition of the root square.

Def.: For a point set P in a square $Q = [x_Q, x'_Q] \times [y_Q, y'_Q]$ define the quadtree $\mathcal{T}(P)$

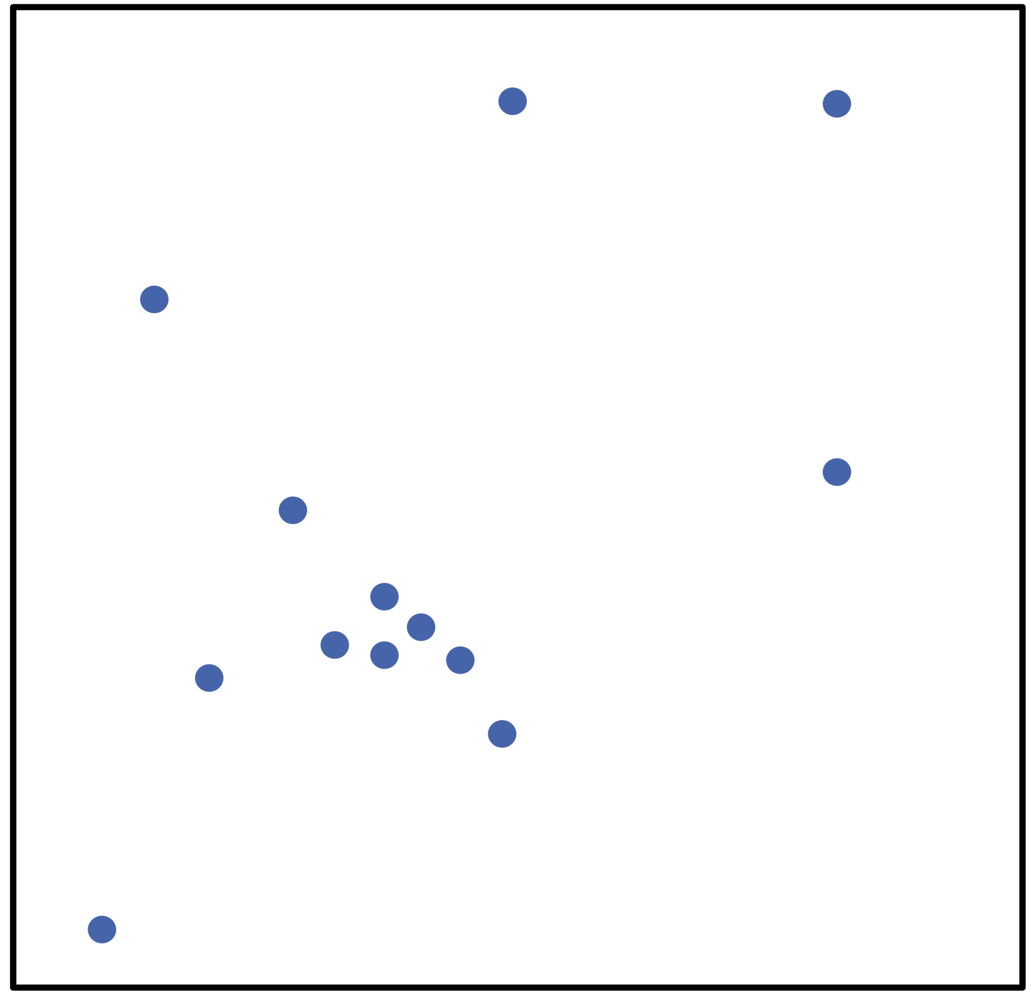
- if $|P| \leq 1$ then $\mathcal{T}(P)$ is a leaf, then Q stores P
- otherwise let $x_{\text{mid}} = \frac{x_Q + x'_Q}{2}$ and $y_{\text{mid}} = \frac{y_Q + y'_Q}{2}$ and

$$\begin{aligned}
P_{NE} &:= \{p \in P \mid p_x > x_{\text{mid}} \text{ and } p_y > y_{\text{mid}}\} \\
P_{NW} &:= \{p \in P \mid p_x \leq x_{\text{mid}} \text{ and } p_y > y_{\text{mid}}\} \\
P_{SW} &:= \{p \in P \mid p_x \leq x_{\text{mid}} \text{ and } p_y \leq y_{\text{mid}}\} \\
P_{SE} &:= \{p \in P \mid p_x > x_{\text{mid}} \text{ and } p_y \leq y_{\text{mid}}\}
\end{aligned}$$

  $\mathcal{T}(P)$ has root v, then Q has 4 children storing P_i and Q_i ($i \in \{NE, NW, SW, SE\}$).

