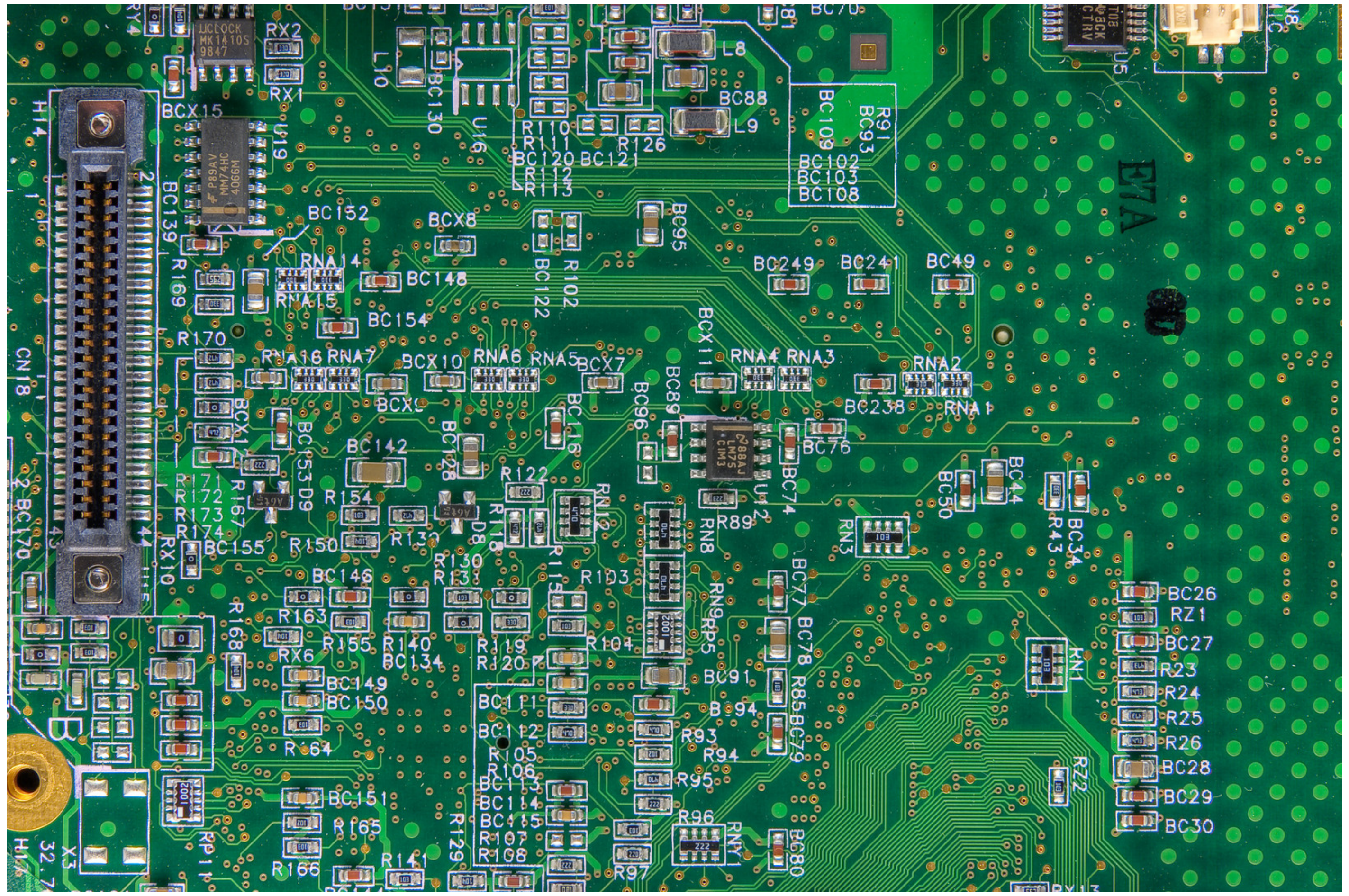
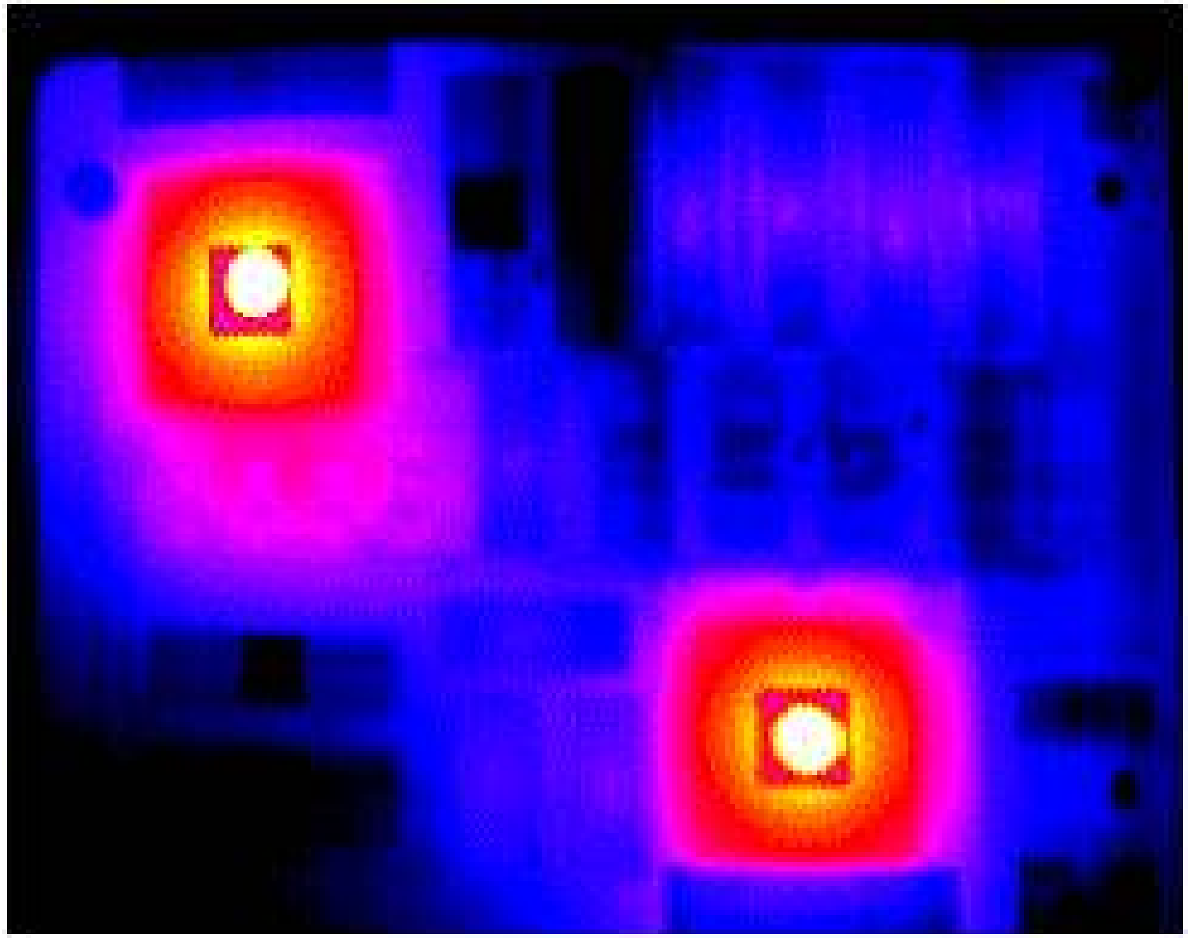
Quadtrees and Meshing

definition & properties
balanced quadtrees
meshing with quadtrees

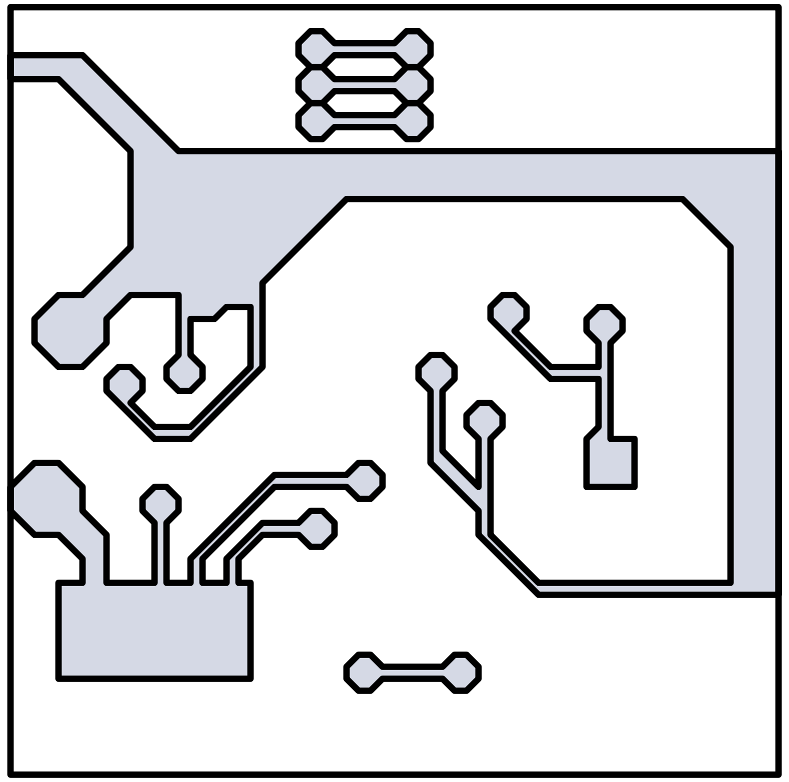
Motivation: VLSI design

Simulation of heat emission on printed circuit boards





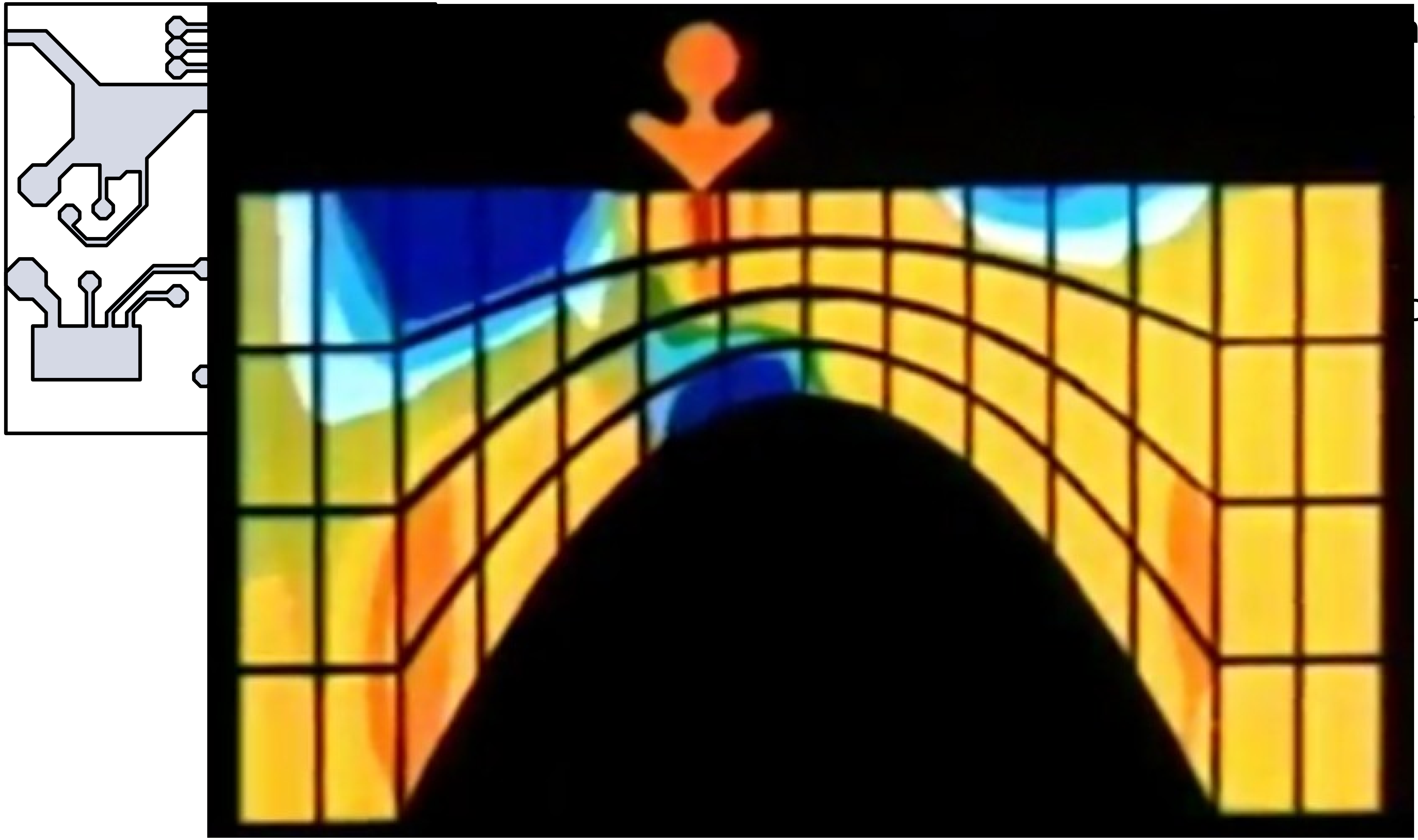
Motivation: VLSI design



To simulate heat emission, use finite element method:

- partition board into small homogeneous elements (e.g. triangles) $\rightarrow$ mesh
- based on heat emission of each element and influence of neighbors numerically approximate the overall heat emission

Motivation: VLSI design



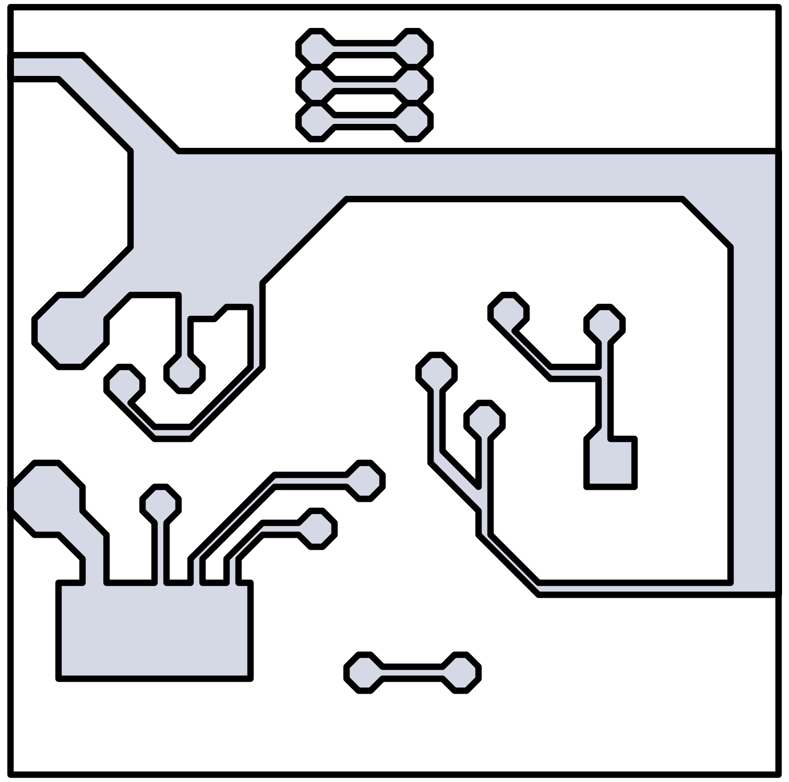
t method:

elements

and

oximate the

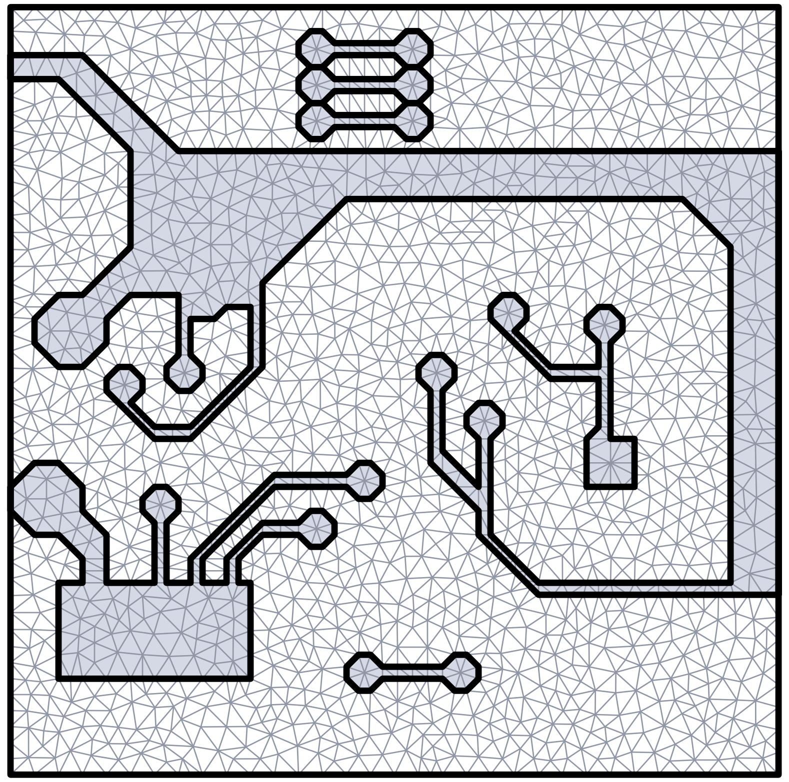
Motivation: VLSI design



To simulate heat emission, use finite element method:

- partition board into small homogeneous elements (e.g. triangles) $\rightarrow$ mesh
- based on heat emission of each element and influence of neighbors numerically approximate the overall heat emission

Motivation: VLSI design



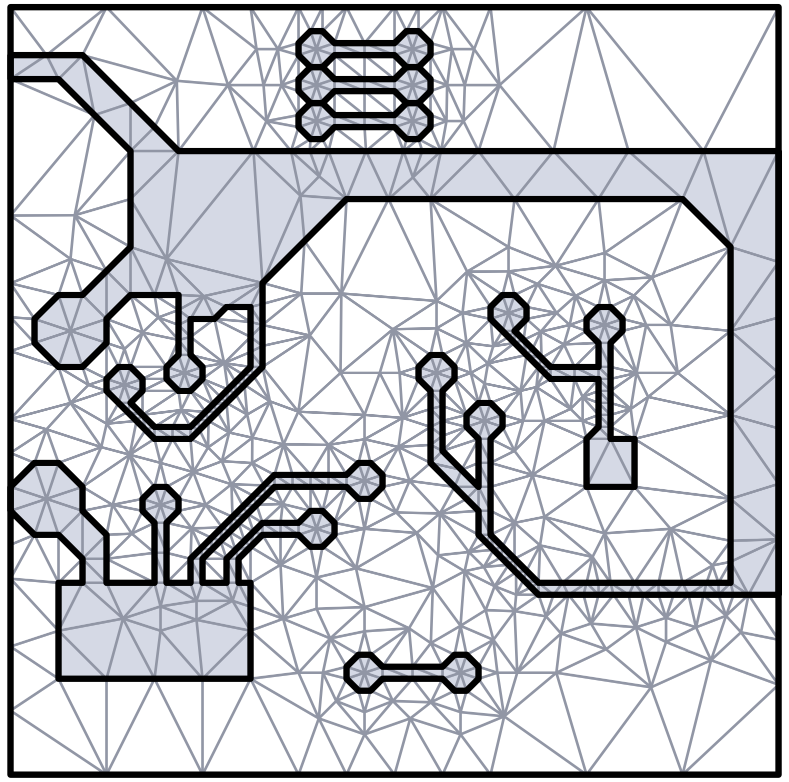
To simulate heat emission, use finite element method:

- partition board into small homogeneous elements (e.g. triangles) $\longrightarrow$ mesh
- based on heat emission of each element and influence of neighbors numerically approximate the overall heat emission

quality criteria:

- finer mesh $\longrightarrow$ better approximation
- coarser mesh $\longrightarrow$ faster computation
- compact elements $\longrightarrow$ faster convergence

Motivation: VLSI design



To simulate heat emission, use finite element method:

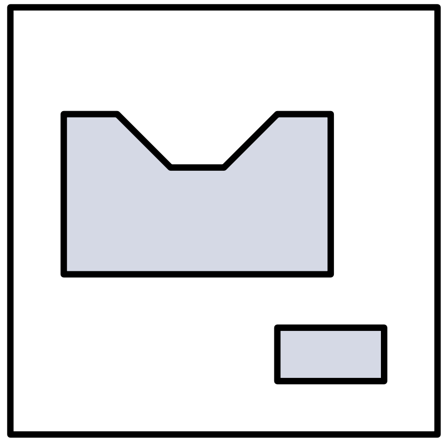
- partition board into small homogeneous elements (e.g. triangles) $\longrightarrow$ mesh
- based on heat emission of each element and influence of neighbors numerically approximate the overall heat emission

quality criteria:
- finer mesh $\longrightarrow$ better approximation
- coarser mesh $\longrightarrow$ faster computation
- compact elements $\longrightarrow$ faster convergence

goal:
- non-uniform mesh $\longrightarrow$ small at boundaries, larger otherwise
- well-shaped triangles $\longrightarrow$ not too thin

Non-uniform meshes

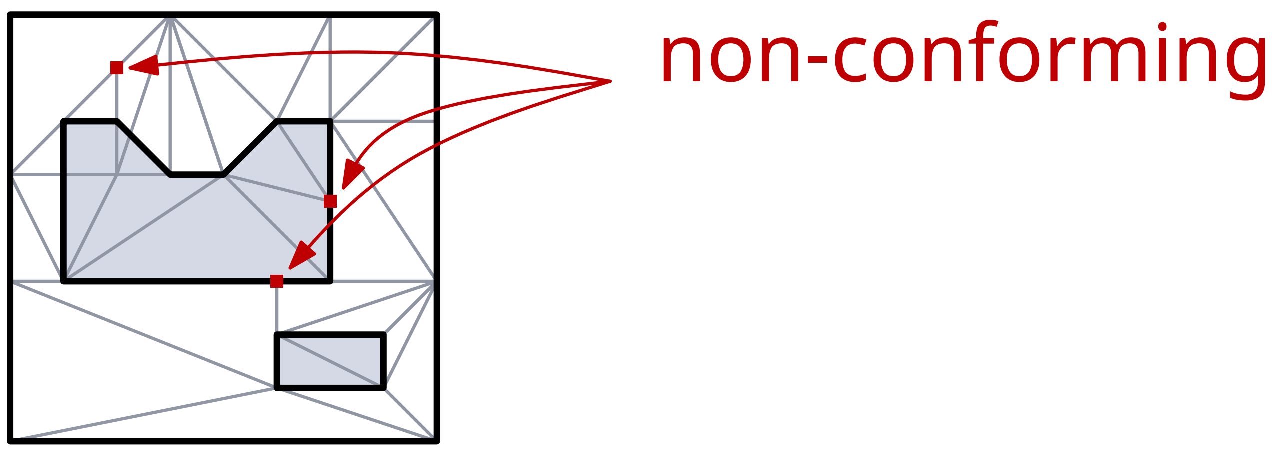
Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two.