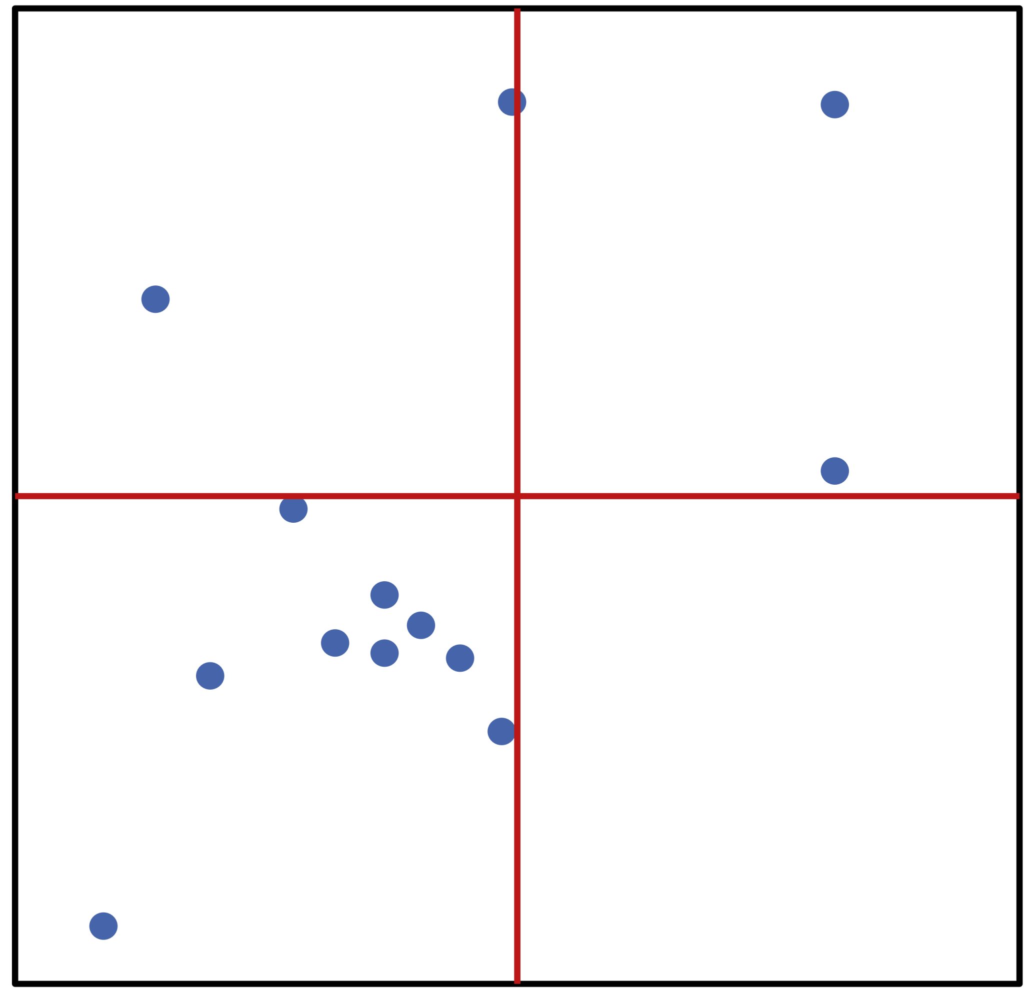
From T. Mchedlidze, KIT

Example



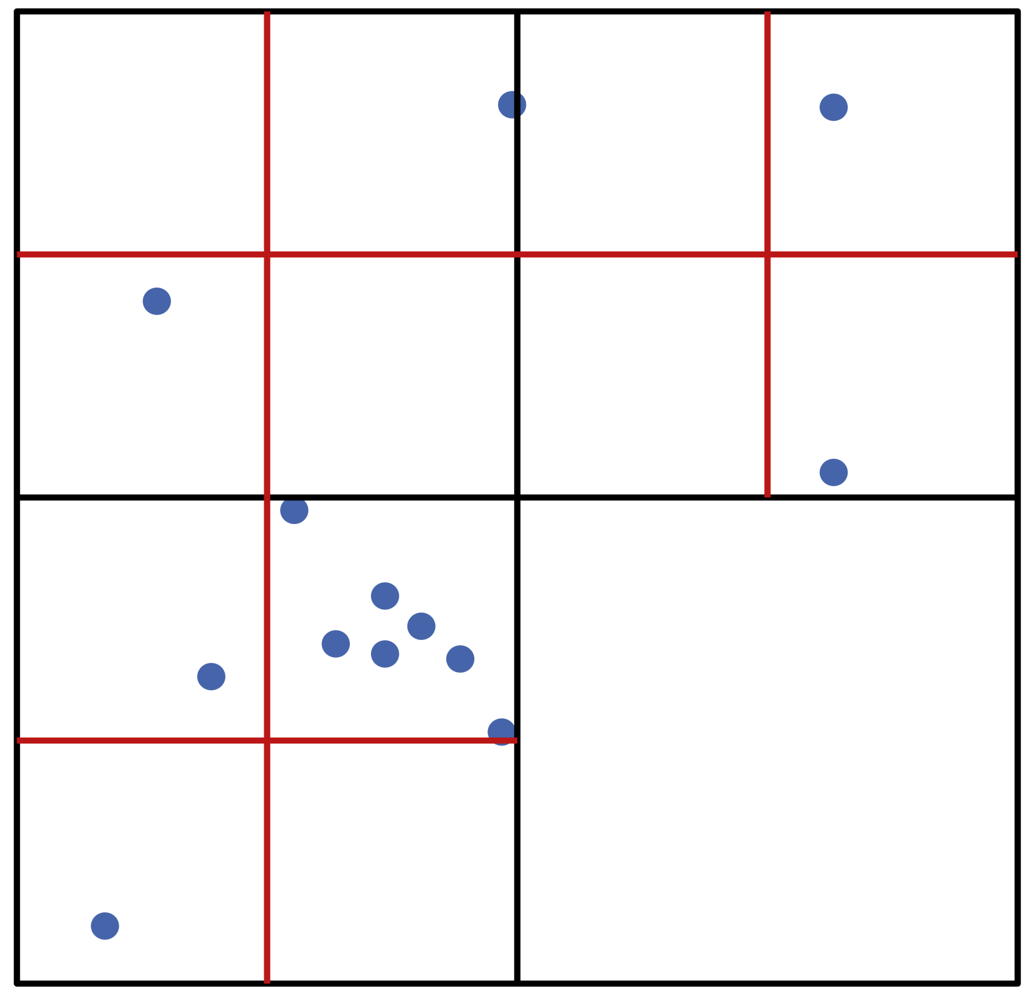
From T. Mchedlidze, KIT

Example



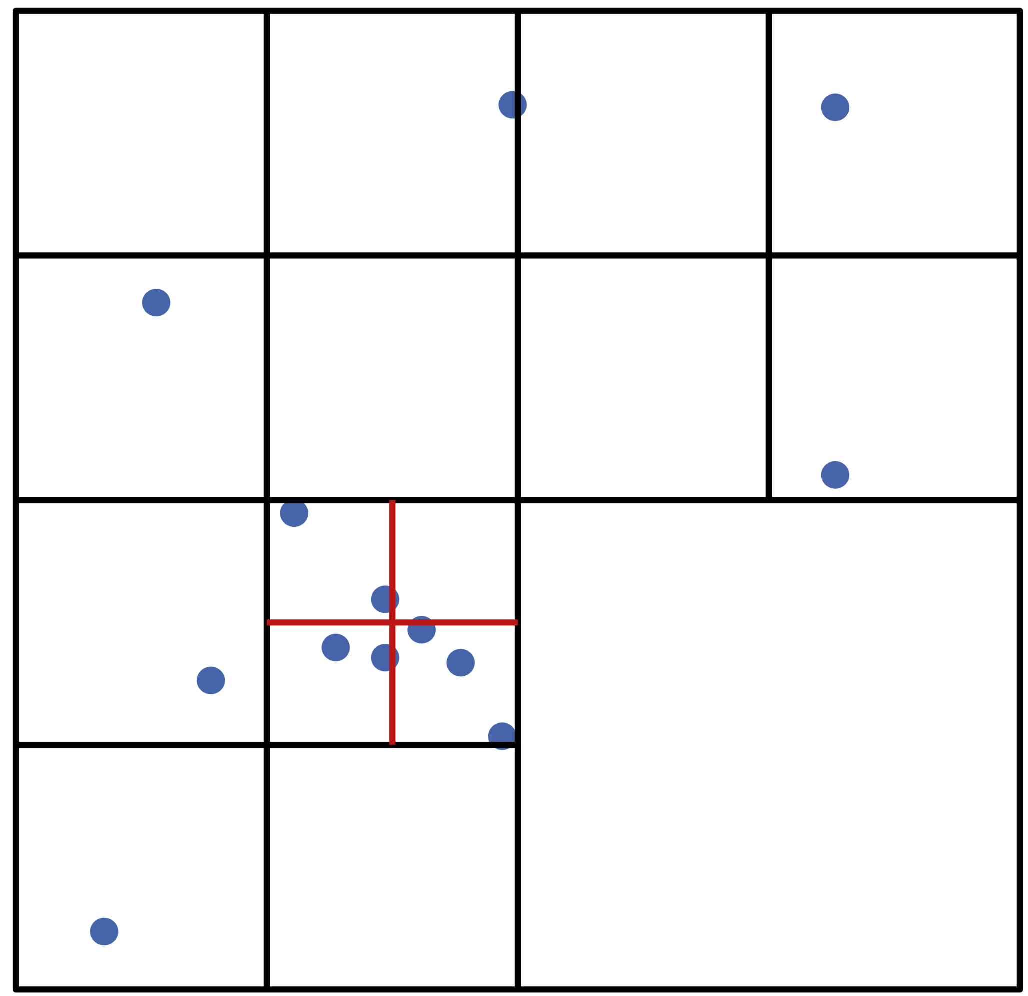
From T. Mchedlidze, KIT

Example



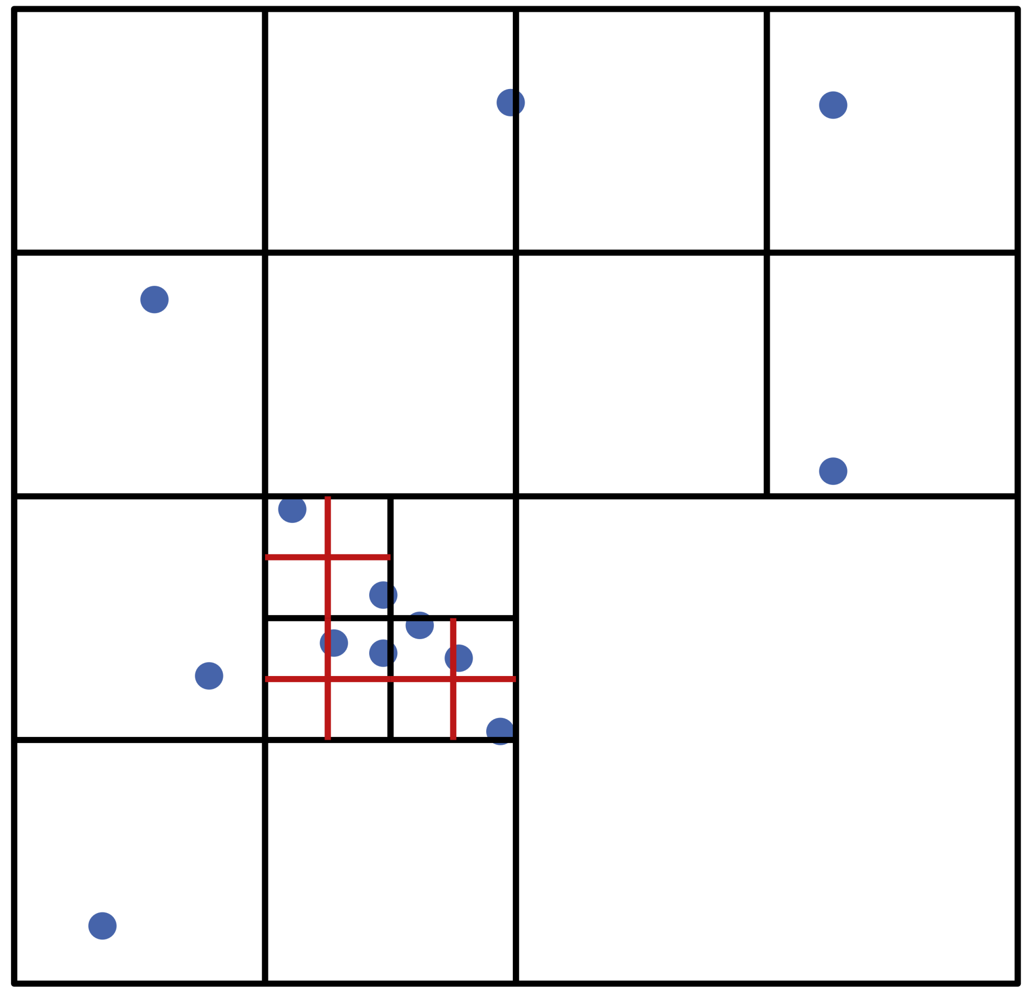
From T. Mchedlidze, KIT

Example



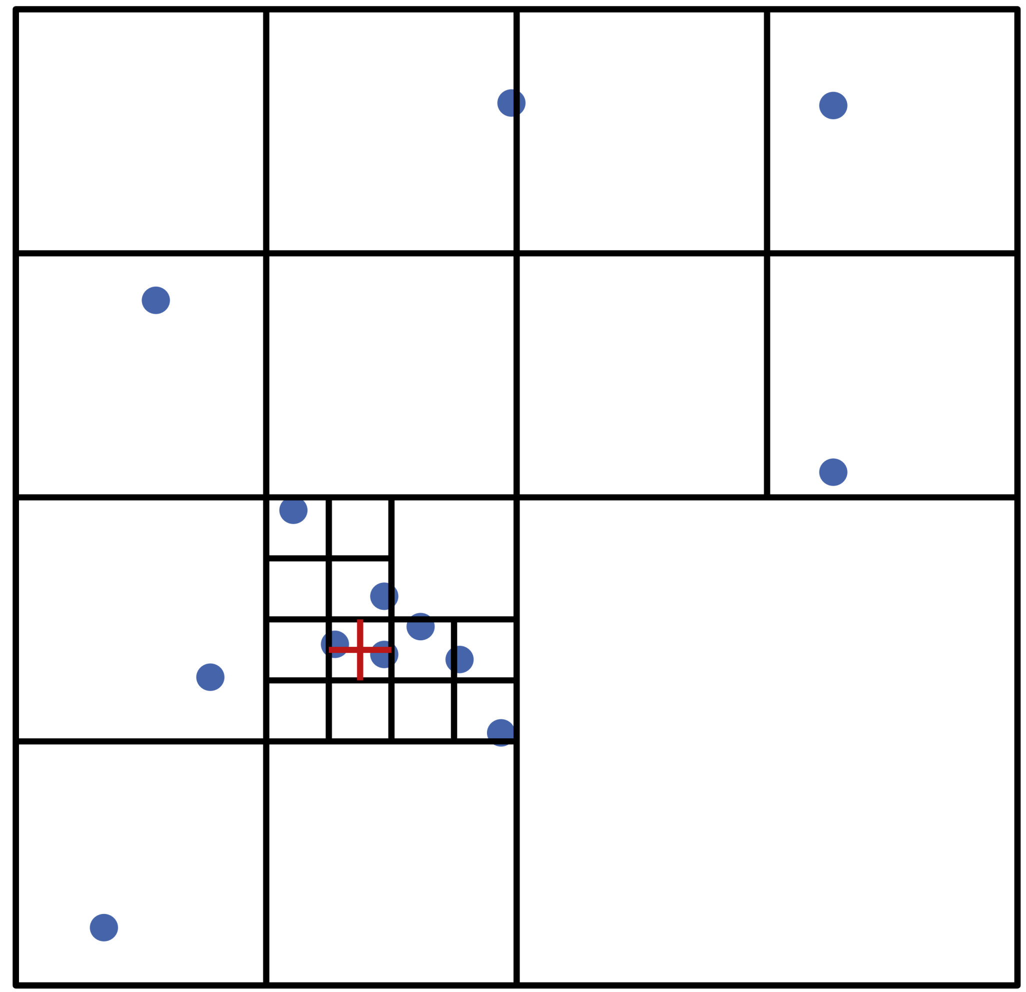
From T. Mchedlidze, KIT

Example



From T. Mchedlidze, KIT

Example



From T. Mchedlidze, KIT

Quadtree Properties

The recursive definition of quadtrees leads directly to an algorithm for constructing them.

From T. Mchedlidze, KIT

Quadtree Properties

The recursive definition of quadtrees leads directly to an algorithm for constructing them.

What is the depth of a quadtree on n points?

Quadtree Properties

The recursive definition of quadtrees leads directly to an algorithm for constructing them.

What is the depth of a quadtree on n points?

Lemma 1: The depth of $\mathcal{T}(P)$ is at most $\log(s/c) + 3/2$, where c is the smallest distance in P and s is the length of a side of Q.