Goal: triangular mesh of Q with the following properties

Non-uniform meshes

Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two.

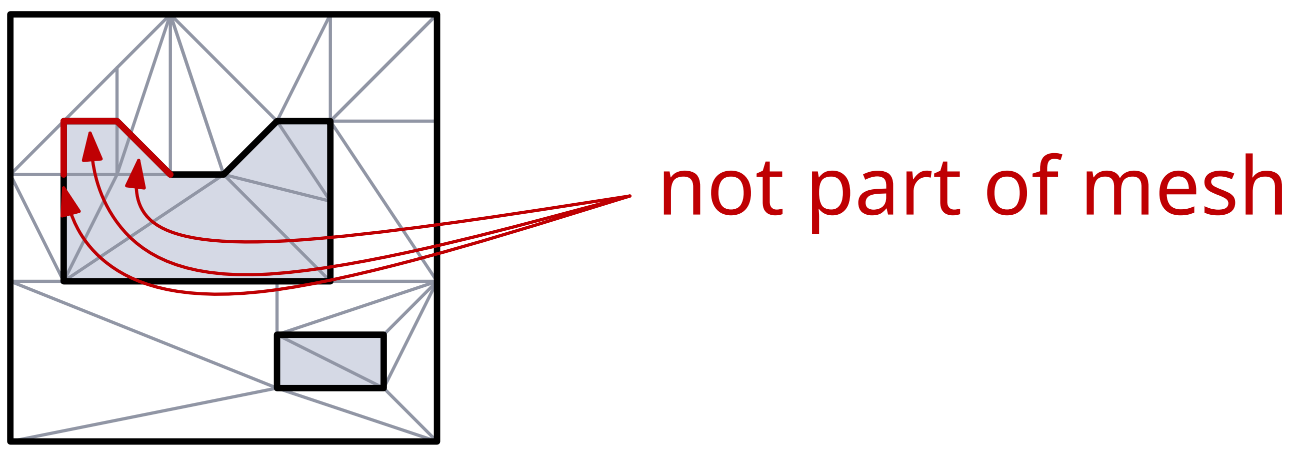


Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges

Non-uniform meshes

Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two.

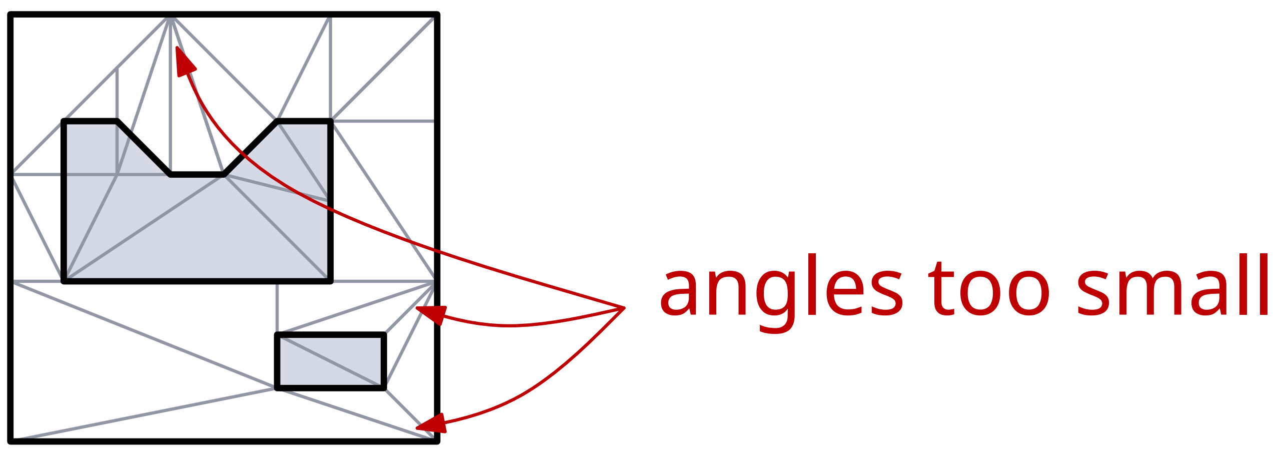


Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges

Non-uniform meshes

Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two.

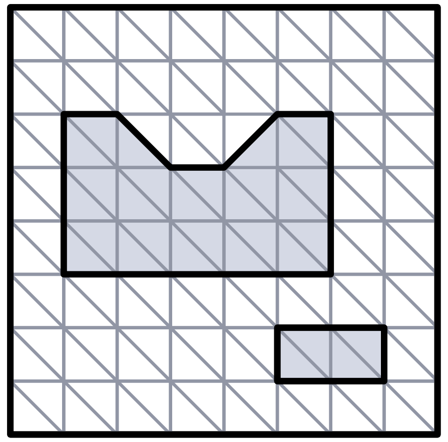


Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges
- well-shaped: angles between 45° and 90°

Non-uniform meshes

Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two.



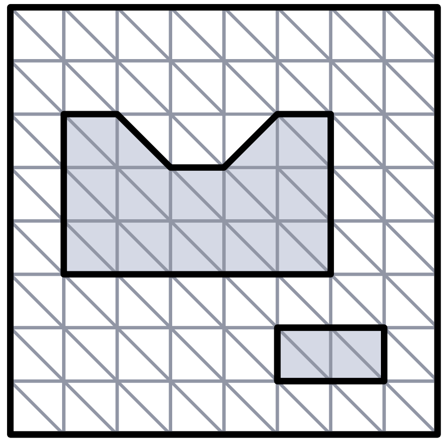
uniform mesh

Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges
- well-shaped: angles between 45° and 90°
- non-uniform: fine near boundaries, coarse otherwise

Non-uniform meshes

Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two.



uniform mesh

Do we need Steiner points (i.e. non-input vertices)?

Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges
- well-shaped: angles between 45° and 90°
- non-uniform: fine near boundaries, coarse otherwise

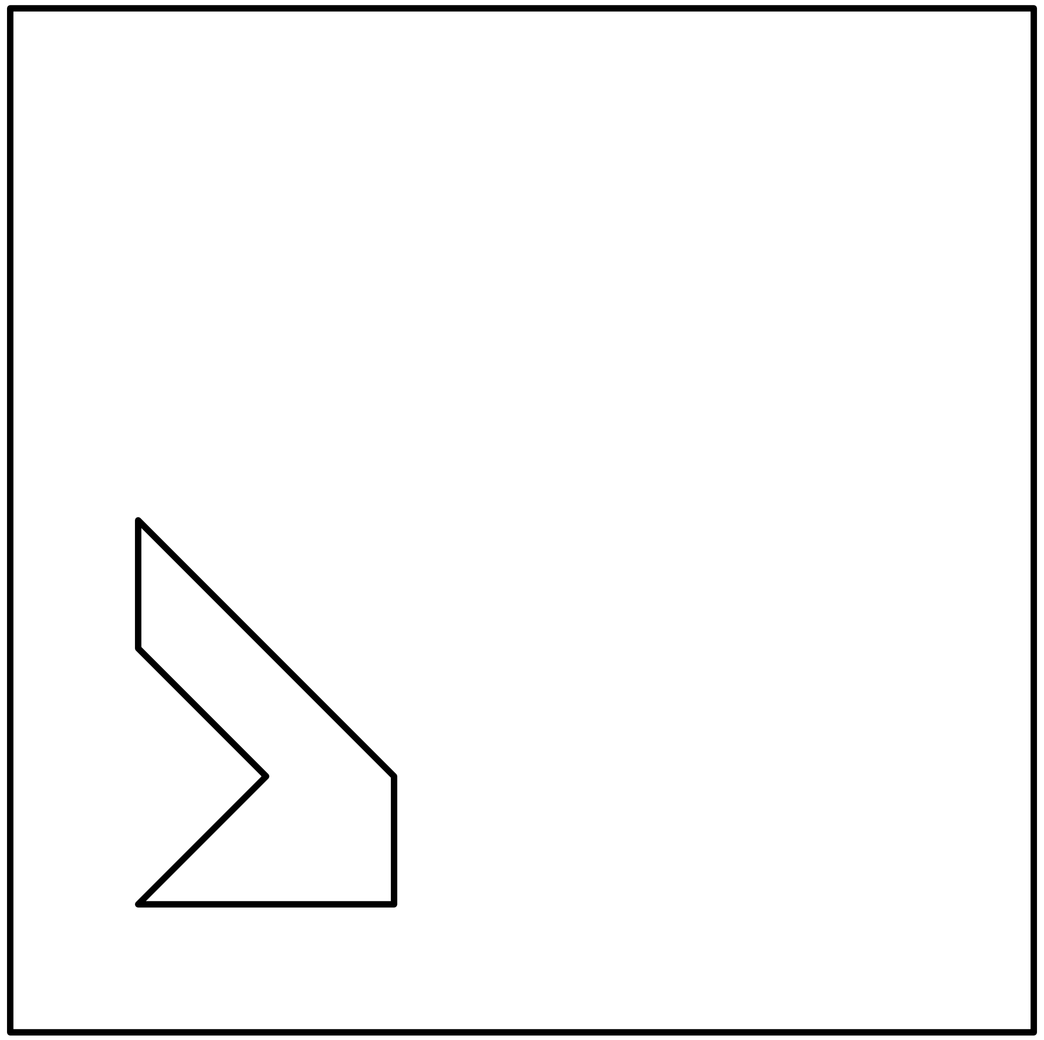
Triangulation of subdivision?

- maximize smallest angle?

Triangulation of subdivision?

- maximize smallest angle?

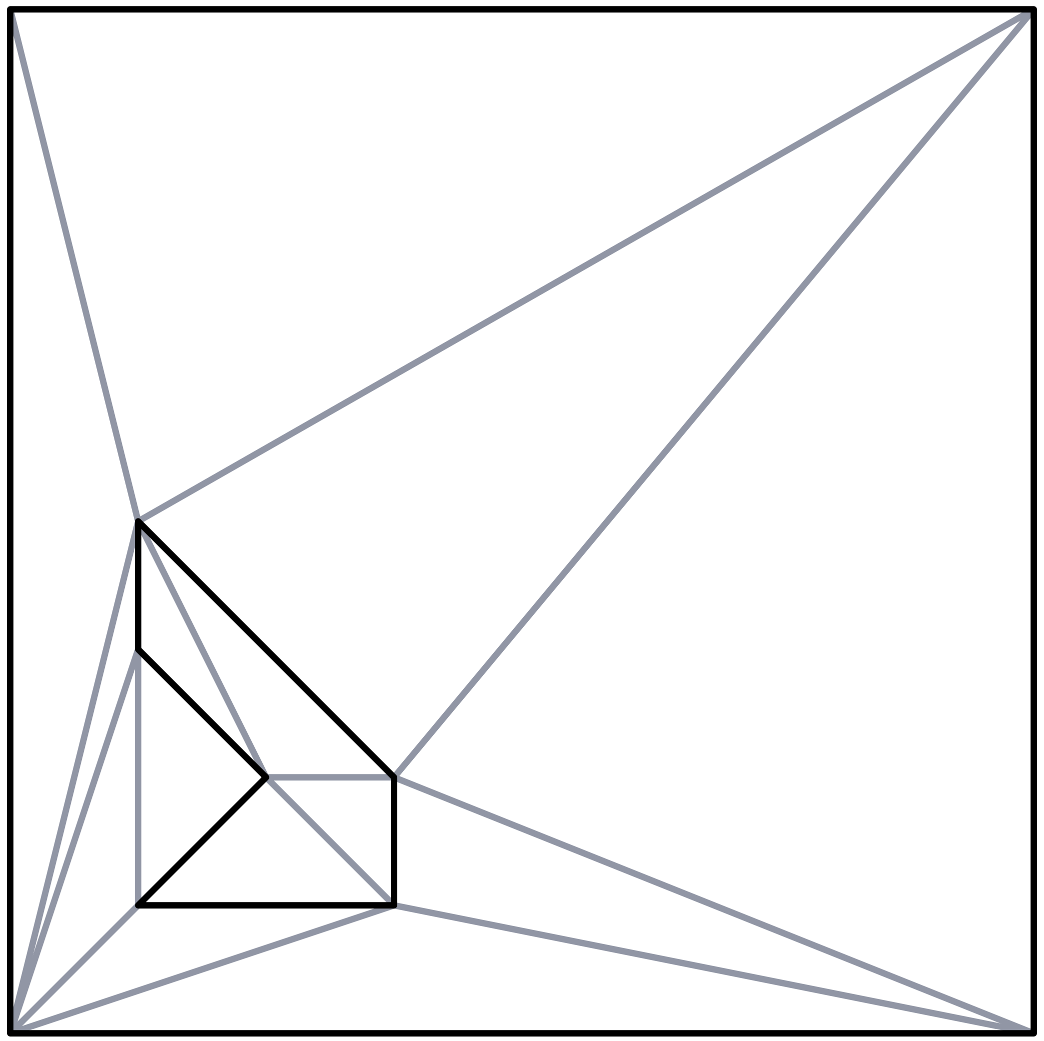
Exercise:



Triangulation of subdivision?

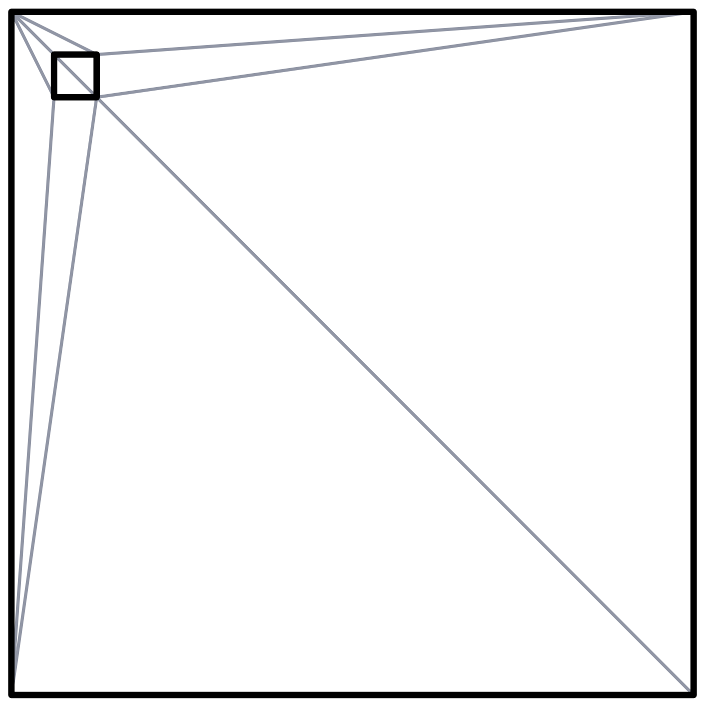
- maximize smallest angle?

Exercise:



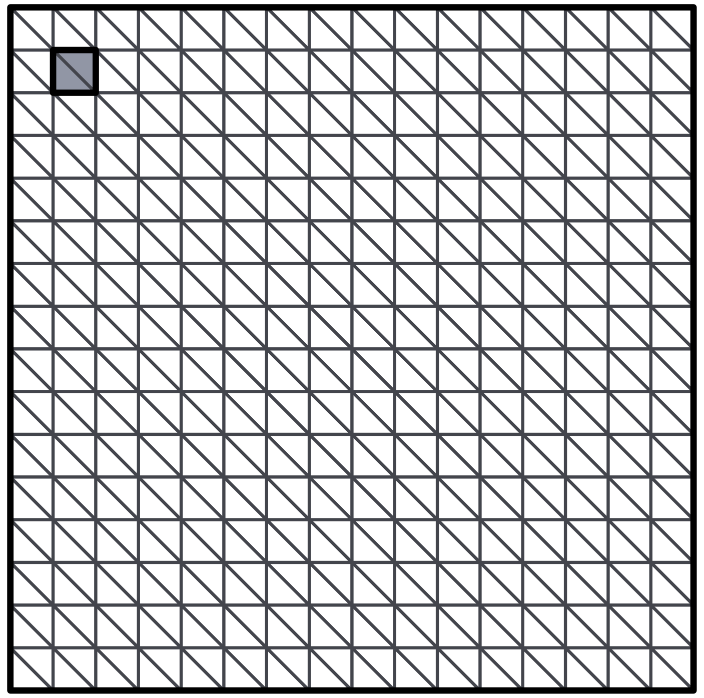
Triangulation of subdivision?

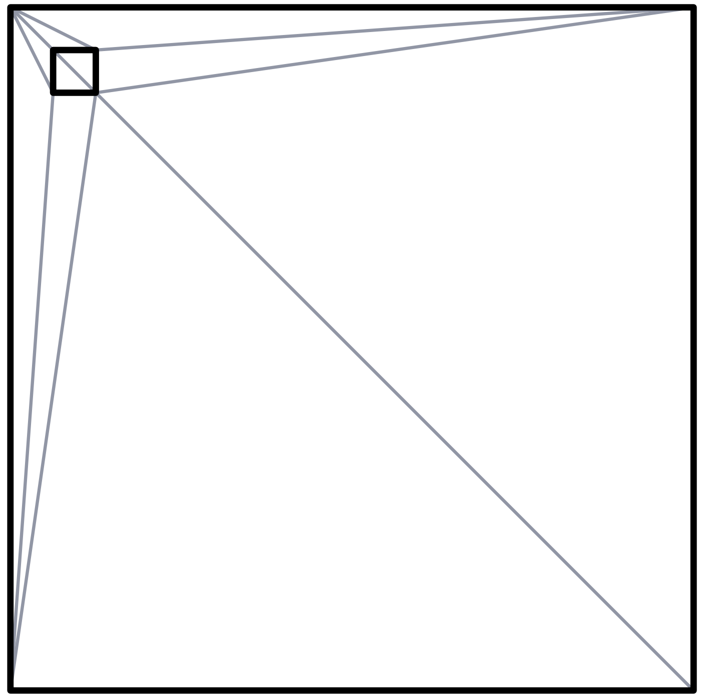
- maximize smallest angle?
- without Steiner points: might have very small angles



Triangulation of subdivision?

- maximize smallest angle?
- without Steiner points: might have very small angles
- with Steiner points:

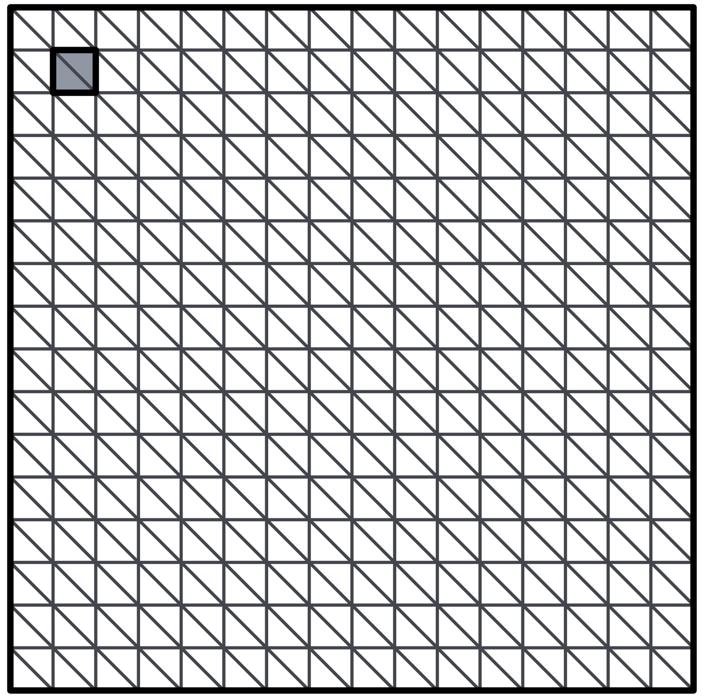




Triangulation of subdivision?