From T. Mchedlidze, KIT

Quadtree Properties

The recursive definition of quadtrees leads directly to an algorithm for constructing them.

What is the depth of a quadtree on n points?

Lemma 1: The depth of $\mathcal{T}(P)$ is at most $\log(s/c) + 3/2$, where c is the smallest distance in P and s is the length of a side of Q.

Theorem 1: A quadtree $\mathcal{T}(P)$ on n points with depth d has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

From T. Mchedlidze, KIT

Finding Neighbors

NorthNeighbor($v, \mathcal{T}$)

Input: Nodes v in quadtree $\mathcal{T}$

Output: Deepest node v' not deeper than v with $v'.Q$ to the north.
Neighbor of $v.Q$

if $v = \text{root}(\mathcal{T})$ **then return** nil

$\pi \leftarrow \text{parent}(v)$

if $v = SW\text{-}/SE$-child of π **then return** $NW\text{-}/NE$-child of π

$\mu \leftarrow \text{NorthNeighbor}(\pi, \mathcal{T})$

if $\mu =$ nil or μ leaf **then**
  return μ
else
  if $v = NW\text{-}/NE$-child of π **then return** $SW\text{-}/SE$-child of μ

From T. Mchedlidze, KIT

Finding Neighbors

NorthNeighbor$(v, \mathcal{T})$

Input: Nodes v in quadtree $\mathcal{T}$

Output: Deepest node v' not deeper than v with $v'.Q$ to the north.
Neighbor of $v.Q$

if $v =$ root$(\mathcal{T})$ **then return** nil

$\pi \leftarrow$ parent(v)

if $v = SW$-$/SE$-child of π **then return** NW-$/NE$-child of π

$\mu \leftarrow$ NorthNeighbor$(\pi, \mathcal{T})$

if $\mu =$ nil or μ leaf **then**

  return μ

else

  if $v = NW$-$/NE$-child of π **then return** SW-$/SE$-child of μ

Theorem 2: Let $\mathcal{T}$ be a quadtree with depth d. The neighbor of a node v in any direction can be found in $O(d+1)$ time.

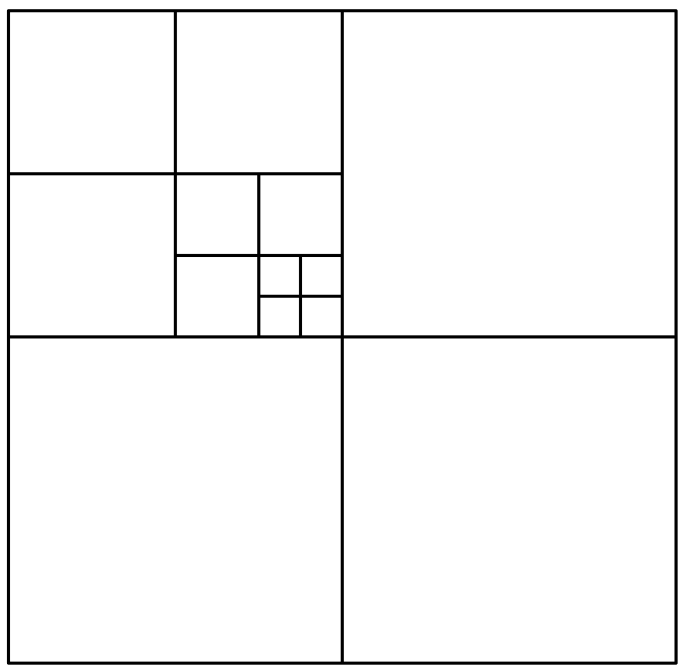
Balanced Subtrees

Def.: A quadtree is called **balanced** if any two neighboring squares differ at most a factor two in size. A quadtree is called balanced if its subdivision is balanced.

From T. Mchedlidze, KIT

Balanced Quadtrees

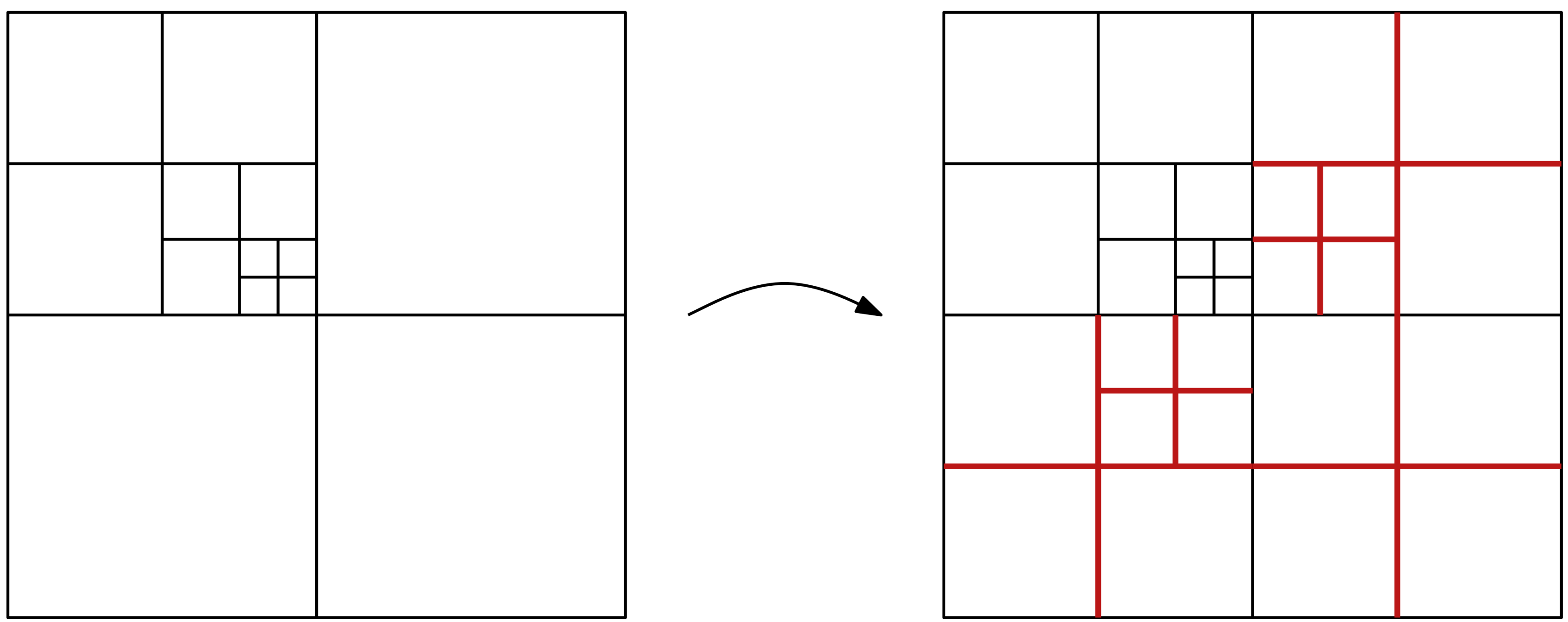
Def.: A quadtree is called **balanced** if any two neighboring squares differ at most a factor two in size. A quadtree is called balanced if its subdivision is balanced.



From T. Mchedlidze, KIT

Balanced Quadtrees

Def.: A quadtree is called **balanced** if any two neighboring squares differ at most a factor two in size. A quadtree is called balanced if its subdivision is balanced.



From T. Mchedlidze, KIT

Balancing Quadtrees

BalanceQuadtree($\mathcal{T}$)

Input: Quadtree $\mathcal{T}$
Output: A balanced version of $\mathcal{T}$
$L \leftarrow$ List of all leaves of $\mathcal{T}$
while L not empty **do**
  $\mu \leftarrow$ extract leaf from L
  if $\mu.Q$ too large **then**
    Divide $\mu.Q$ into four parts and put four leaves in $\mathcal{T}$
    add new leaves to L
    if $\mu.Q$ now has neighbors that are too large **then** add it to L
  return $\mathcal{T}$

From T. Mchedlidze, KIT

Balancing Quadtrees

BalanceQuadtree($\mathcal{T}$)

Input: Quadtree $\mathcal{T}$

Output: A balanced version of $\mathcal{T}$

$L \leftarrow$ List of all leaves of $\mathcal{T}$

while L not empty **do**

  $\mu \leftarrow$ extract leaf from L

  if $\mu.Q$ too large **then**

    Divide $\mu.Q$ into four parts and put four leaves in $\mathcal{T}$

    add new leaves to L

    if $\mu.Q$ now has neighbors that are too large **then** add it to L

  return $\mathcal{T}$

How?

How?

From T. Mchedlidze, KIT

Balanicng Quadtrees

BalanceQuadtree($\mathcal{T}$)

Input: Quadtree $\mathcal{T}$
Output: A balanced version of $\mathcal{T}$
$L \leftarrow$ List of all leaves of $\mathcal{T}$
while L not empty **do**
  $\mu \leftarrow$ extract leaf from L
  if $\mu.Q$ too large **then**
    Divide $\mu.Q$ into four parts and put four leaves in $\mathcal{T}$
    add new leaves to L
    if $\mu.Q$ now has neighbors that are too large **then** add it to L
  return $\mathcal{T}$

How?

How?

How large can a balanced quadtree be?