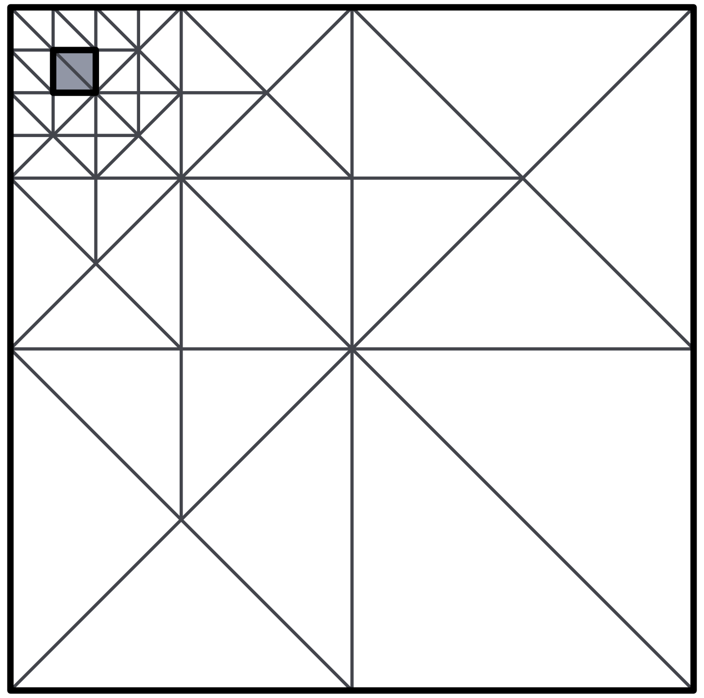
- maximize smallest angle?
- without Steiner points: might have very small angles
- with Steiner points:

well-shaped, but uniform



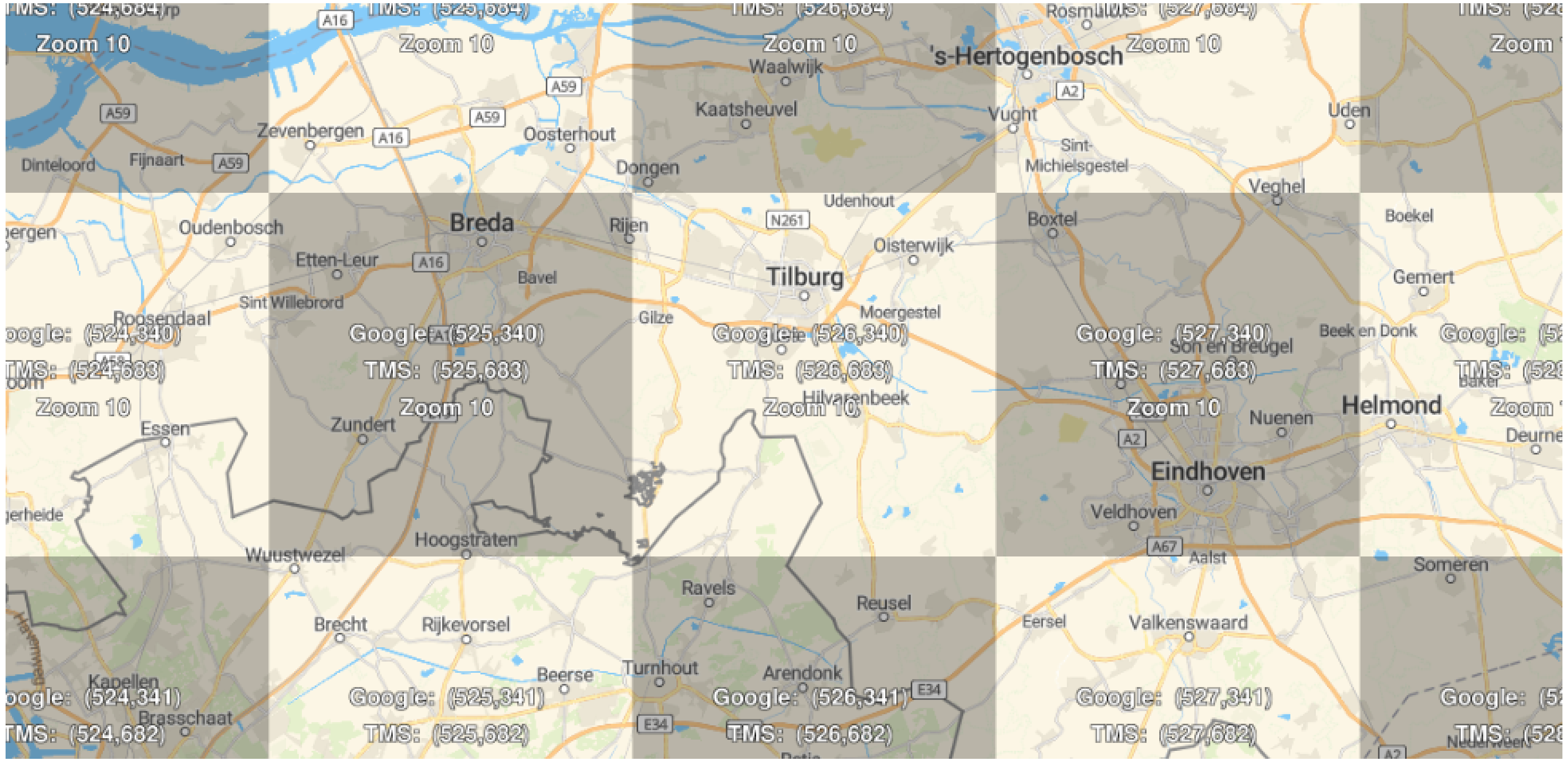
512 triangles

well-shaped, non-uniform



52 triangles

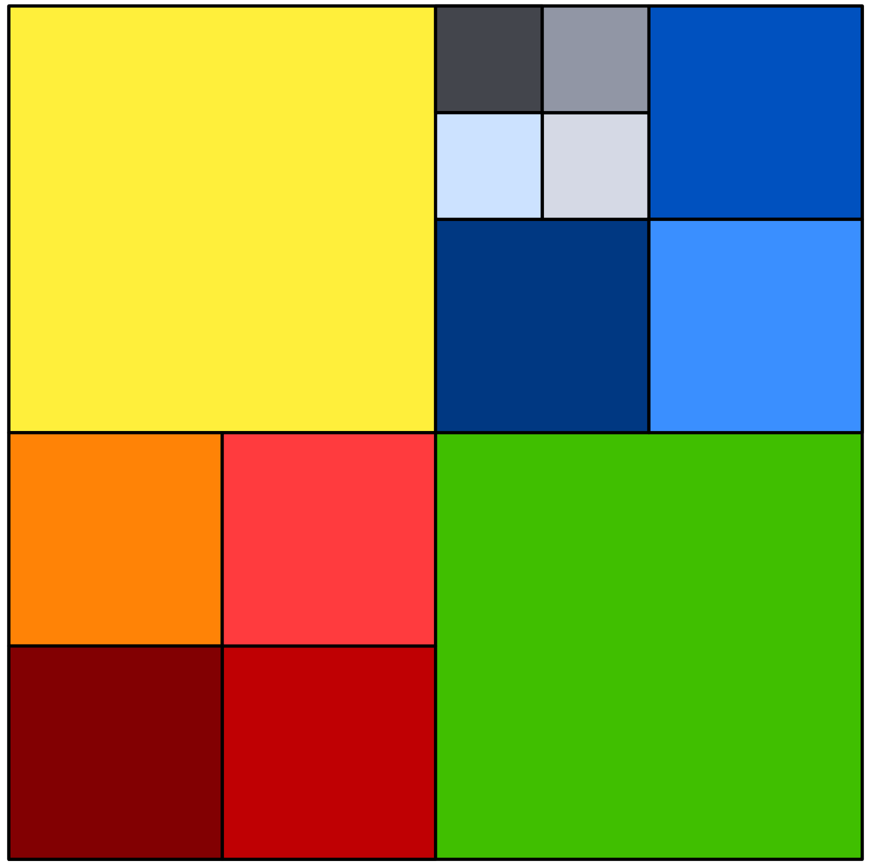
Quadtrees

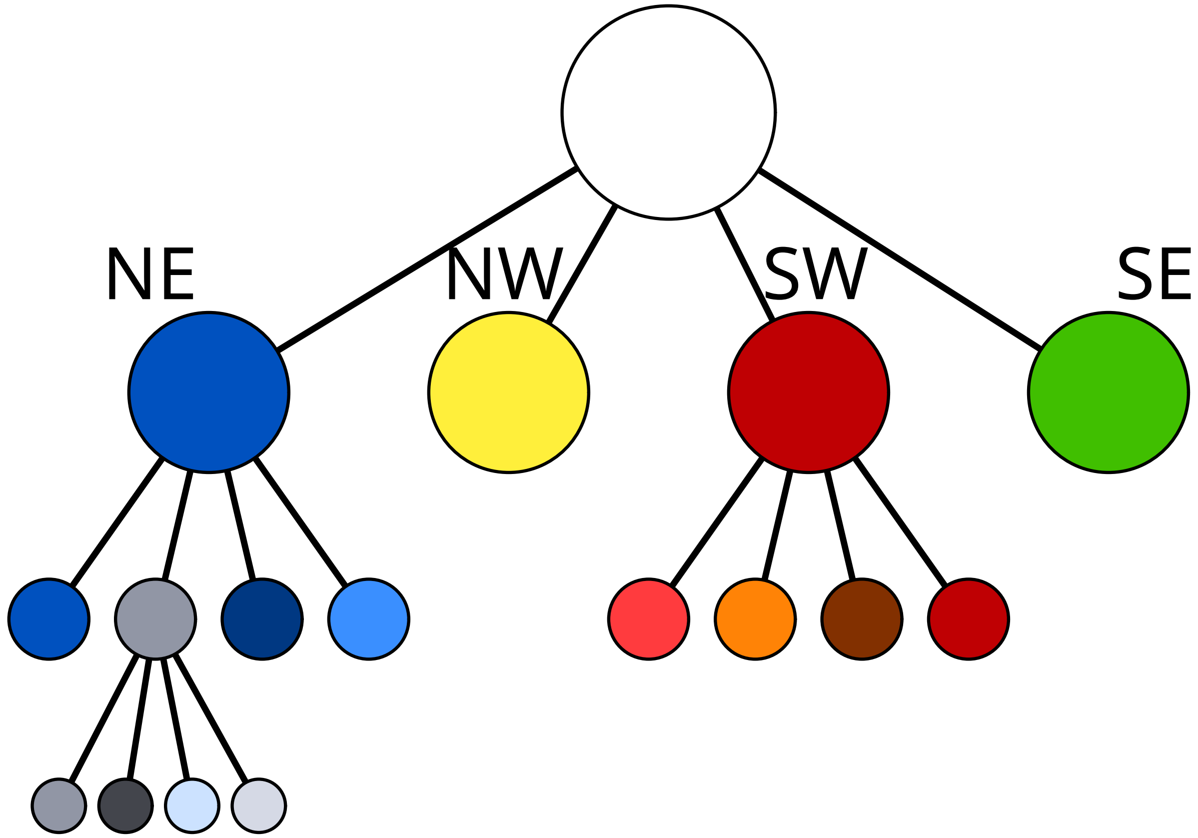


https://google.github.io/closure-library/source/closure/goog/demos/quadtree.html, http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection

Quadtrees

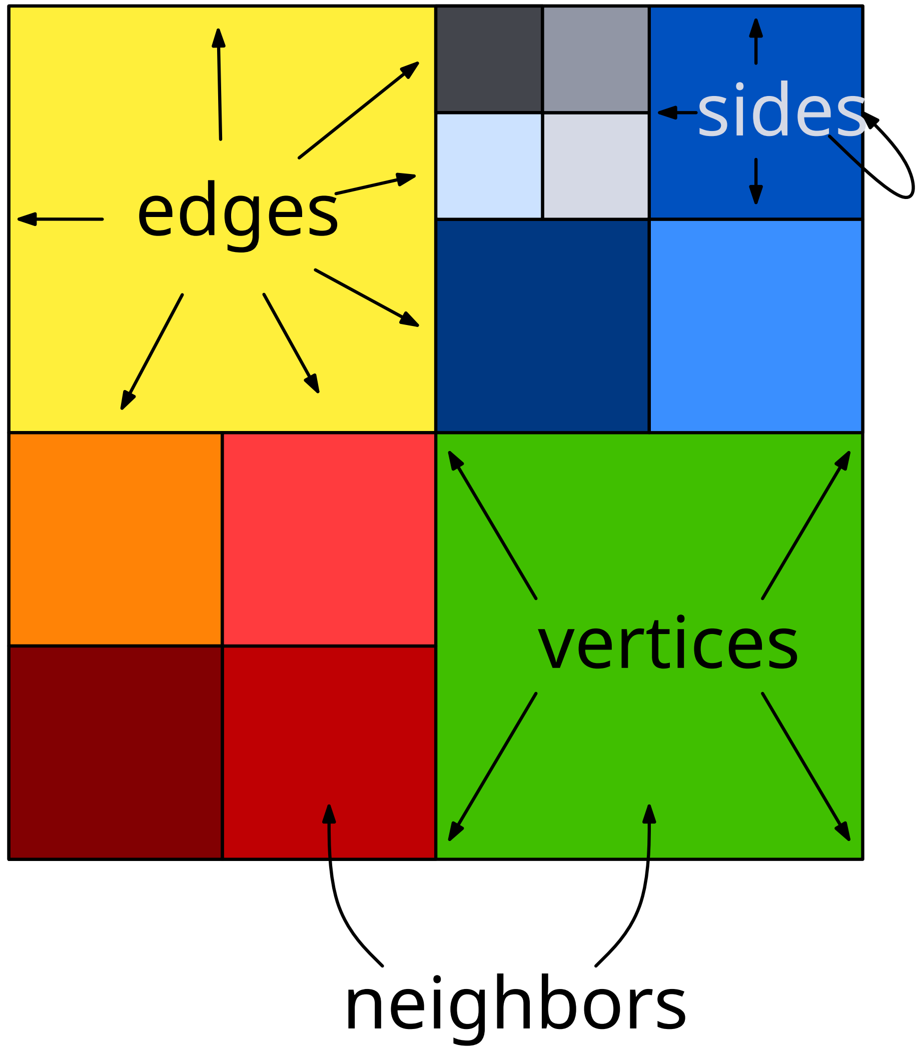
Definition: A quadtree is a rooted tree, in which every interior node has 4 children. Every node corresponds to a square, and the squares of children are the quadrants of the parent's square.

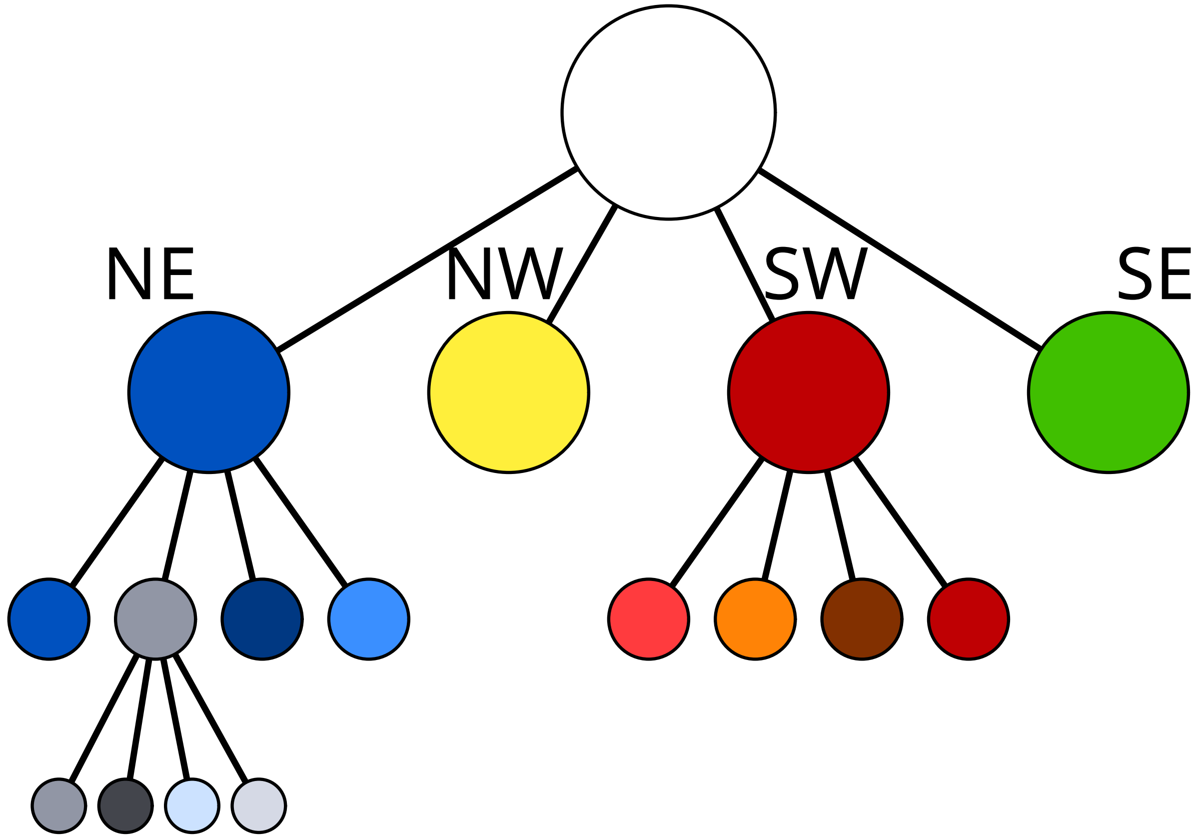




Quadtrees

Definition: A quadtree is a rooted tree, in which every interior node has 4 children. Every node corresponds to a square, and the squares of children are the quadrants of the parent's square.





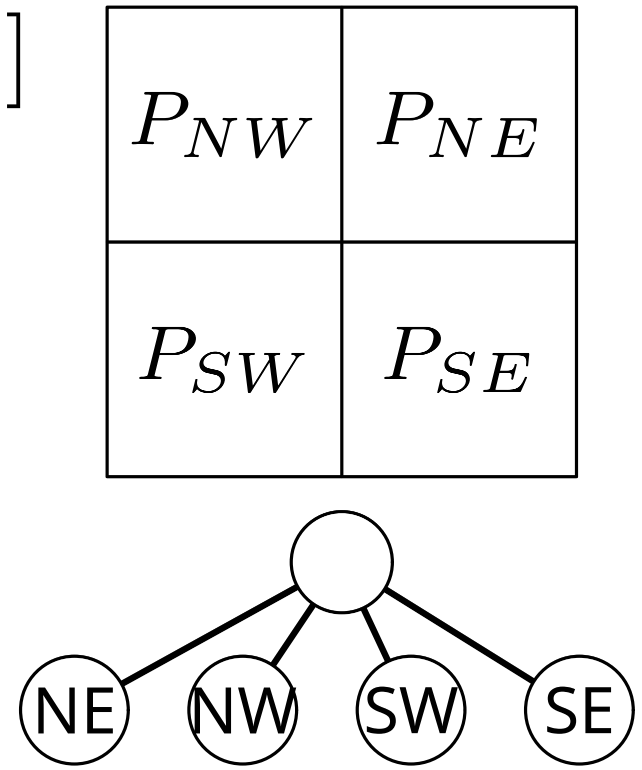
Quadtrees

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

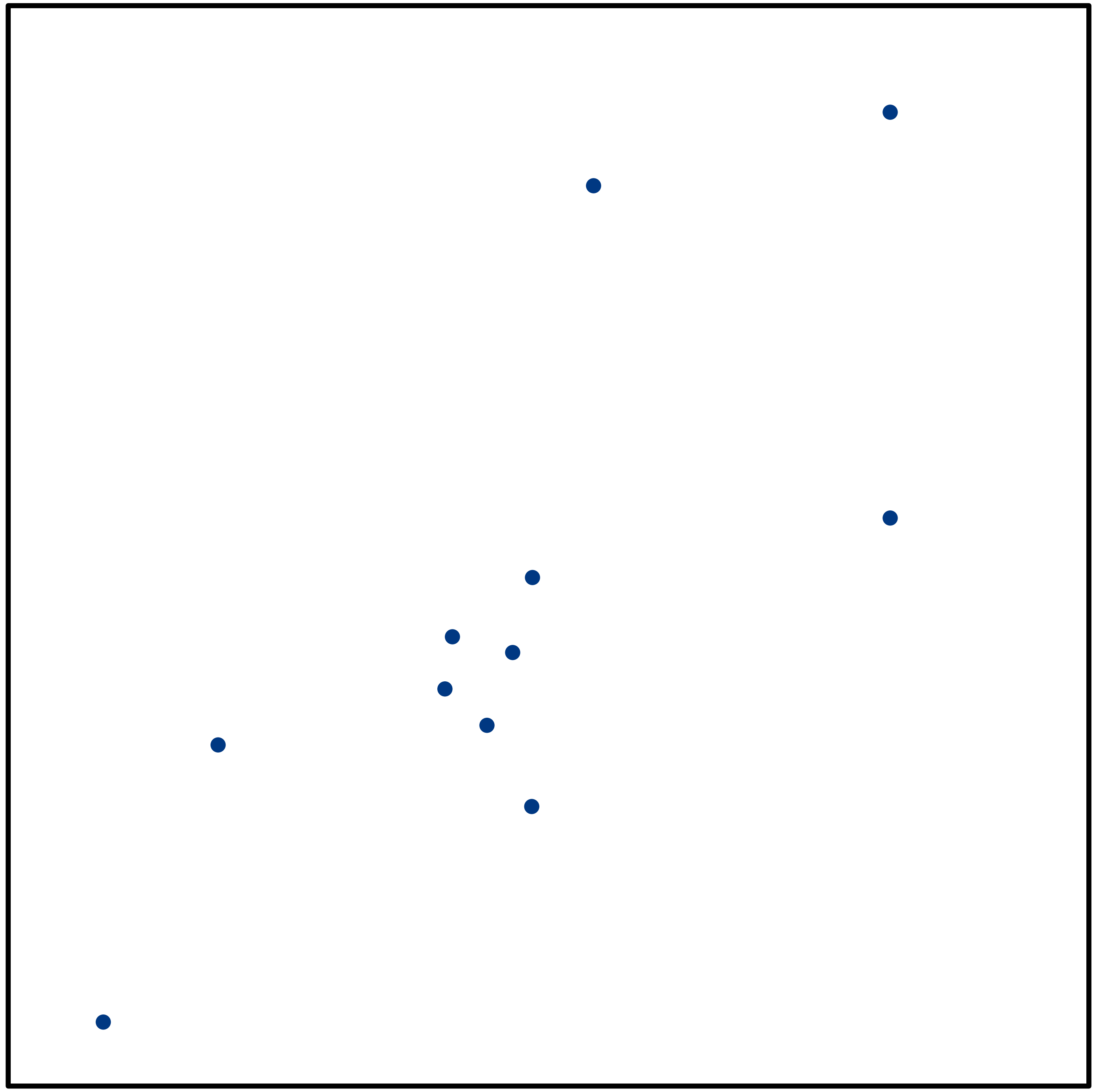
- if $|P| \leq 1$, is a leaf storing P and Q
- otherwise, is a node storing Q with four quadtrees as children in four quadrants for:

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

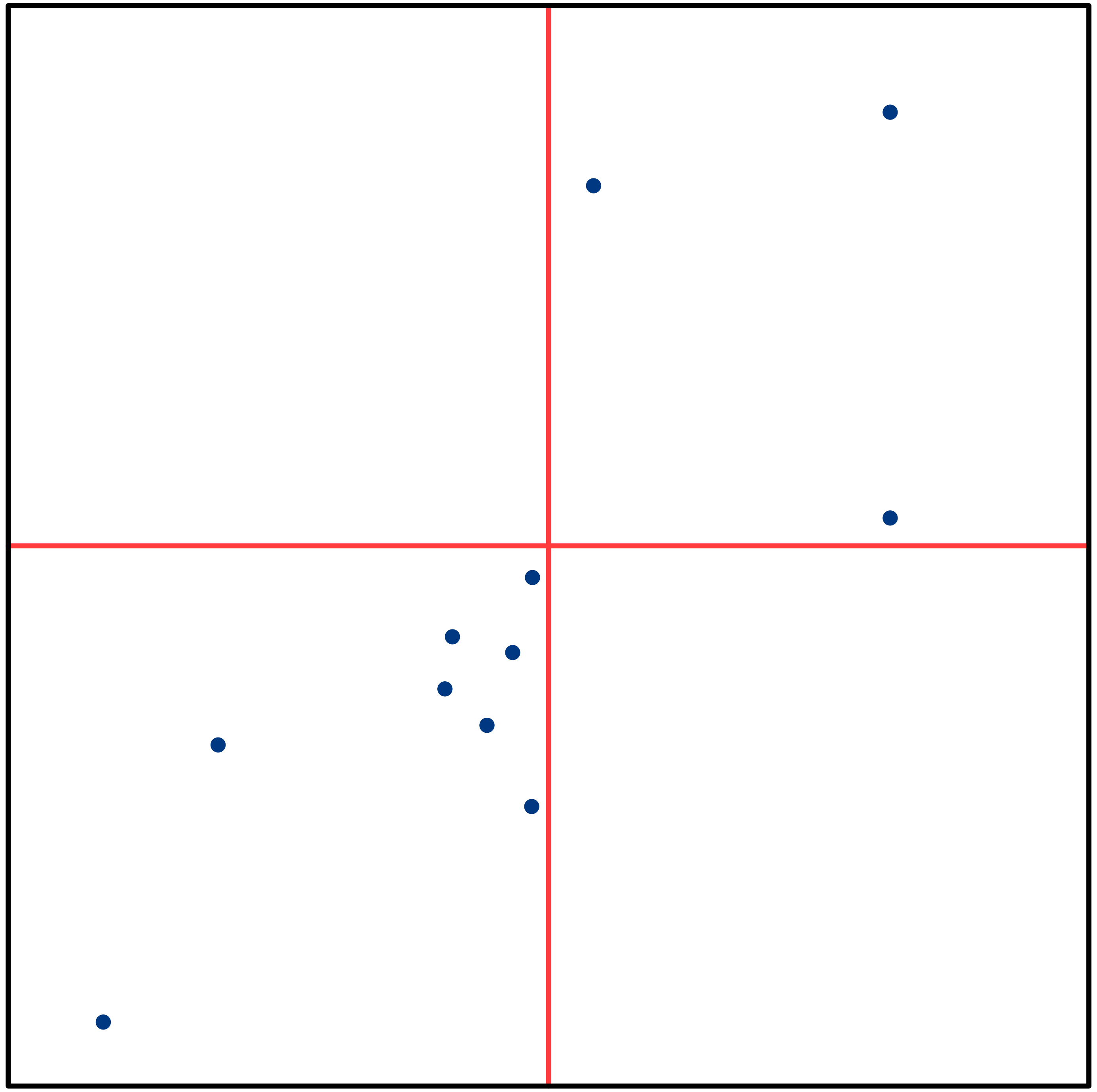
where $x_{\text{mid}} = \frac{x_Q + x'_Q}{2}$ and $y_{\text{mid}} = \frac{y_Q + y'_Q}{2}$.