From T. Mchedlidze, KIT

Saarland Informatics Campus

Balancing Quadtrees

BalanceQuadtree($\mathcal{T}$)

Input: Quadtree $\mathcal{T}$
Output: A balanced version of $\mathcal{T}$

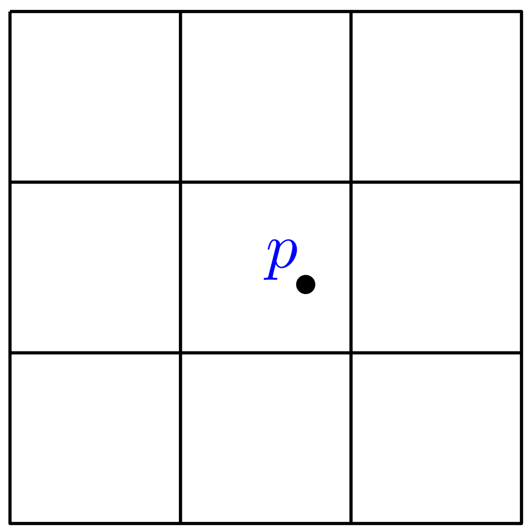
```
L ← List of all leaves of T
while L not empty do
    μ ← extract leaf from L
    if μ.Q too large then
        Divide μ.Q into four parts and put four leaves in T
        add new leaves to L
        if μ.Q now has neighbors that are too large then add it to L
    return T
```

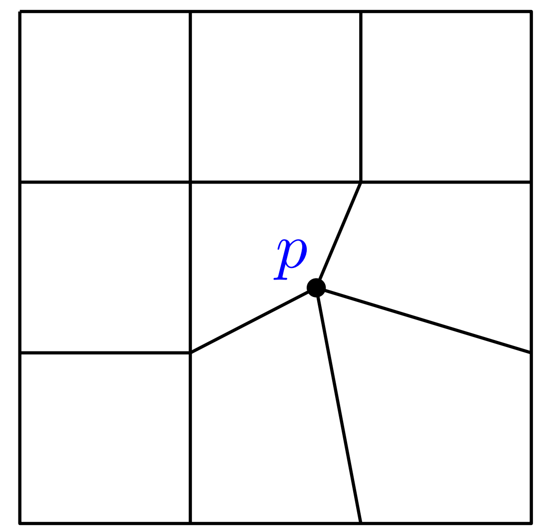
Thm 3: Let $\mathcal{T}$ be a quadtree with m nodes and depth d. The balanced version $\mathcal{T}_B$ of $\mathcal{T}$ has $O(m)$ nodes and can be constructed in $O((d+1)m)$ time.

From T. Mchedlidze, KIT

Quadtrees for Nice Triangulations

Bern, Eppstein, Gilbert: *Provably Good Mesh Generation* (1994)

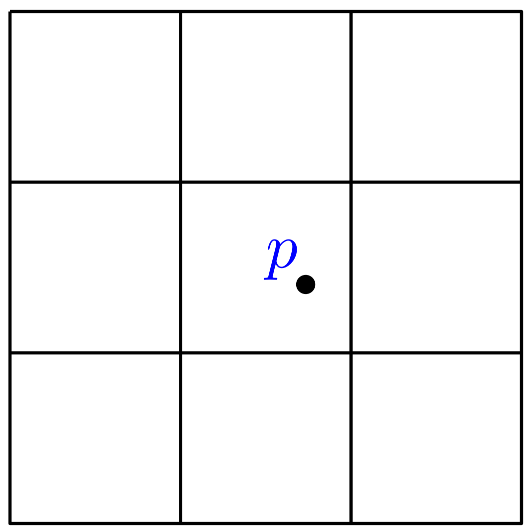


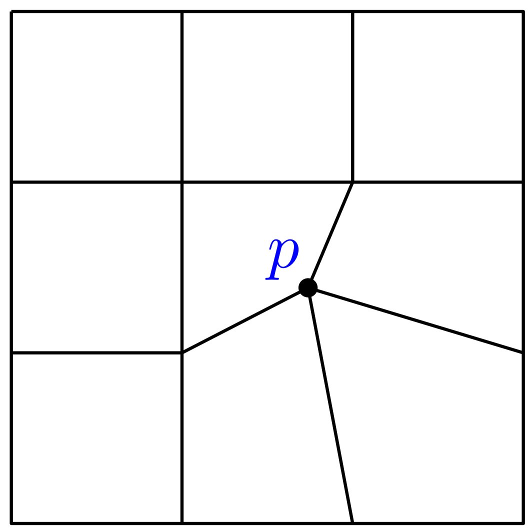


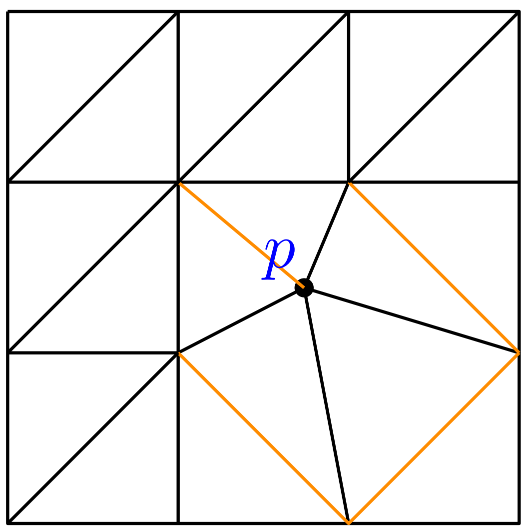
Quadtrees for Nice Triangulations

Bern, Eppstein, Gilbert: *Provably Good Mesh Generation* (1994)

In each quadrilateral incident to p choose diagonal that yields triangles with better aspect ratio.