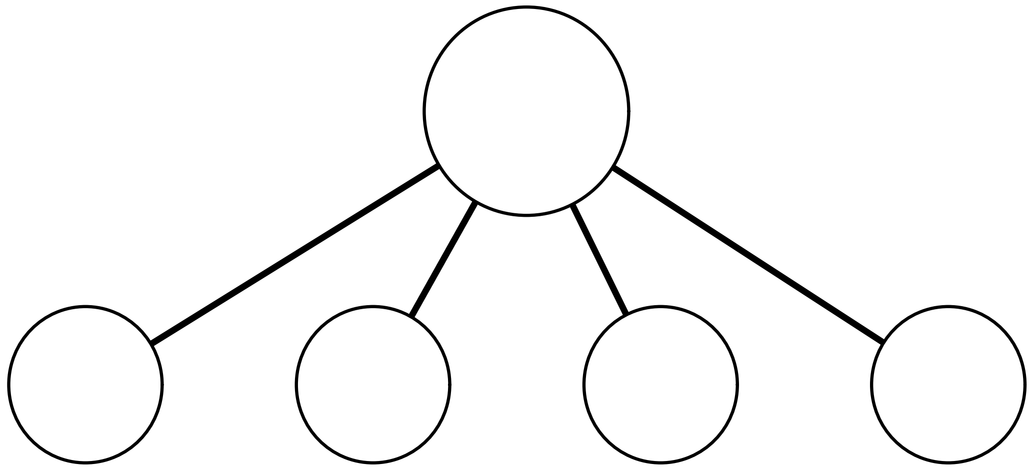


Example

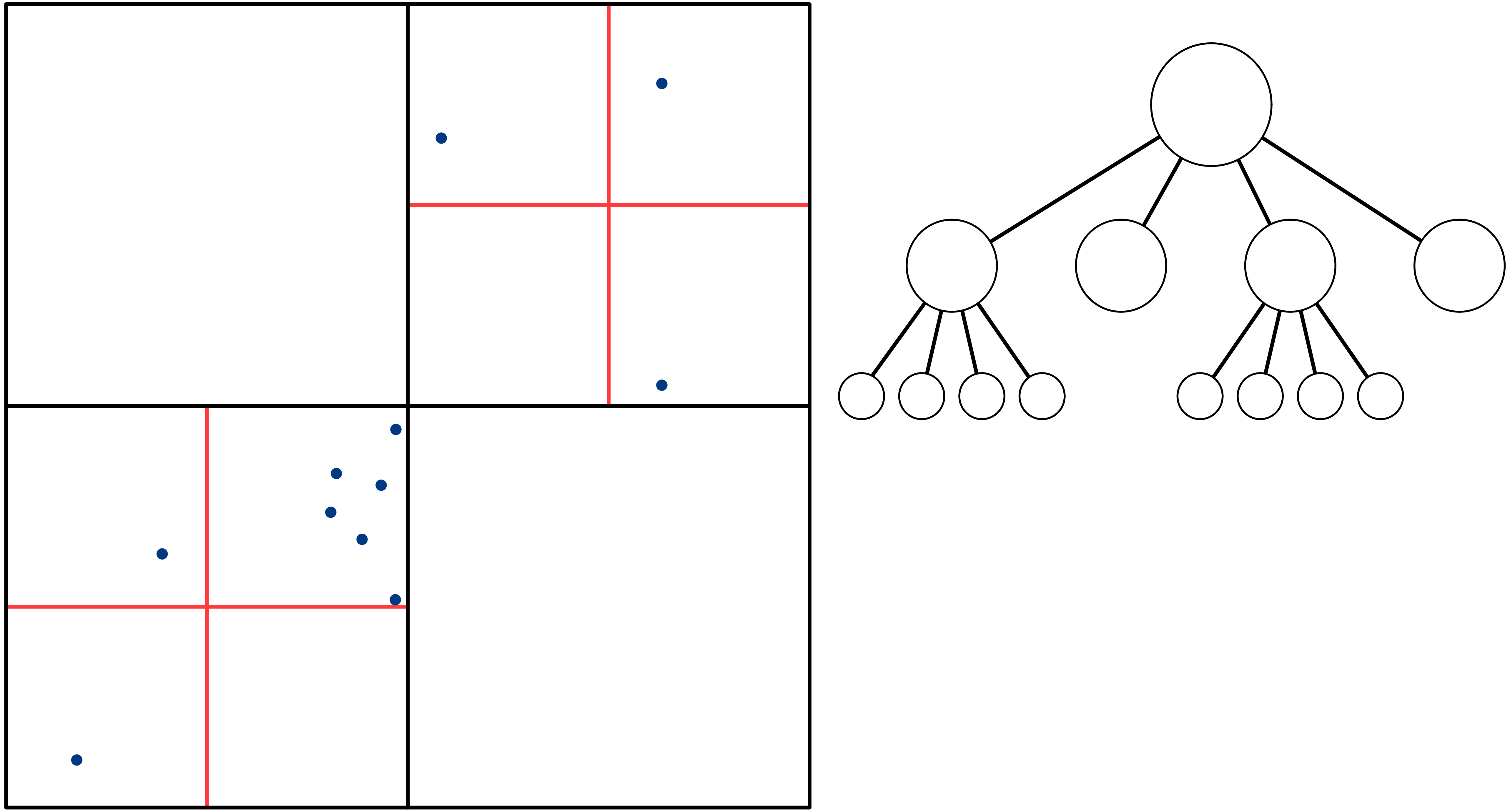


Example

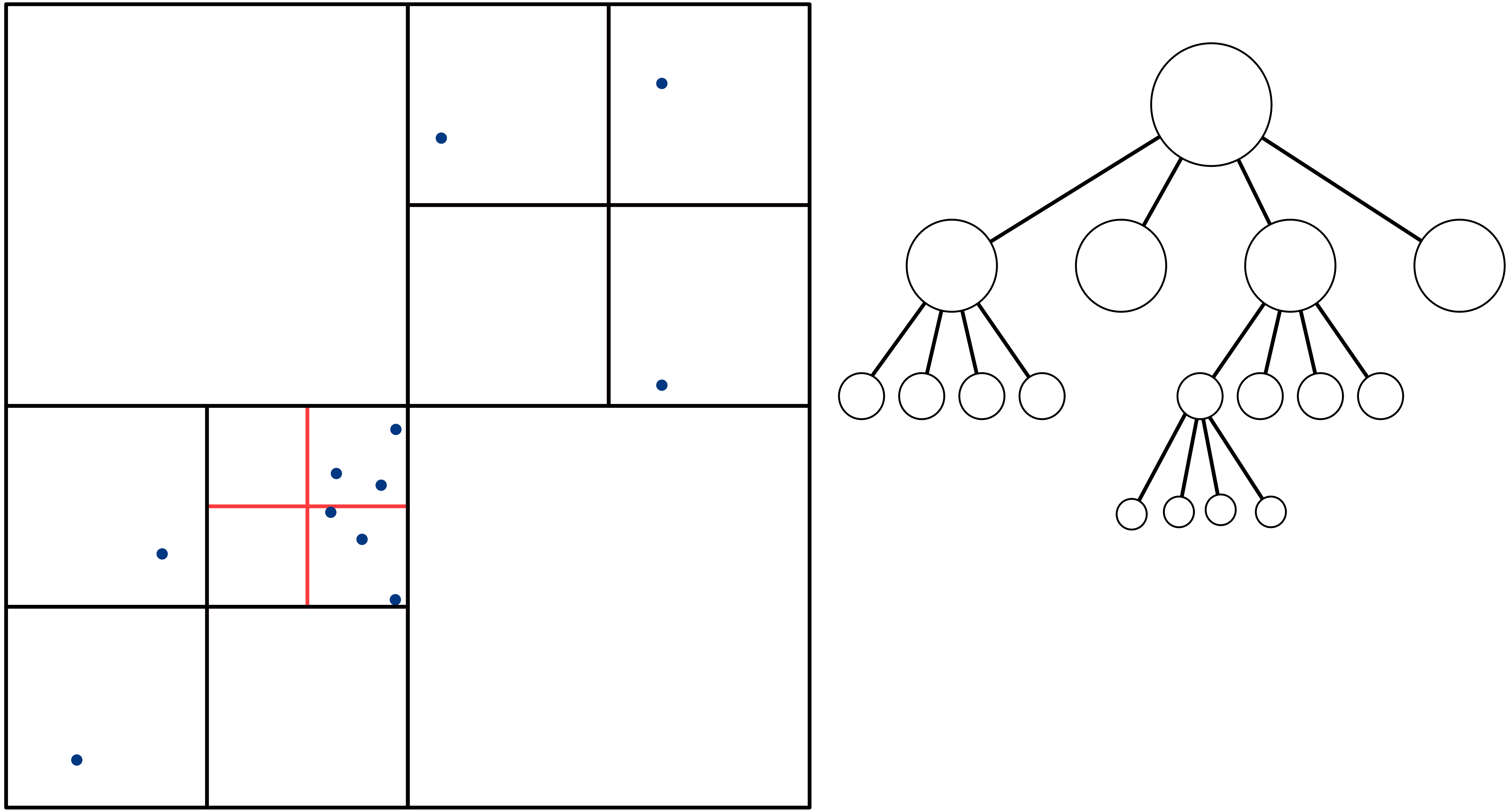




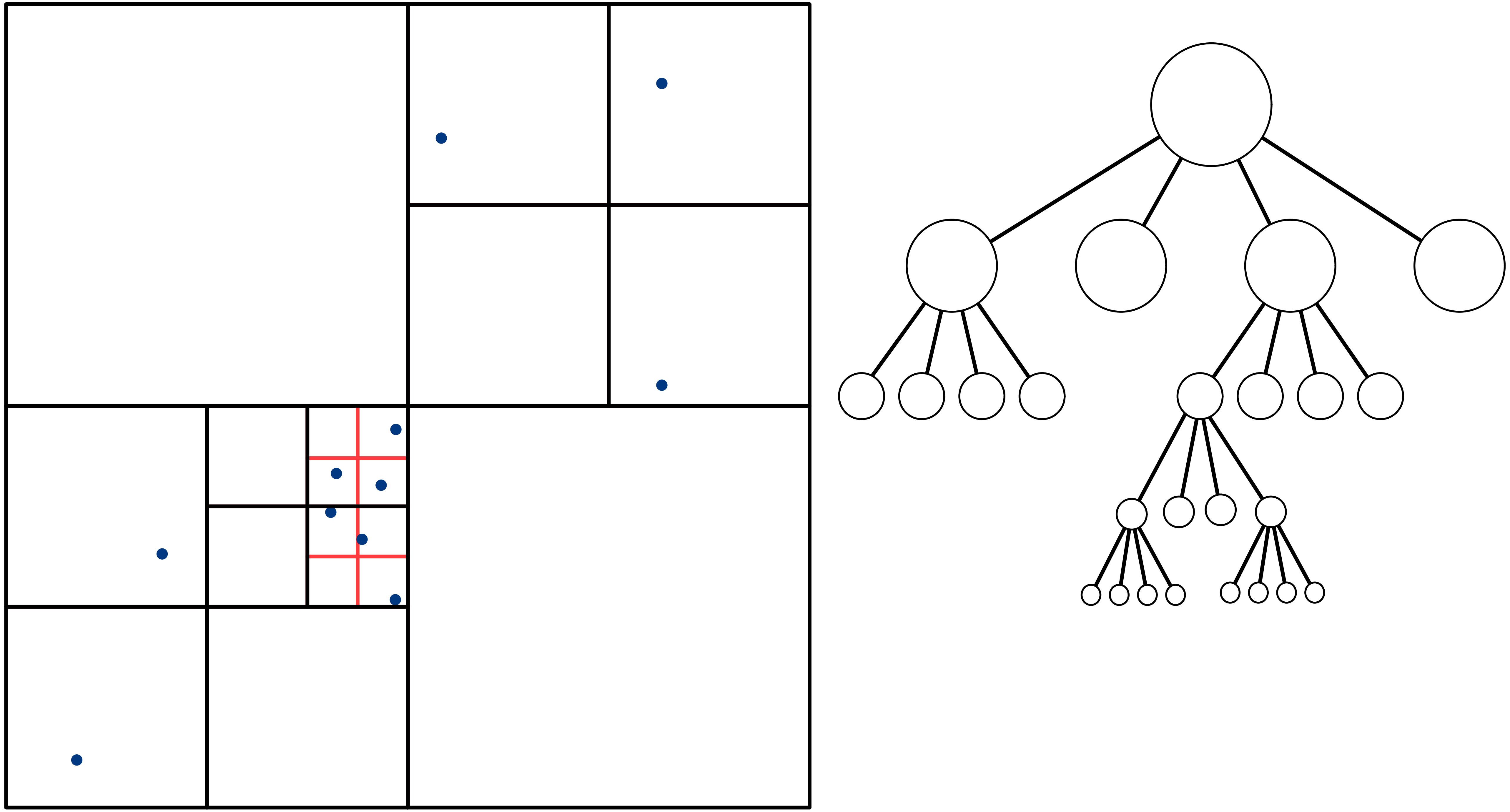
Example



Example

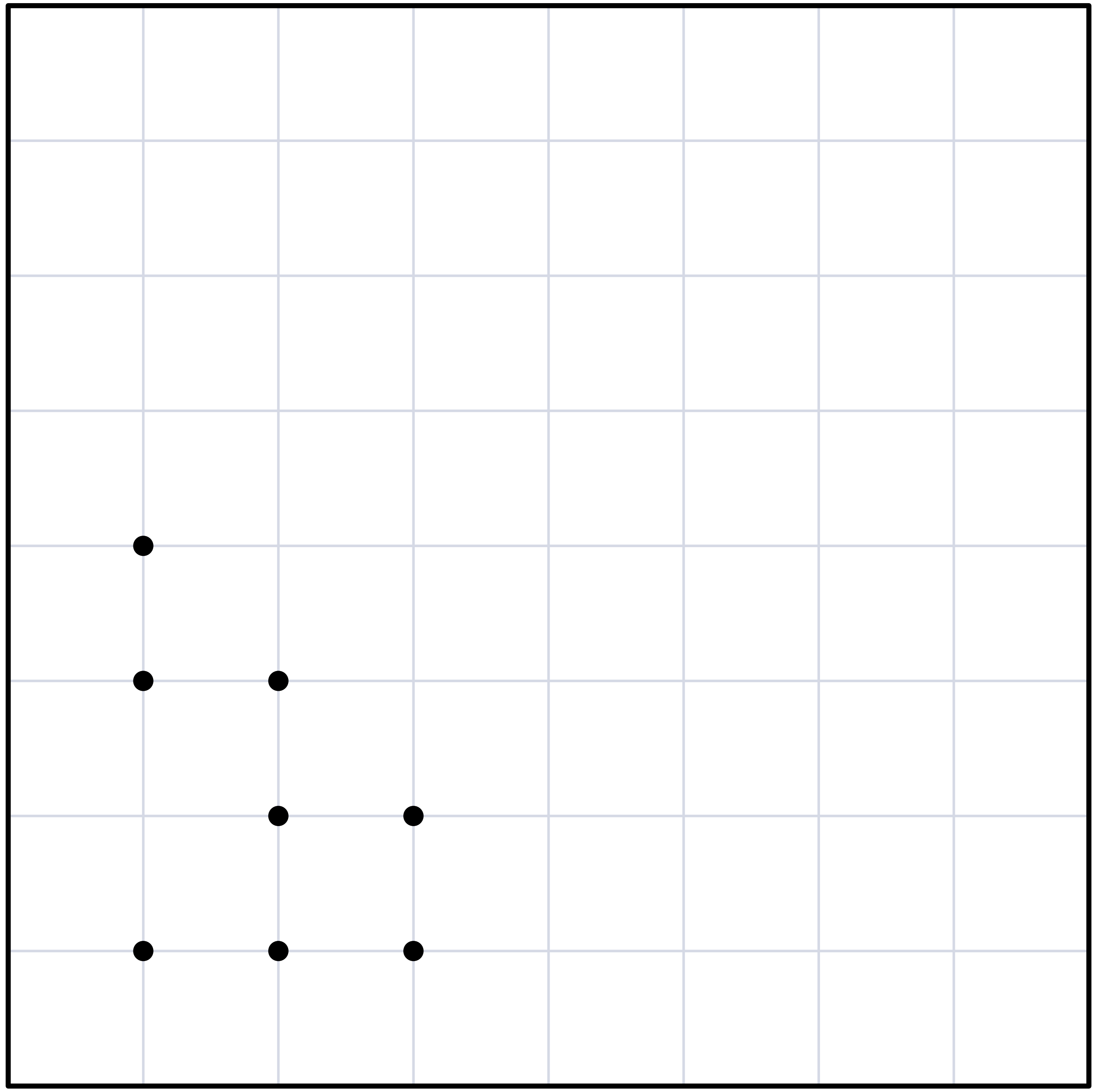


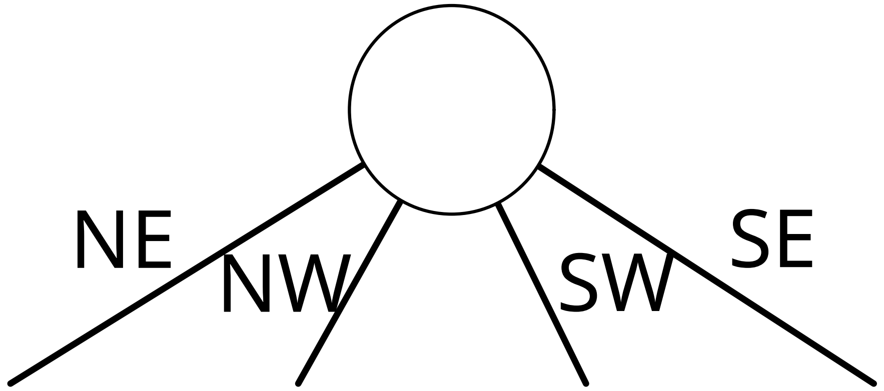
Example



Example

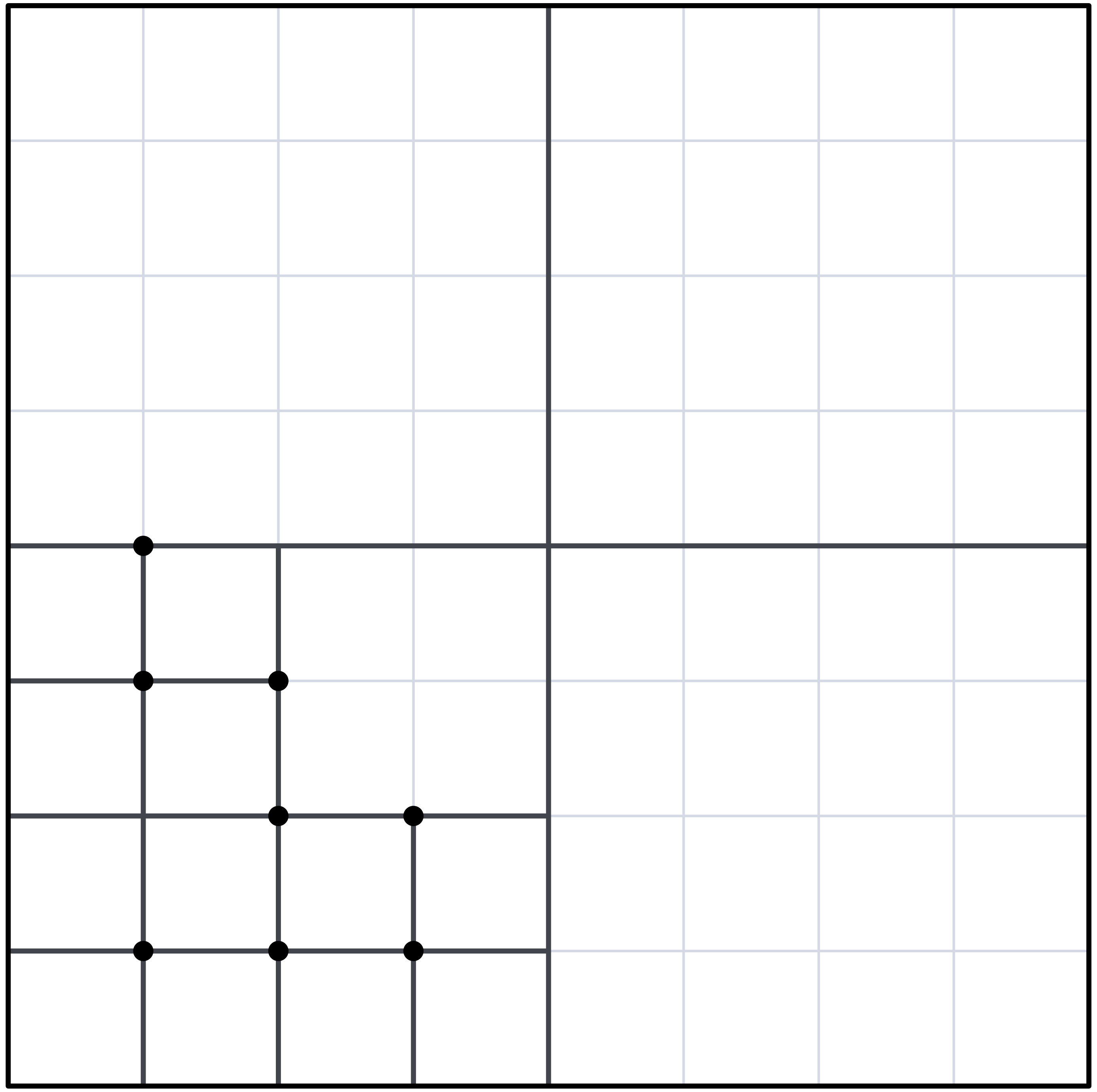
Exercise:

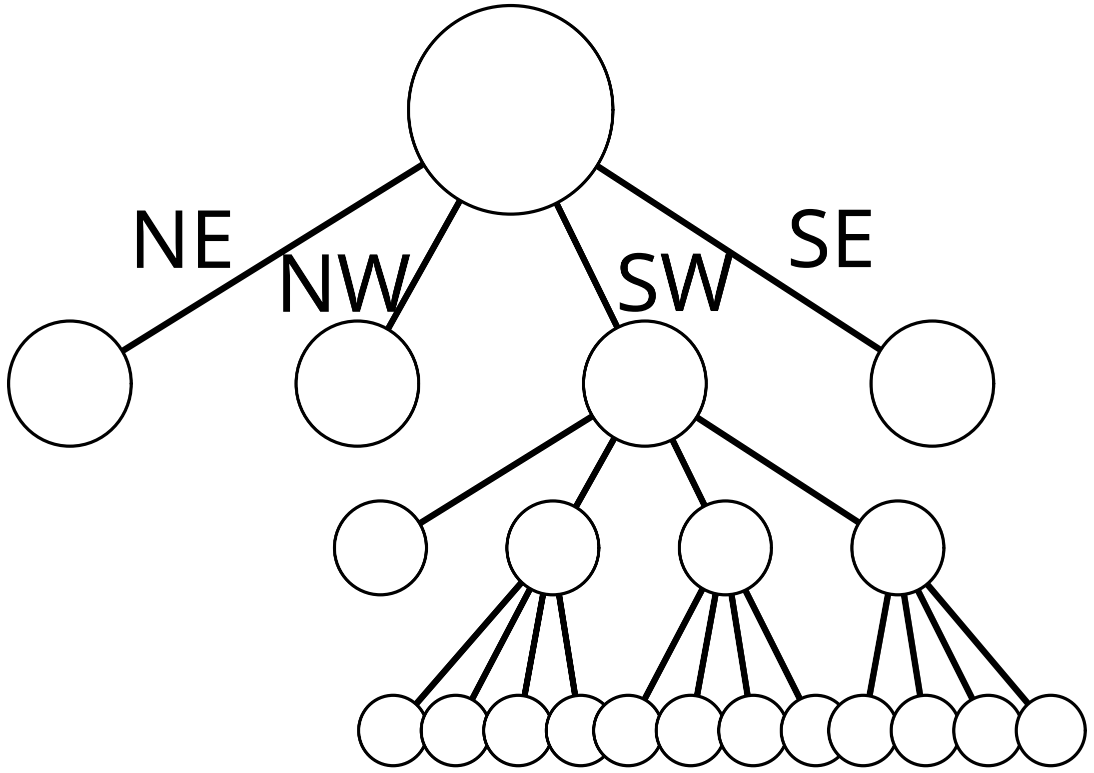




Example

Exercise:





Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Quadtree properties

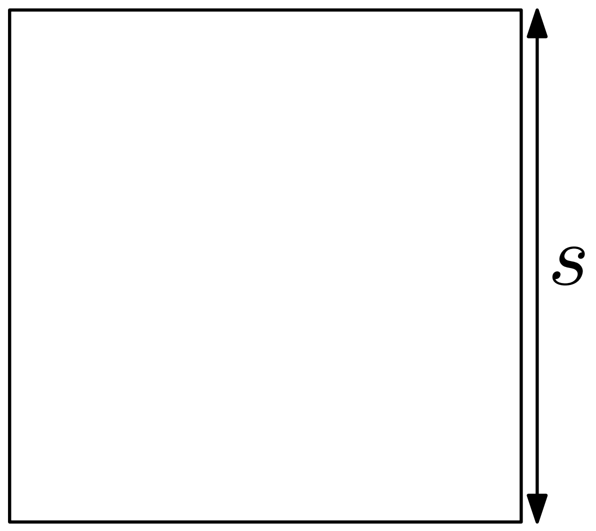
The recursive definition of a quadtrees immediately results in an algorithm

Question: What is the depth of a quadtree with n nodes?

Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.



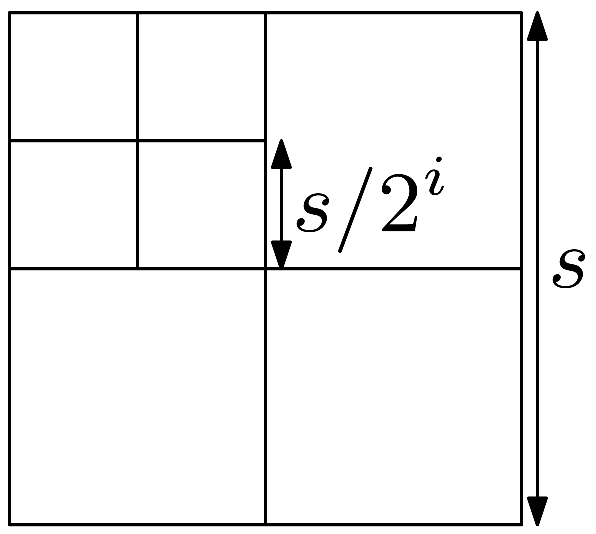
Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Proof:

- consider square σ of depth i with side length $s/2^i$



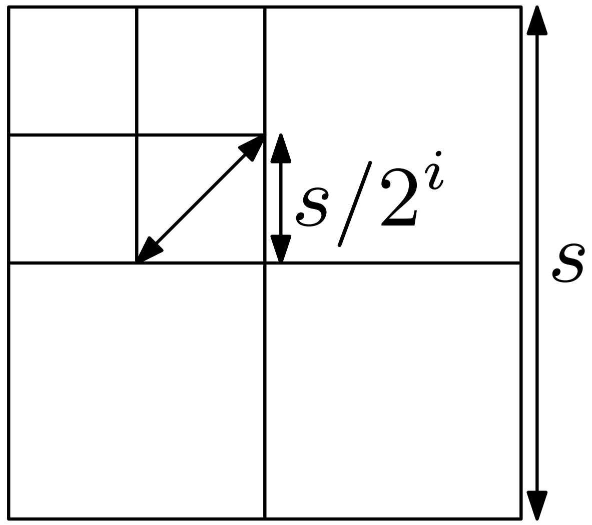
Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Proof:

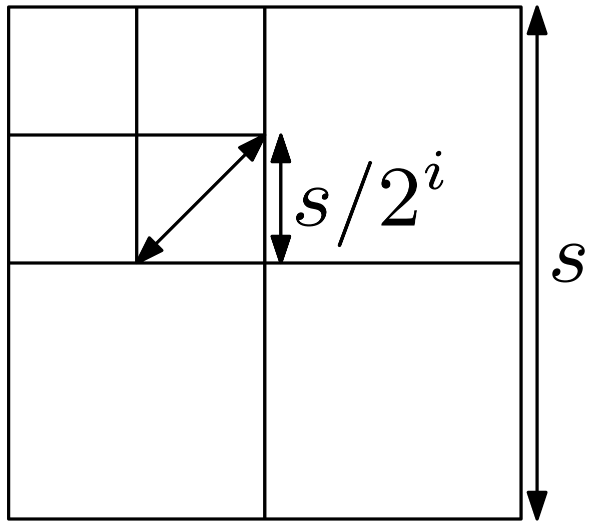
- consider square σ of depth i with side length $s/2^i$
- maximum distance between two points in σ: $\sqrt{2}s/2^i$



Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.



Proof:

- consider square σ of depth i with side length $s/2^i$
- maximum distance between two points in σ: $\sqrt{2}s/2^i$

$\Rightarrow$ if depth is i, $\sqrt{2}s/2^i \geq c$

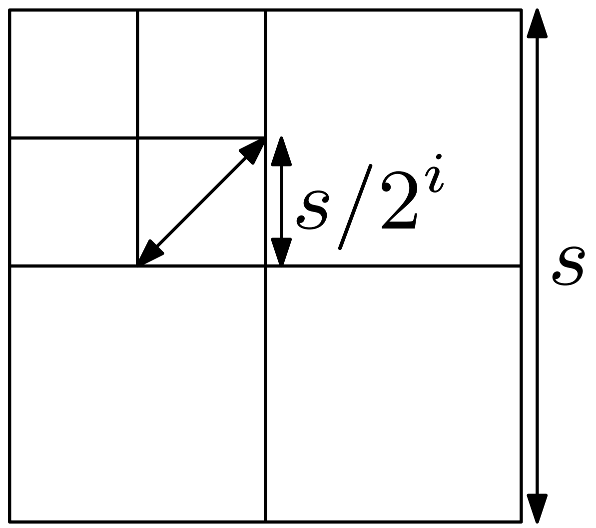
Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Proof:

- consider square σ of depth i with side length $s/2^i$
- maximum distance between two points in σ: $\sqrt{2}s/2^i$



$\Rightarrow$ if depth is i, $\sqrt{2}s/2^i \geq c$

$\Rightarrow i \leq \log(\sqrt{2}s/c) = \log(s/c) + 1/2$

Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem 1: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem 1: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Proof:

- Inner nodes have 4 children $\quad \Rightarrow \quad$ #leaves $= 1 + 3 \cdot$#inner nodes

Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem 1: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Proof:

- Inner nodes have 4 children $\quad \Rightarrow \quad$ #leaves $= 1 + 3 \cdot$#inner nodes
- Inner nodes correspond to disjoint squares with ≥ 2 points
  $\Rightarrow \quad \leq n$ squares per layer corresponding to inner nodes

Quadtree properties

The recursive definition of a quadtrees immediately results in an algorithm

Lemma 1: Let c be the smallest distance between any two points in a point set P, and let s be the side length of the initial (biggest) square. Then the depth of a quadtree for P is at most $\log(s/c) + 3/2$.

Theorem 1: A quadtree of depth d storing n points has $O((d+1)n)$ nodes and can be constructed in $O((d+1)n)$ time.

Proof:

- Inner nodes have 4 children $\quad\Rightarrow\quad$ #leaves $= 1 + 3\cdot$#inner nodes
- Inner nodes correspond to disjoint squares with ≥ 2 points
  $\Rightarrow \quad \leq n$ squares per layer corresponding to inner nodes

$\Rightarrow$ for depth d overall $O((d+1)n)$ nodes.

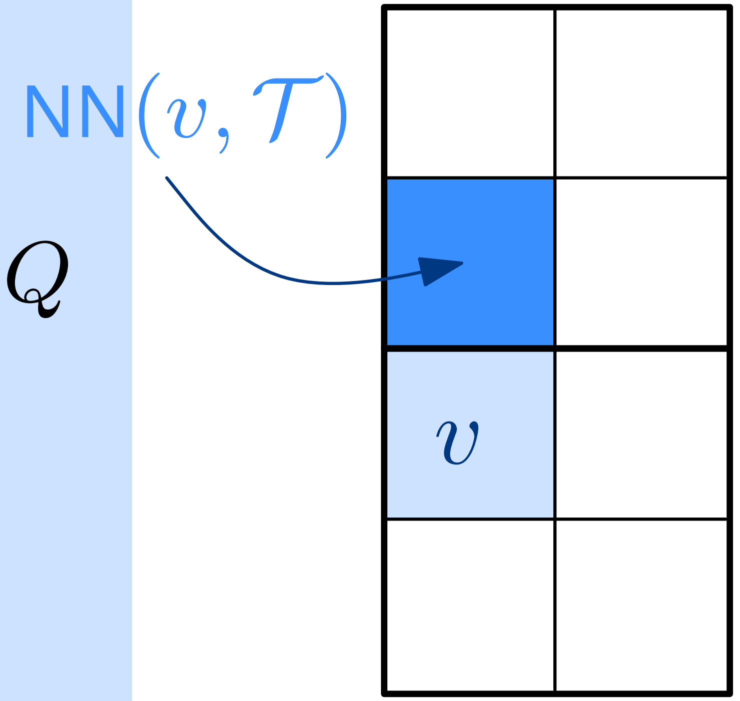
Finding neighbors

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

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

Output: deepest v' not deeper than v, with $v'.Q$ 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 is a leaf **then return** μ
if $v =$ NW(NE)-child of π **then return** SW(SE)-child of μ



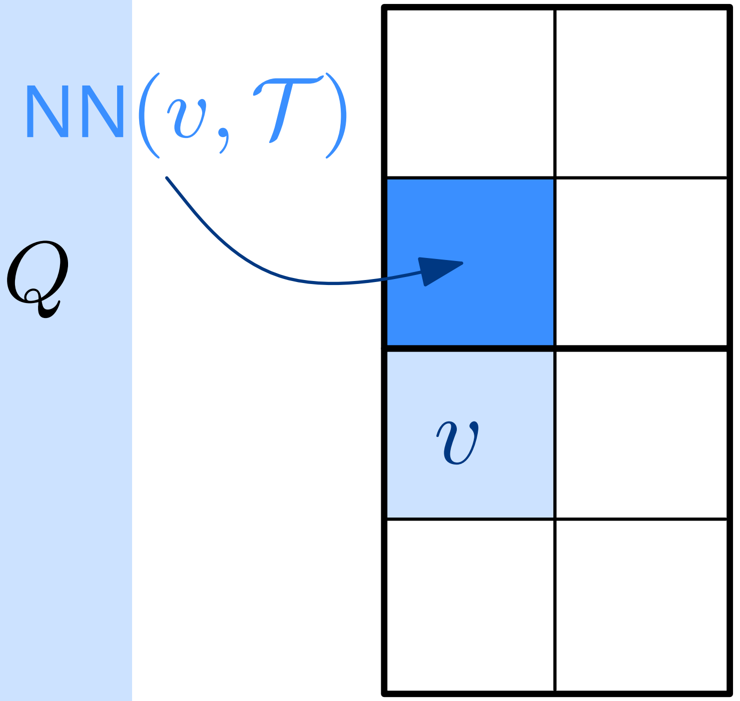
Finding neighbors

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

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

Output: deepest v' not deeper than v, with $v'.Q$ north neighbor of $v.Q$

```
if v = root(T) then return NIL
π ← parent(v)
if v = SW(SE)-child of π then return NW(NE)-child of π
μ ← NorthNeighbor(π, T)
if μ = NIL or μ is a leaf then return μ
if v = NW(NE)-child of π then return SW(SE)-child of μ
```



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

Finding neighbors

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

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

Output: deepest v' not deeper than v, with $v'.Q$ north neighbor of $v.Q$

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

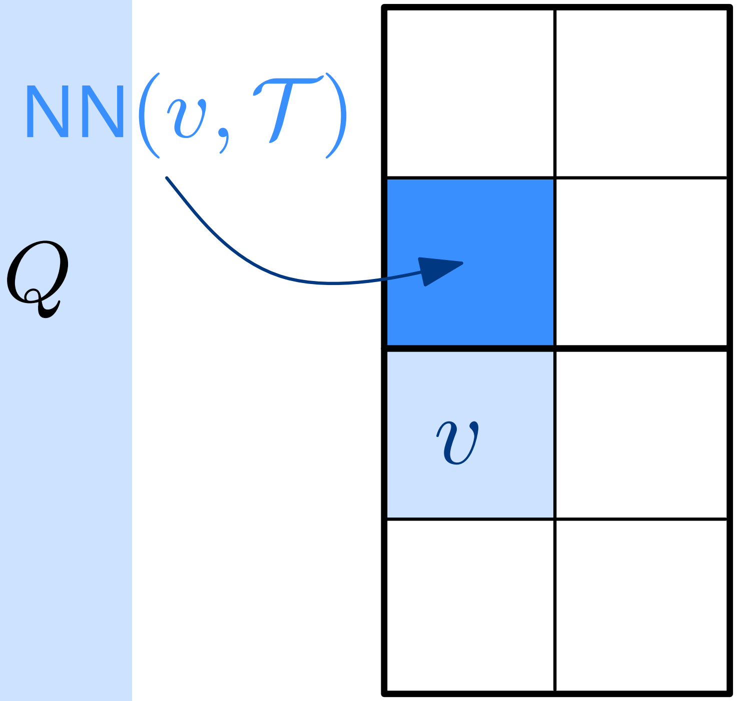
$\pi \leftarrow$ parent(v)

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 is a leaf **then return** μ

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



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

Proof:

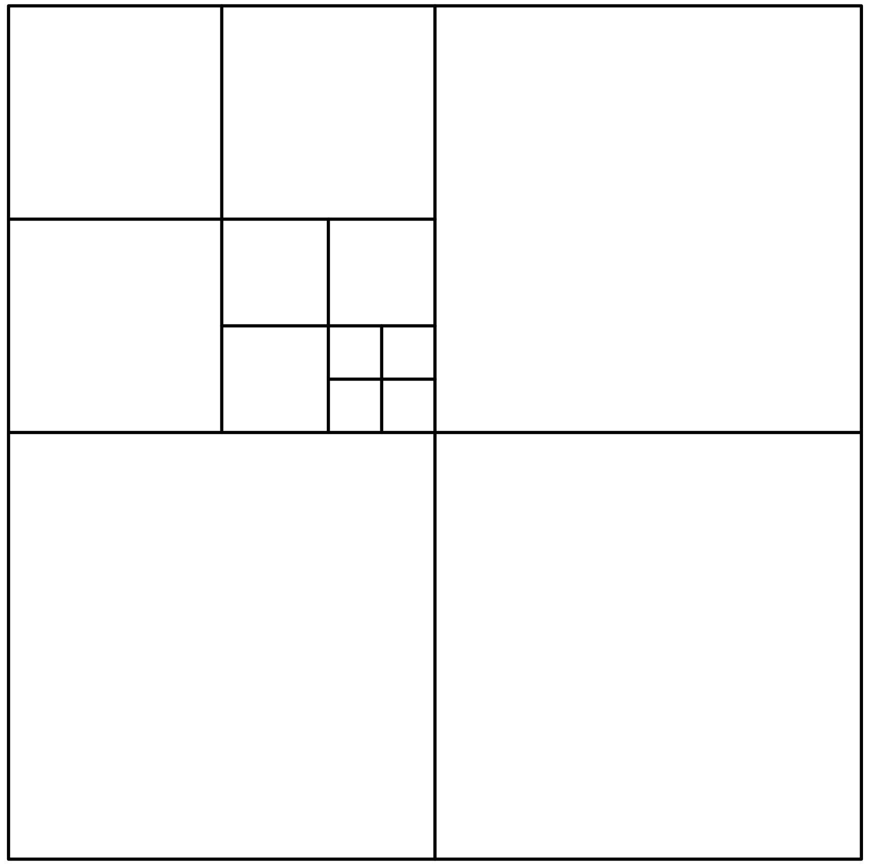
- depth of recursion is $O(d+1)$
- cost per recursive step is $O(1)$

Balanced quadtrees

Definition: a quadtree is balanced if any two neighboring nodes differ by at most 1 in depth

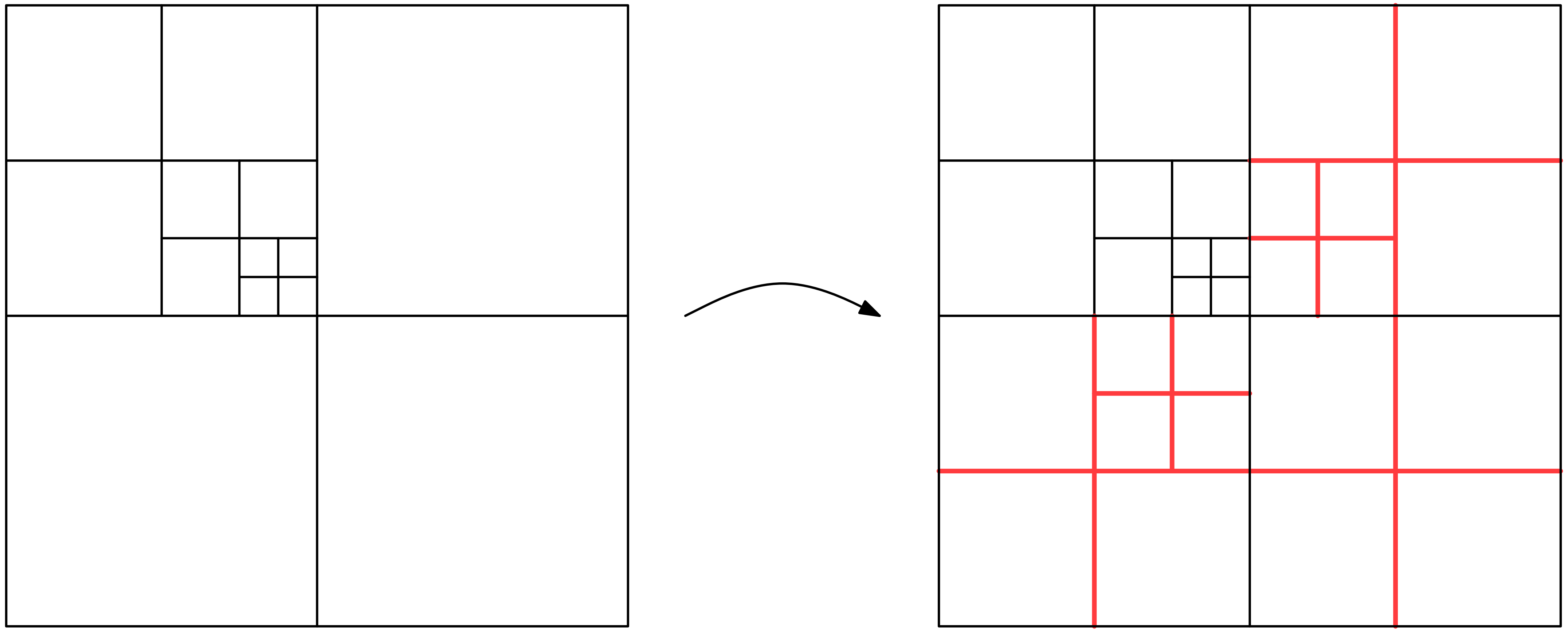
Balanced quadtrees

Definition: a quadtree is balanced if any two neighboring nodes differ by at most 1 in depth



Balanced quadtrees

Definition: a quadtree is balanced if any two neighboring nodes differ by at most 1 in depth



Balancing quadtrees

BALANCEQUADTREE($\mathcal{T}$)

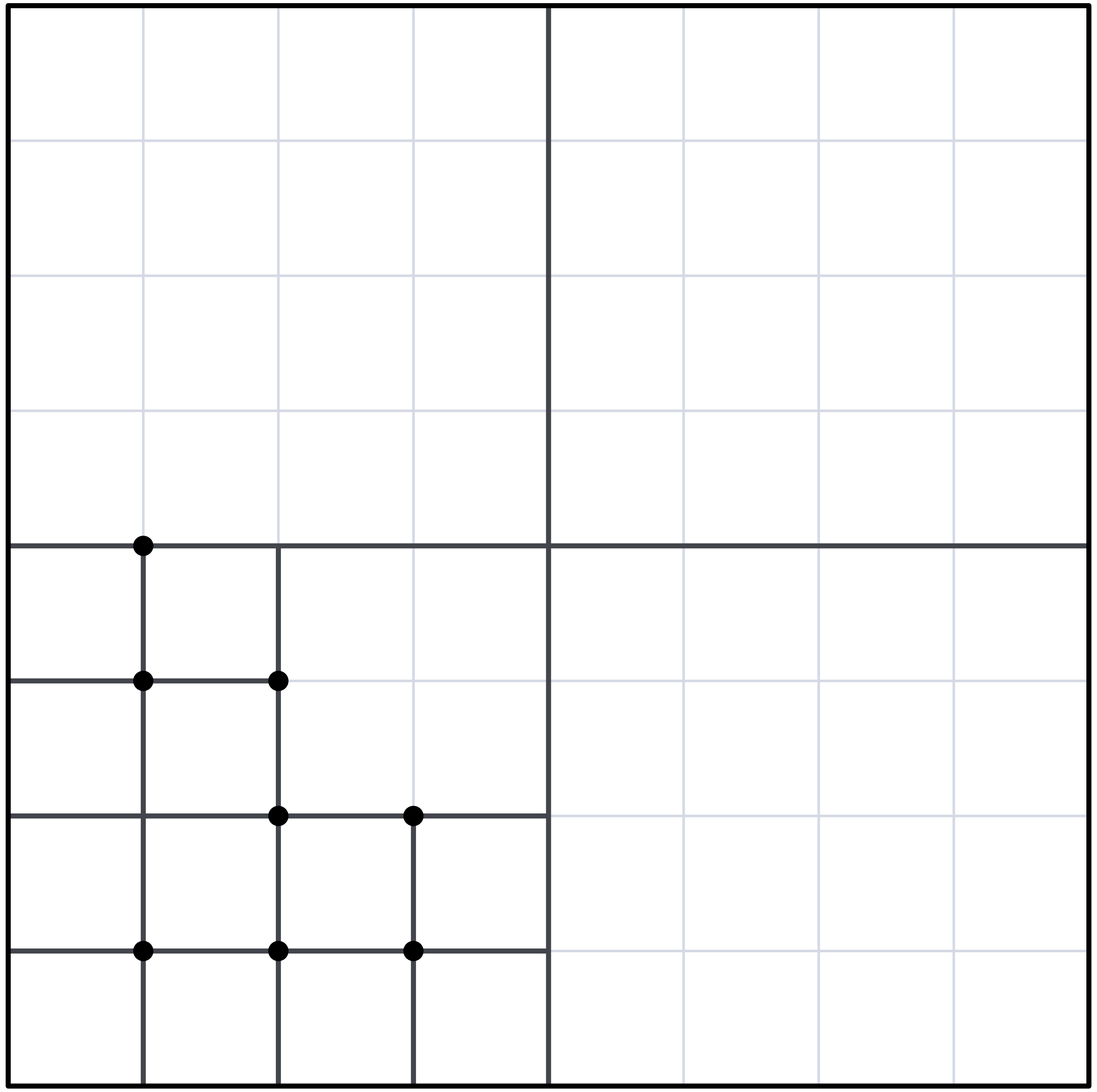
Input: Quadtree $\mathcal{T}$

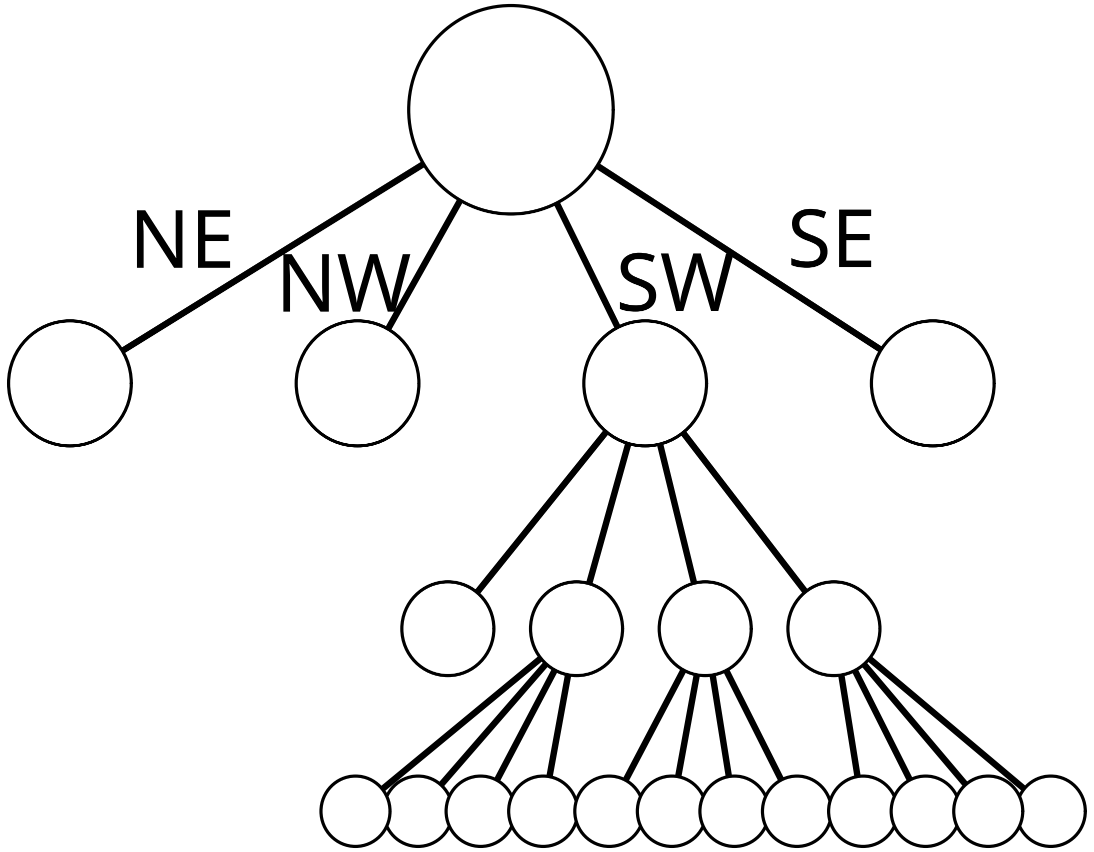
Output: Balanced quadtree $\mathcal{T}$

$L \leftarrow$ list of all leafs of $\mathcal{T}$
while L is not empty **do**
    $\mu \leftarrow$ extract a leaf from L
    if $\mu.Q$ is too large **then**
        partition $\mu.Q$ into 4 quadrants and add 4 leaves to $\mathcal{T}$
        insert new leaves into L
        if any neighbors of $\mu.Q$ are too large **then**
            insert them into L
return $\mathcal{T}$

Balancing quadtrees

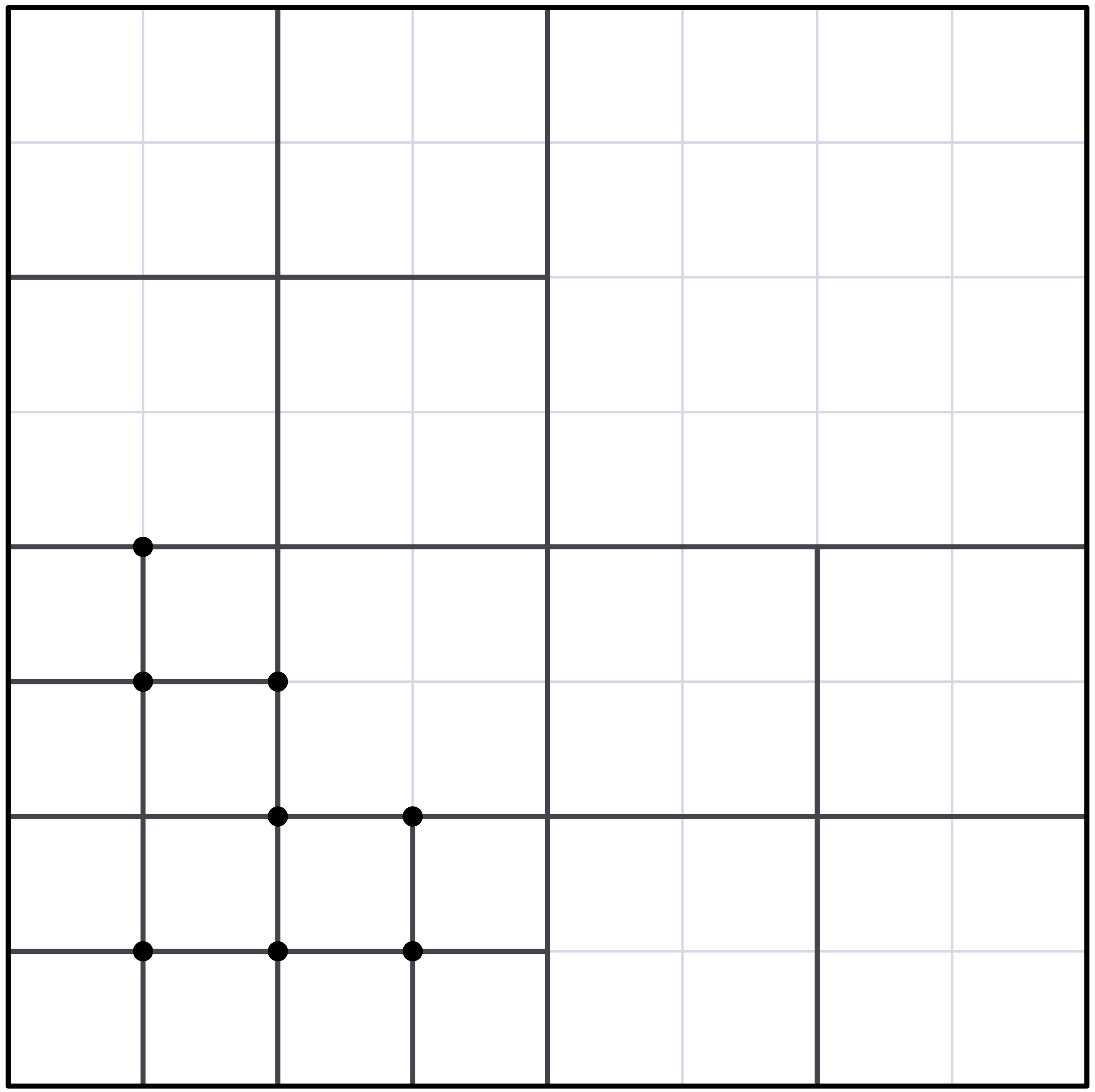
Exercise:

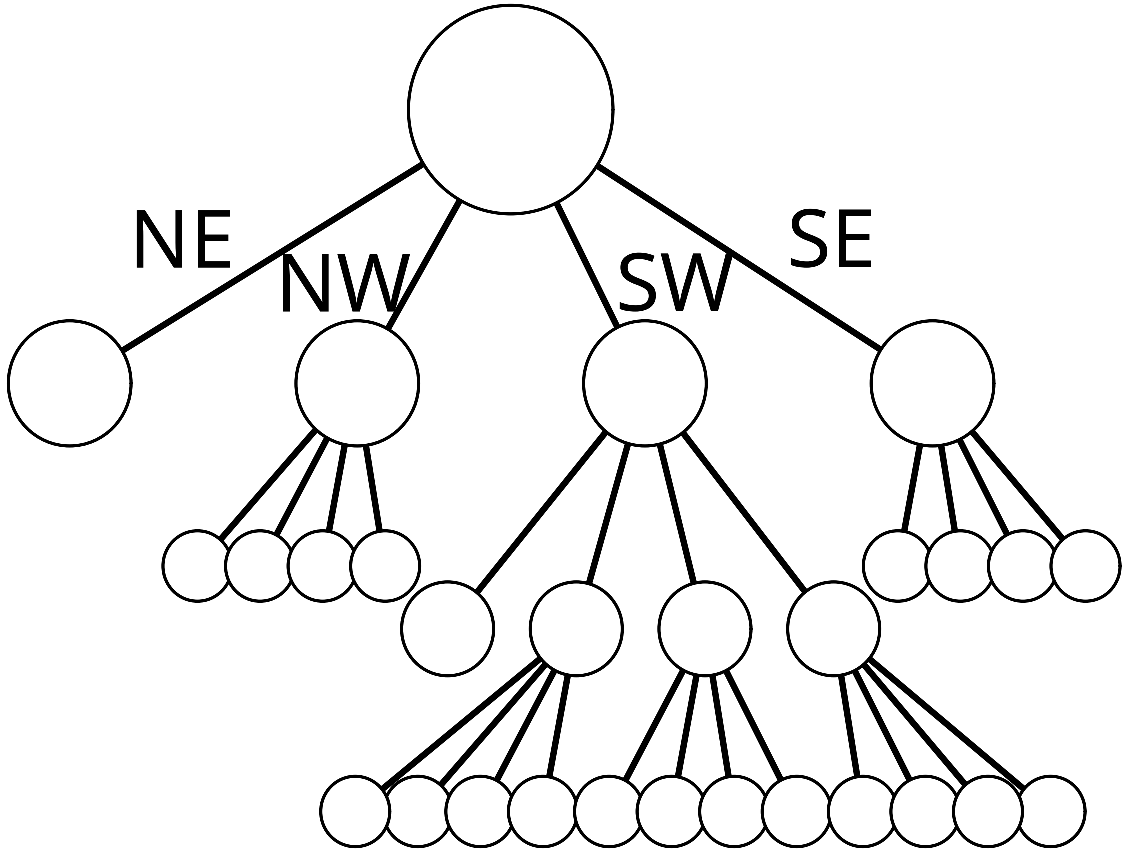




Balancing quadtrees

Exercise:





Balancing quadtrees

BALANCEQUADTREE($\mathcal{T}$)

Input: Quadtree $\mathcal{T}$

Output: Balanced quadtree $\mathcal{T}$

$L \leftarrow$ list of all leafs of $\mathcal{T}$
while L is not empty **do**
    $\mu \leftarrow$ extract a leaf from L
    if $\mu.Q$ is too large **then**
        partition $\mu.Q$ into 4 quadrants and add 4 leaves to $\mathcal{T}$
        insert new leaves into L
        if any neighbors of $\mu.Q$ are too large **then**
            insert them into L
return $\mathcal{T}$

Balancing quadtrees

BALANCEQUADTREE($\mathcal{T}$)

Input: Quadtree $\mathcal{T}$

Output: Balanced quadtree $\mathcal{T}$

$L \leftarrow$ list of all leafs of $\mathcal{T}$

while L is not empty **do**

    $\mu \leftarrow$ extract a leaf from L How?