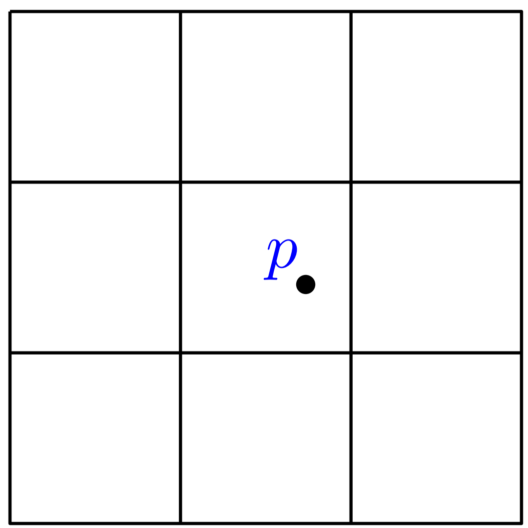


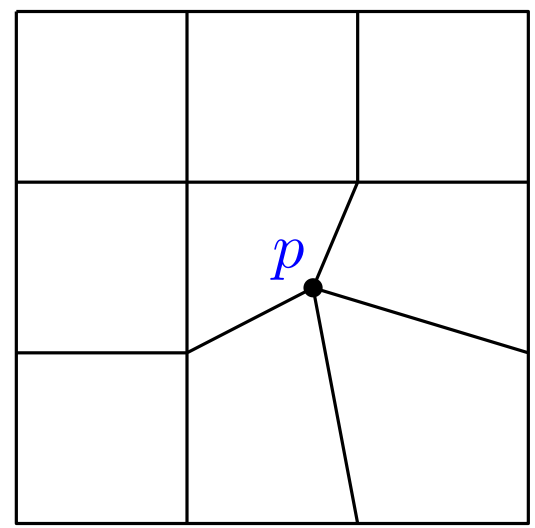
Saarland Informatics Campus

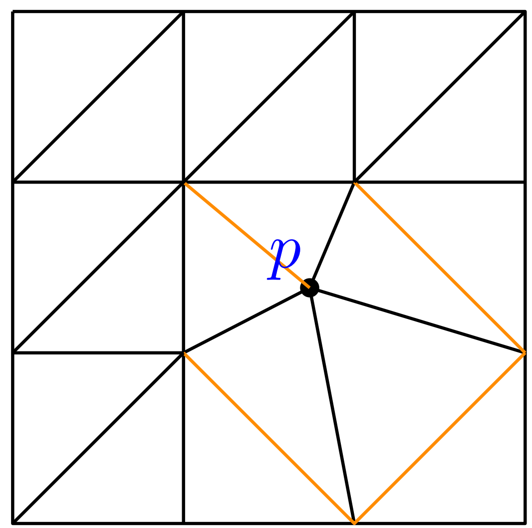
Quadtrees for Nice Triangulations

Bern, Eppstein, Gilbert: *Provably Good Mesh Generation* (1994)

In each quadrilateral incident to p choose diagonal that yields triangles with better aspect ratio.





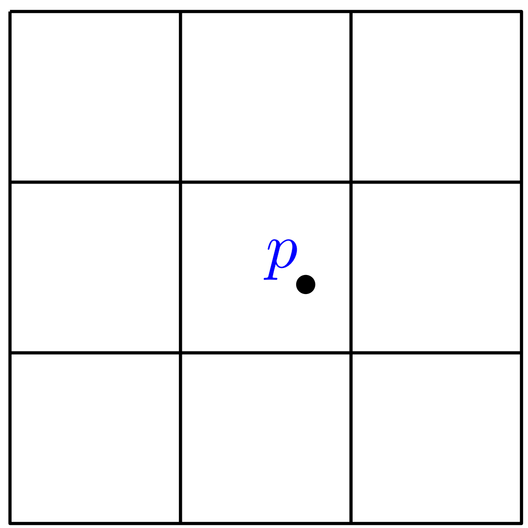


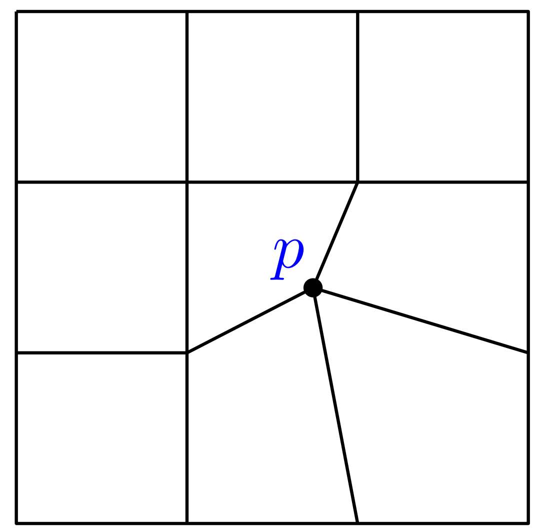
Lemma: For each triangle Δ incident to an orange edge the aspect ratio is at most 4, i.e. $A(\Delta) \leq 4$.

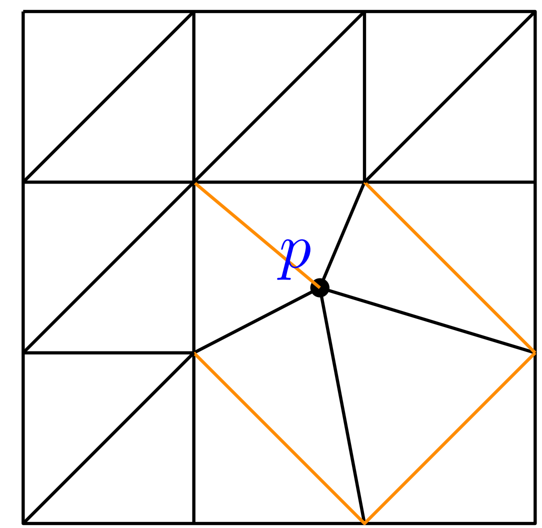
Quadtrees for Nice Triangulations

Bern, Eppstein, Gilbert: *Provably Good Mesh Generation* (1994)

In each quadrilateral incident to p choose diagonal that yields triangles with better aspect ratio.







Point needs one layer of empty boxes around its own box.
Need non-interference between layers of different points.

Splitting Crowded Boxes

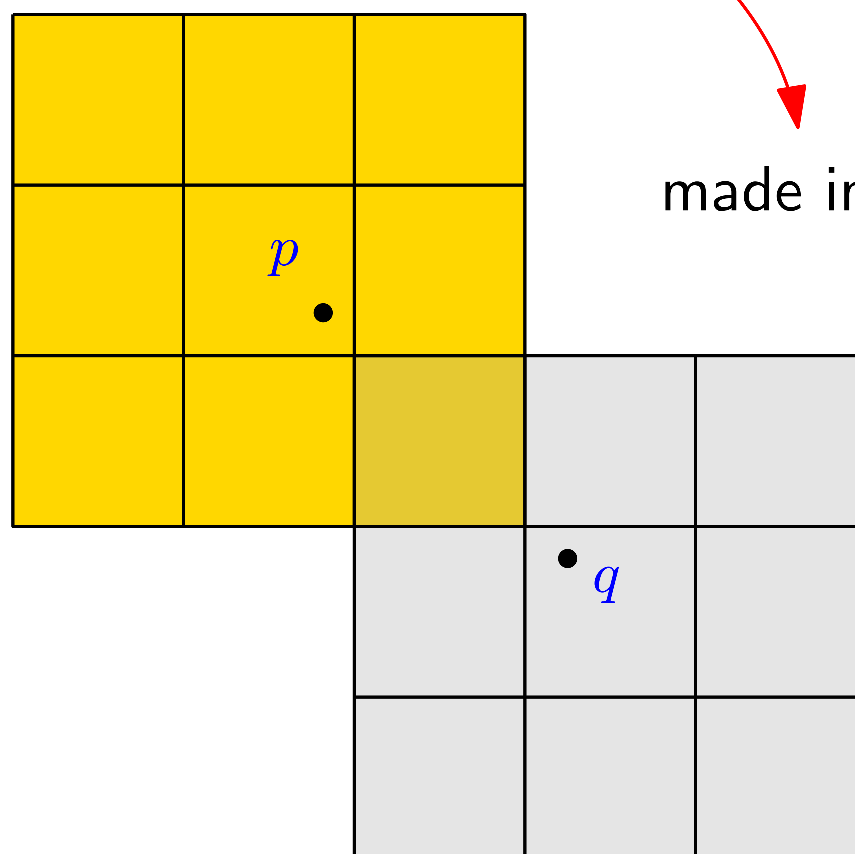
Box b is crowded if at least one of the following holds:

- b contains more than one point of S
- b has side length ℓ, contains a single point $p \in S$, but some other point of S is closer than $2\sqrt{2}\ell$ to p.
- b contains one point of S but one of the 8 neighbors around b has a split side.