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

        partition $\mu.Q$ into 4 quadrants and add 4 leaves to $\mathcal{T}$

        insert new leaves into L

        if any neighbors of $\mu.Q$ are too large **then**

            insert them into L

return $\mathcal{T}$

How?

Balancing quadtrees

BalanceQuadtree$(\mathcal{T})$

Input: Quadtree $\mathcal{T}$

Output: Balanced quadtree $\mathcal{T}$

```
L ← list of all leafs of T
while L is not empty do
    μ ← extract a leaf from L          How?
    if μ.Q is too large then
        partition μ.Q into 4 quadrants and add 4 leaves to T
        insert new leaves into L
        if any neighbors of μ.Q are too large then
            insert them into L
return T                               How?
```

Question: How large can a balanced quadtree get?

Balancing quadtrees

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

Balancing quadtrees

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

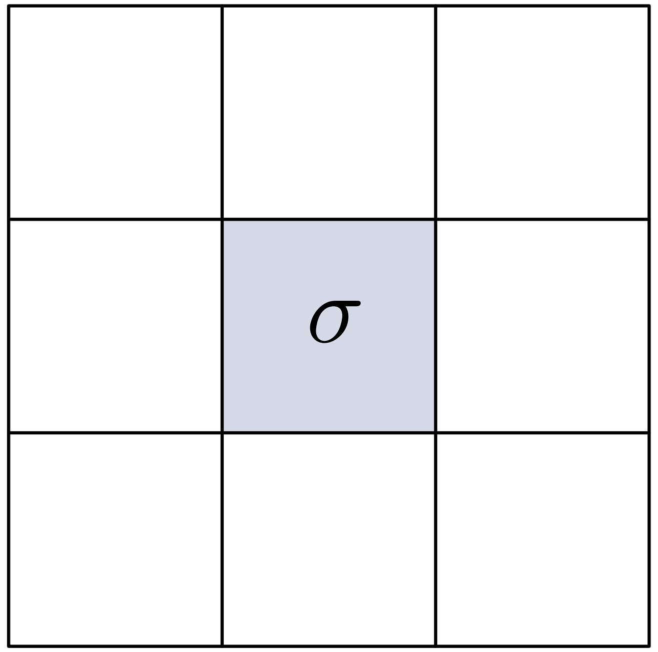
Proof: At most $8m$ splits occur. Why?

Balancing quadtrees

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

Proof: At most $8m$ splits occur. Why?

Idea: “old” vs “new” squares



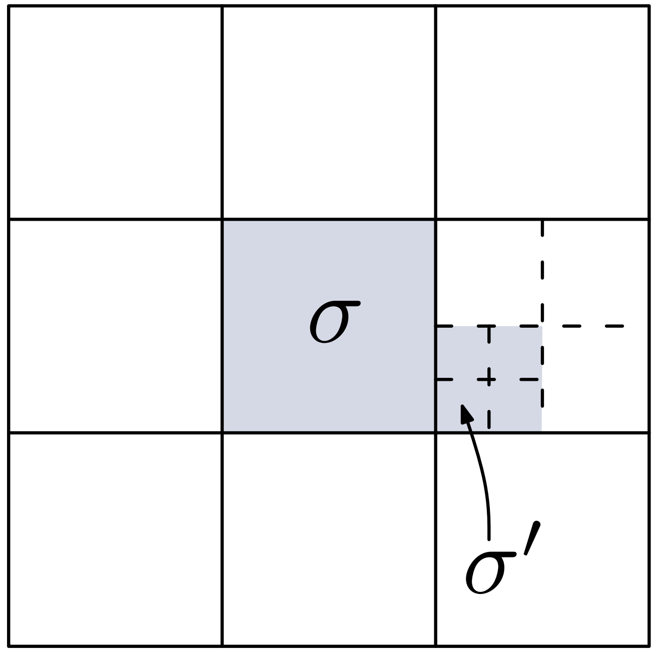
Balancing quadtrees

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

Proof: At most $8m$ splits occur. Why?

Idea: “old” vs “new” squares

- assume square σ is split $\Rightarrow$ σ has “old” nbr (among 8 nbrs)
  - suppose not $\rightarrow$ consider smallest σ violating the claim
  - let σ' be the reason for splitting σ
  - then σ' contradicts the minimality of σ



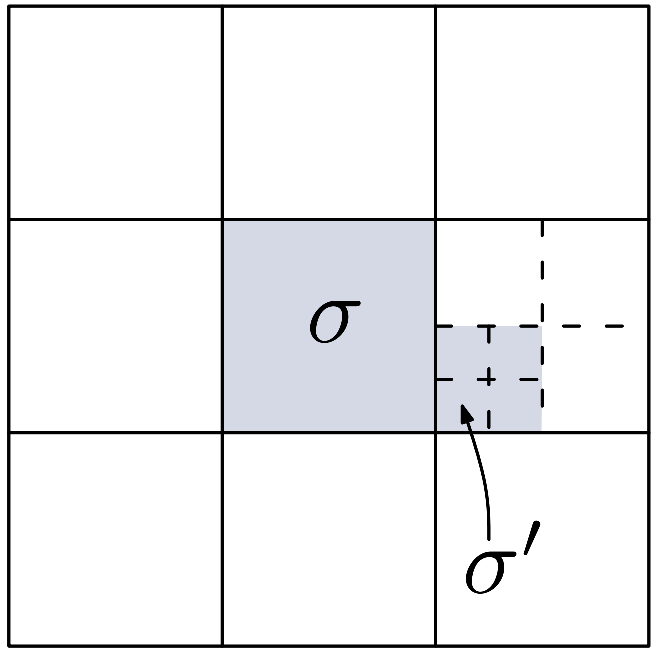
Balancing quadtrees

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

Proof: At most $8m$ splits occur. Why?

Idea: “old” vs “new” squares

- assume square σ is split $\Rightarrow$ σ has “old” nbr (among 8 nbrs)
  - suppose not $\rightarrow$ consider smallest σ violating the claim
  - let σ' be the reason for splitting σ
  - then σ' contradicts the minimality of σ
- charge the split to the “old” nbr $\rightarrow$ at most 8 per “old” nbr

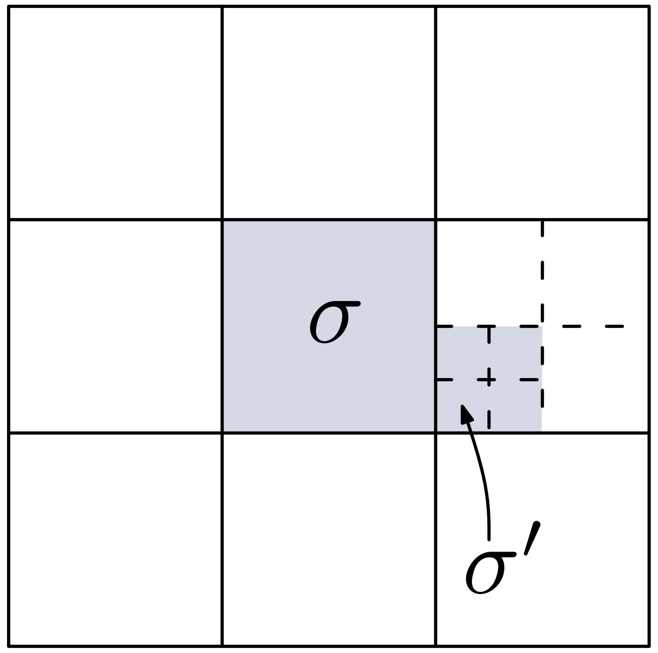


Balancing quadtrees

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

Proof: At most $8m$ splits occur. Why?

Idea: "old" vs "new" squares

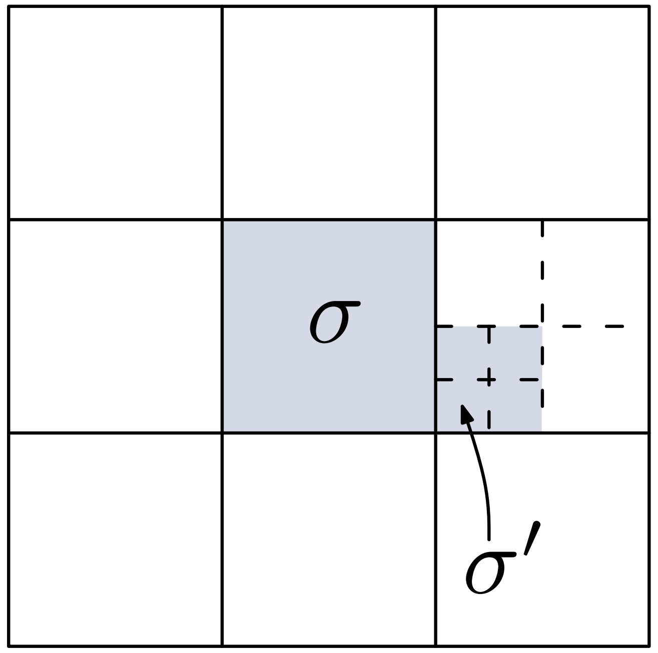


- assume square σ is split $\Rightarrow$ σ has "old" nbr (among 8 nbrs)
  - suppose not $\rightarrow$ consider smallest σ violating the claim
  - let σ' be the reason for splitting σ
  - then σ' contradicts the minimality of σ
- charge the split to the "old" nbr $\rightarrow$ at most 8 per "old" nbr
- each split needs $O(1)$ searches for nbrs $\rightarrow$ $O(d+1)$ time per split

Balancing quadtrees

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

Proof: At most $8m$ splits occur. Why?

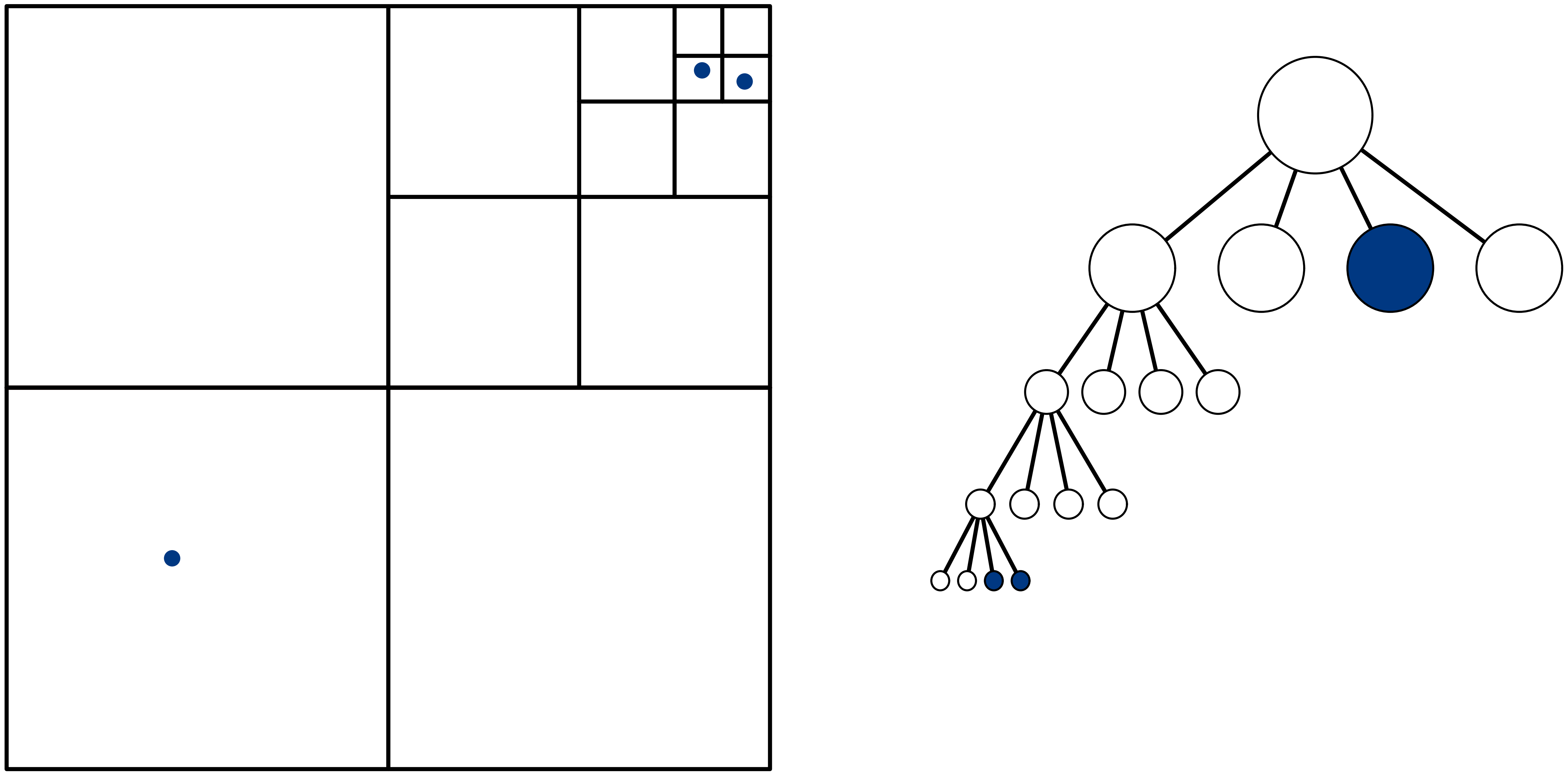


Idea: "old" vs "new" squares

- assume square σ is split $\Rightarrow$ σ has "old" nbr (among 8 nbrs)
  - suppose not $\rightarrow$ consider smallest σ violating the claim
  - let σ' be the reason for splitting σ
  - then σ' contradicts the minimality of σ
- charge the split to the "old" nbr $\rightarrow$ at most 8 per "old" nbr
- each split needs $O(1)$ searches for nbrs $\rightarrow$ $O(d+1)$ time per split

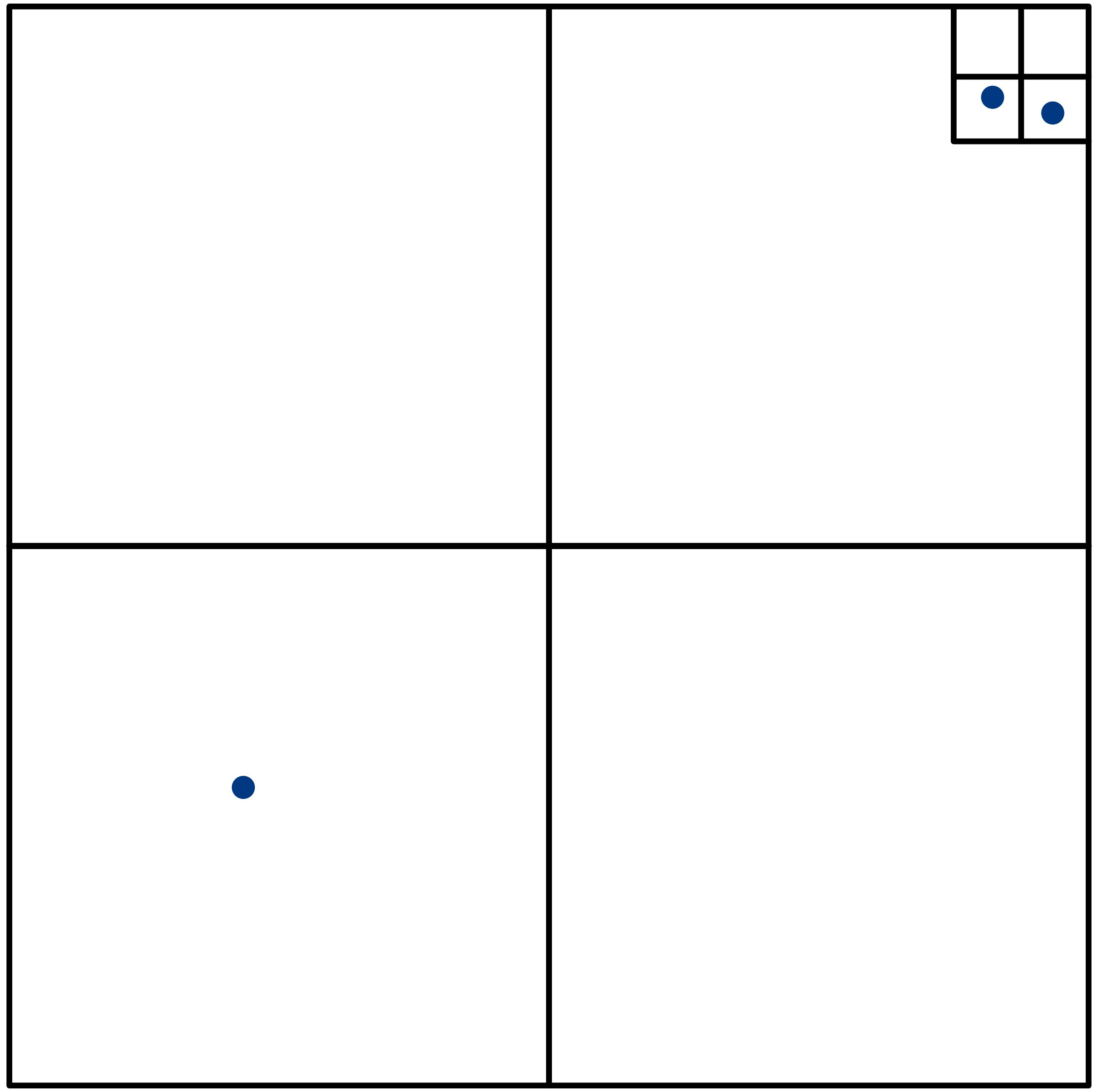
$\Rightarrow$ $O(m)$ nodes and $O((d+1)m)$ time

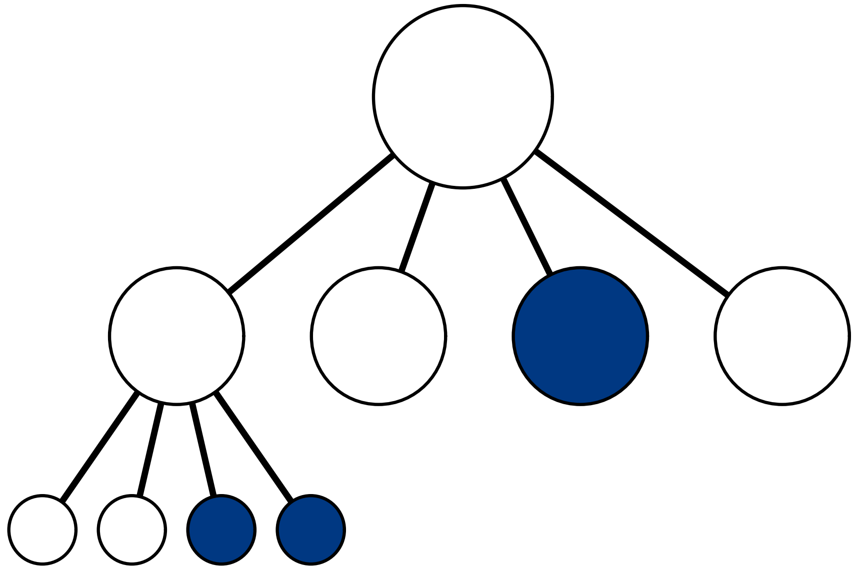
Compressed quadtrees



Paths of nodes with only one non-empty child can be compressed to an edge

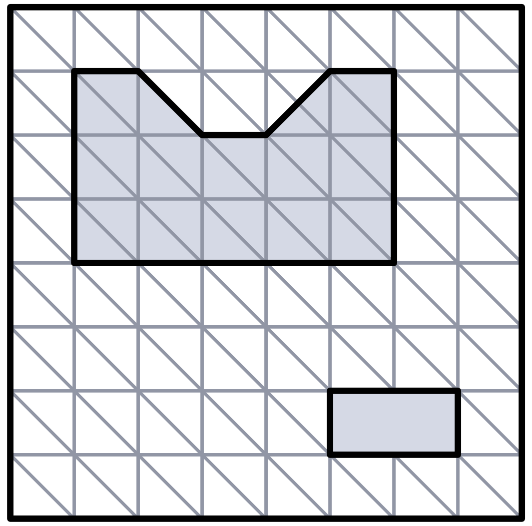
Compressed quadtrees





Paths of nodes with only one non-empty child can be compressed to an edge $\quad \rightarrow$ size $O(n)$

Quadtrees and non-uniform meshes

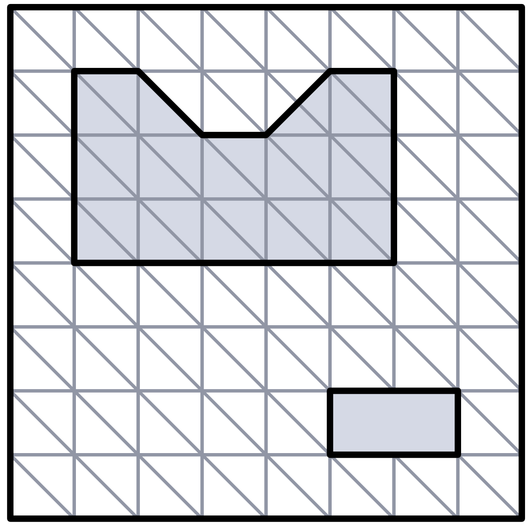


Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two

Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges
- well-shaped: angles between $45°$ and $90°$
- non-uniform: fine near boundaries, coarse otherwise

Quadtrees and non-uniform meshes



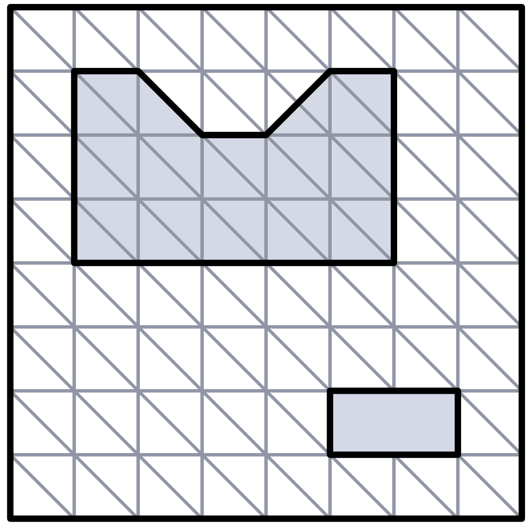
Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two

Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges
- well-shaped: angles between $45°$ and $90°$
- non-uniform: fine near boundaries, coarse otherwise

Idea: use quadtree as a base!