Splitting Crowded Boxes

Box b is crowded if at least one of the following holds:

- b contains more than one point of S
- b has side length ℓ, contains a single point $p \in S$, but some other point of S is closer than $2\sqrt{2}\ell$ to p.
- b contains one point of S but one of the 8 neighbors around b has a split side.

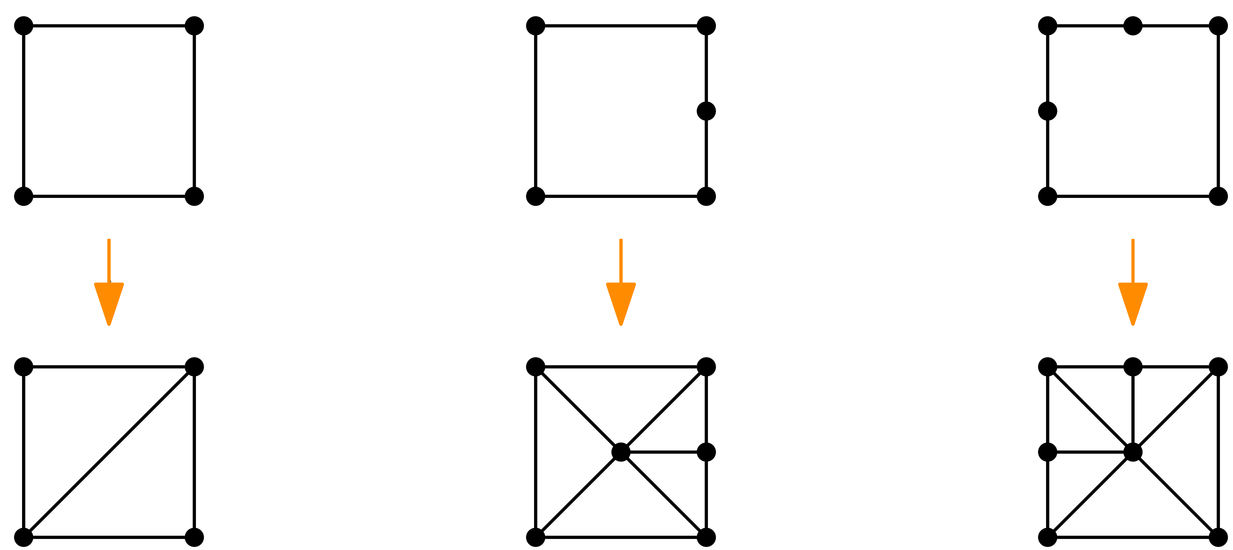


Nice triangulation for S:

0. Put a sufficiently large square box Q around S and make the root of a quadtree.
1. while there is a crowded box, split it and ensure balance
2. for each $p \in S$ move the closest corner of its containing leaf box to p and triangulate the incident quadrilaterals with aspect ratio at most 4
3. triangulate each empty leaf box into at most 8 isosceles right triangles (aspect ratio $\sqrt{2}$)

Nice triangulation for S:

0. Put a sufficiently large square box Q around S and make the root of a quadtree.
1. while there is a crowded box, split it and ensure balance
2. for each $p \in S$ move the closes corner of its containing leaf box to p and traingulate the incident quadrilaterals with aspect ratio at most 4
3. triangulate each empty leaf box into at most 8 isosceles right triangles (aspect ratio $\sqrt{2}$)



Results

Lemma: This algorithm produces a triangulation $\mathcal{T}$ for S with aspect ration $A(\mathcal{T})$ at most 4.

Results

Lemma: This algorithm produces a triangulation $\mathcal{T}$ for S with aspect ration $A(\mathcal{T})$ at most 4.

Lemma: There is a constant c independent of S so that the size of $\mathcal{T}$ is at most

$$c \cdot \sum_{\Delta \in \mathcal{DT}(S)} \log R(\Delta)$$

where $\mathcal{DT}(S)$ is the Delaunay triangulation of S, and $R(\Delta)$ is the ratio of longest and shortest side of triangle Δ.

Results

Lemma: This algorithm produces a triangulation $\mathcal{T}$ for S with aspect ration $A(\mathcal{T})$ at most 4.

Lemma: There is a constant c independent of S so that the size of $\mathcal{T}$ is at most

$$c \cdot \sum_{\Delta \in \mathcal{DT}(S)} \log R(\Delta)$$

where $\mathcal{DT}(S)$ is the Delaunay triangulation of S, and $R(\Delta)$ is the ratio of longest and shortest side of triangle Δ.

Theorem: There is a constant d independent of S so that for any triangulation $\mathcal{T}'$ that has S in its vertex set we have

$$|\mathcal{T}| \leq d \cdot |\mathcal{T}'| \log A(\mathcal{T}')$$

Outlook “Nice Triangulations” (Meshing)

Drawbacks of this result:

- bad constants
- not anisotropic (rotating the coordinate systems changes triangulations)

Viable alternativ algorithms via refining Delaunay triangulations.

Generalization to meshing problems when edges are given as input are possible, but quadtree based approaches have similar shortcomings.

Quadtree based approaches do generalize to higher dimensions.

Outlook Quadtrees

- Quadtrees find many applications in Computer Graphics, Image Processing, Geographic Information Systerm, etc.
  They are useful whenver different scales are to be represented
- There is a variant "compressed quadtree" that uses just space linear in the size of the input.
- "skip quadtrees" allow searches and also updates in logarithmic expected time.
- Quadtrees readily generalize to higher dimensions ("octtree").