Quadtrees and non-uniform meshes



Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two

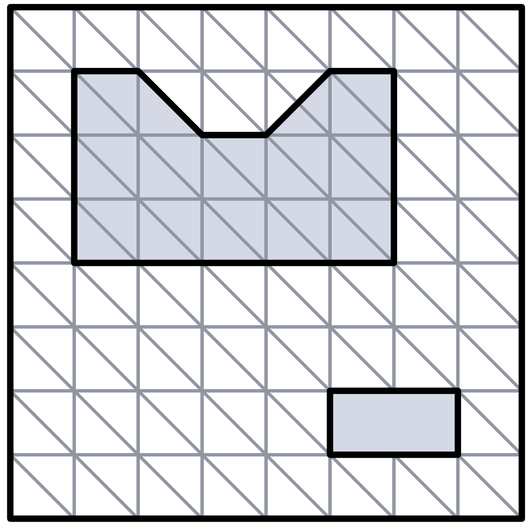
Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges
- well-shaped: angles between $45°$ and $90°$
- non-uniform: fine near boundaries, coarse otherwise

Idea: use quadtree as a base!

Question: What do we need to adapt?

Quadtrees and non-uniform meshes



Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two

Goal: triangular mesh of Q with the following properties

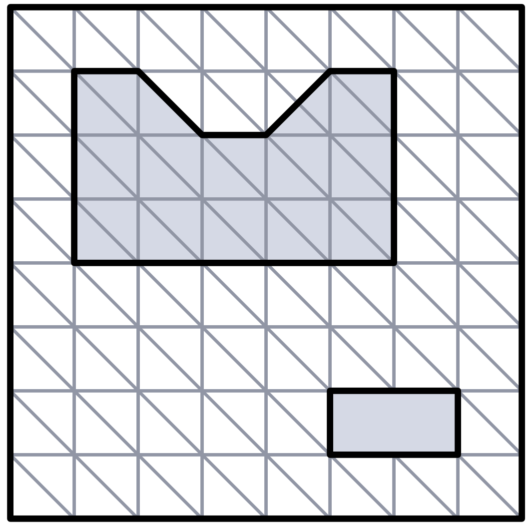
- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges
- well-shaped: angles between $45°$ and $90°$
- non-uniform: fine near boundaries, coarse otherwise

Idea: use quadtree as a base!

Question: What do we need to adapt?

Adaptation: partition squares until they no longer intersect a polygon or size is 1

Quadtrees and non-uniform meshes



Given: octilinear polygons with integer coordinates within a square $Q = [0, U] \times [0, U]$ with $U = 2^j$ a power of two

Goal: triangular mesh of Q with the following properties

- conforming: exactly one triangle on each side of interior edges
- respect input: edges of input must be part of union of mesh edges
- w … and 90°
- n … es, coarse otherwise

Idea …

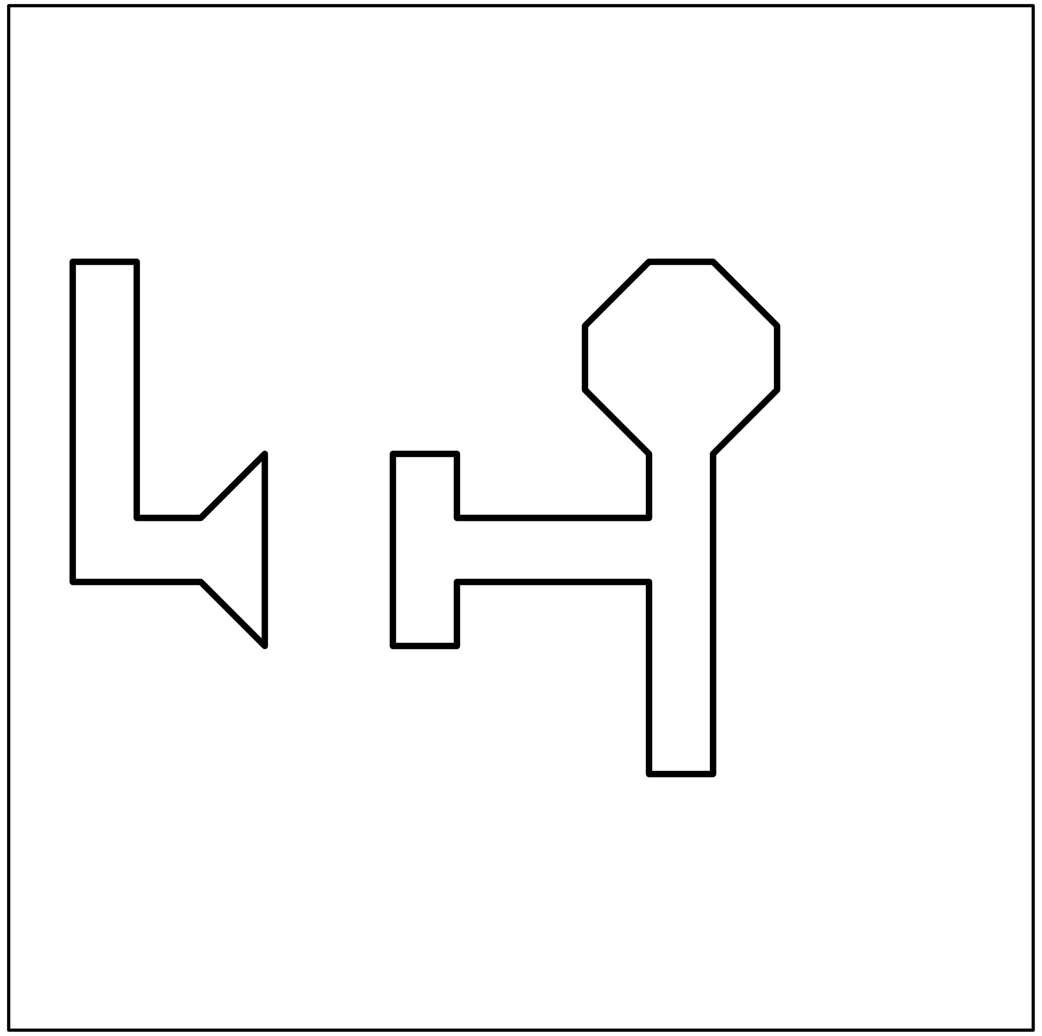
Ques … t?

intersections also include:

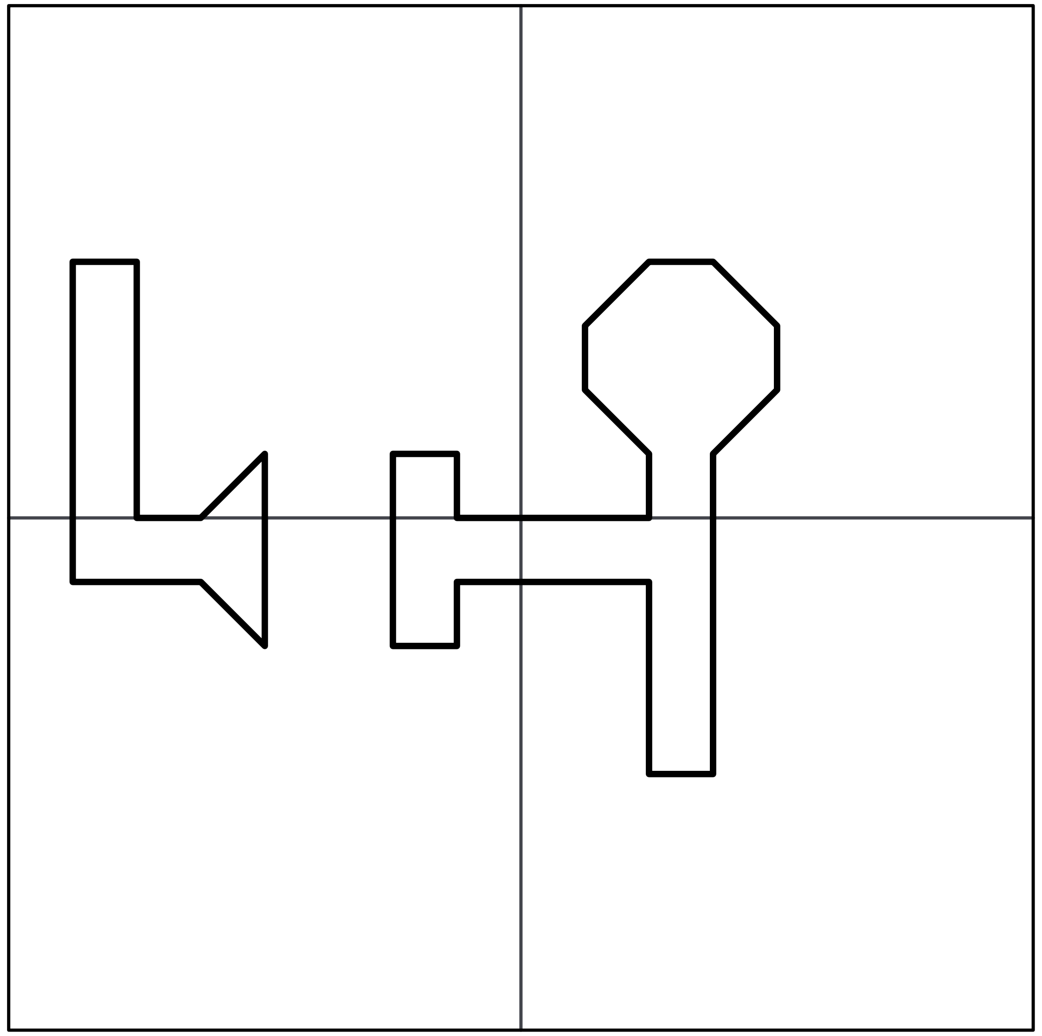
- polygon boundary in square
- common edge, or even just common point

Adaptation: partition squares until they no longer intersect a polygon or size is 1

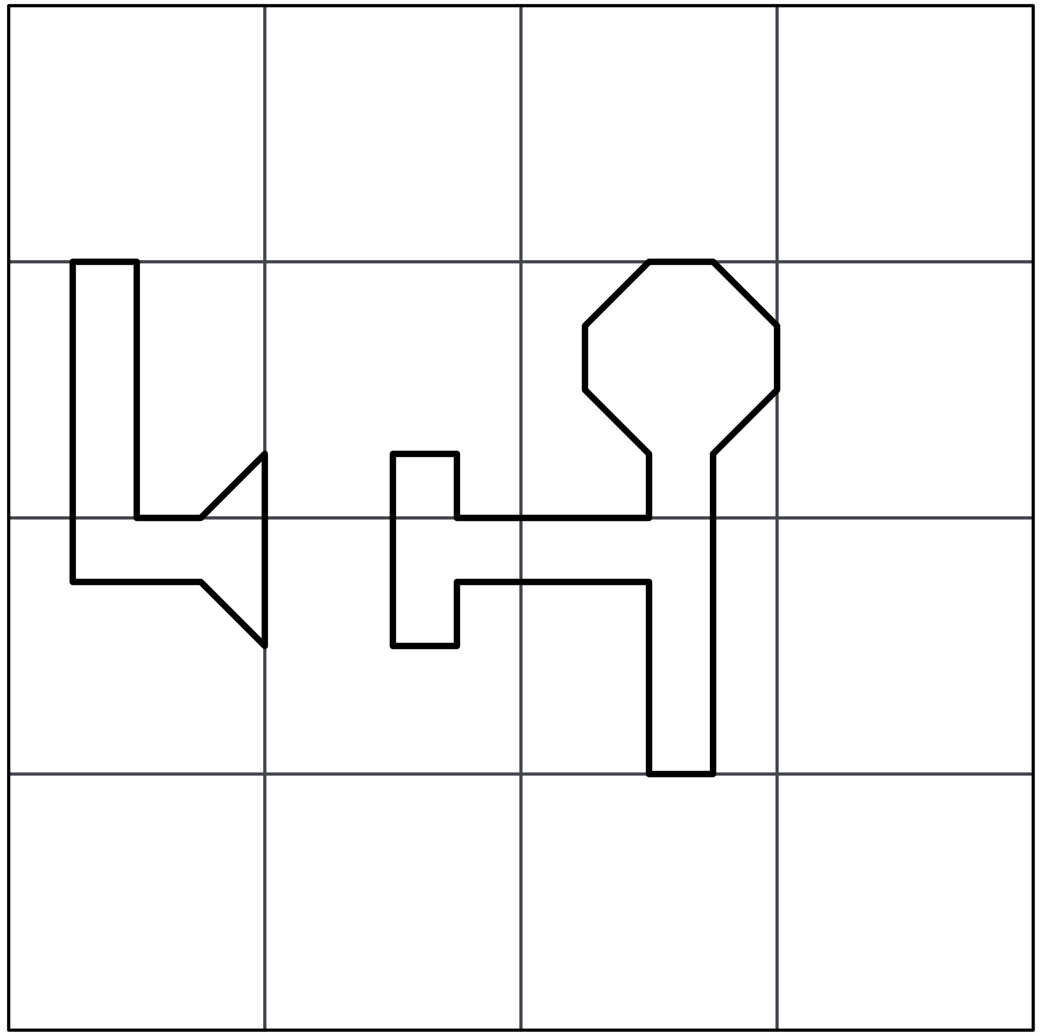
From quadtrees to meshes



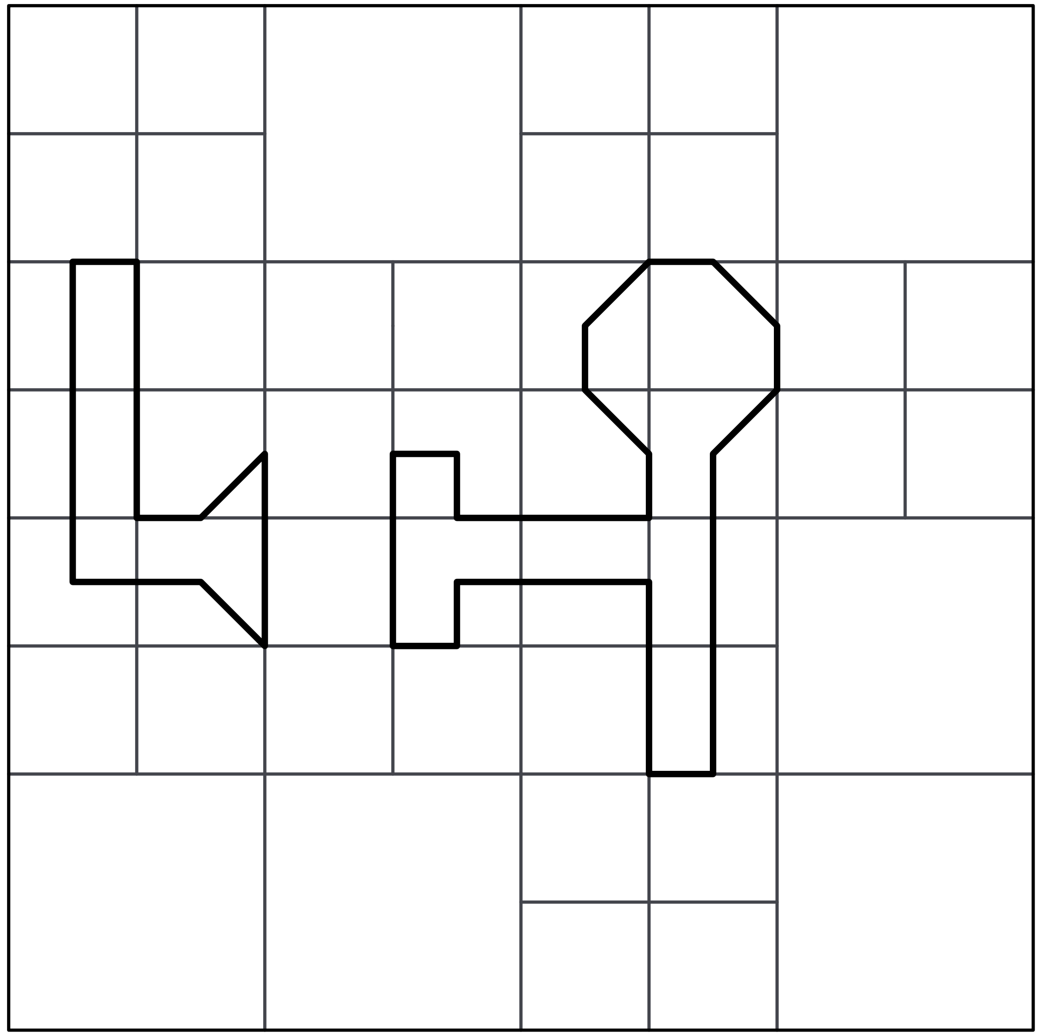
From quadtrees to meshes



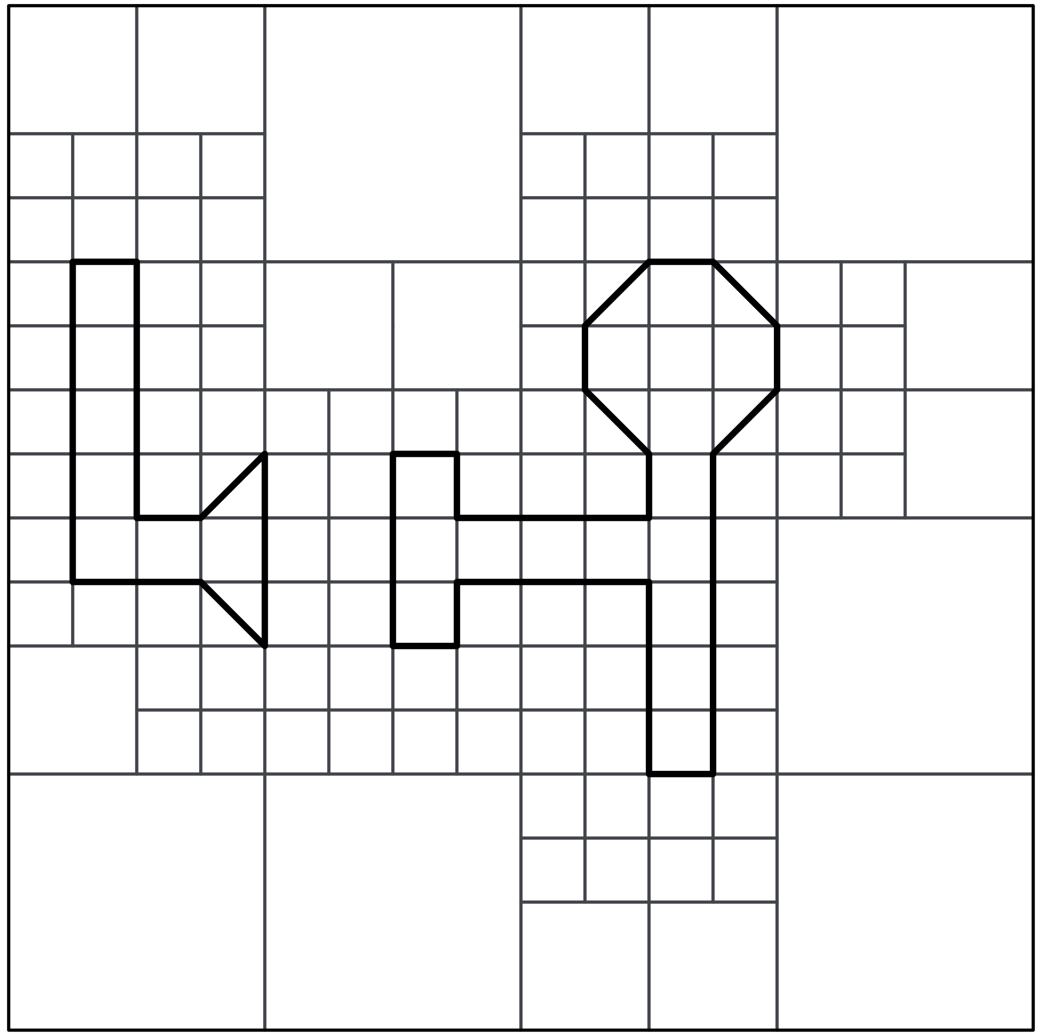
From quadtrees to meshes



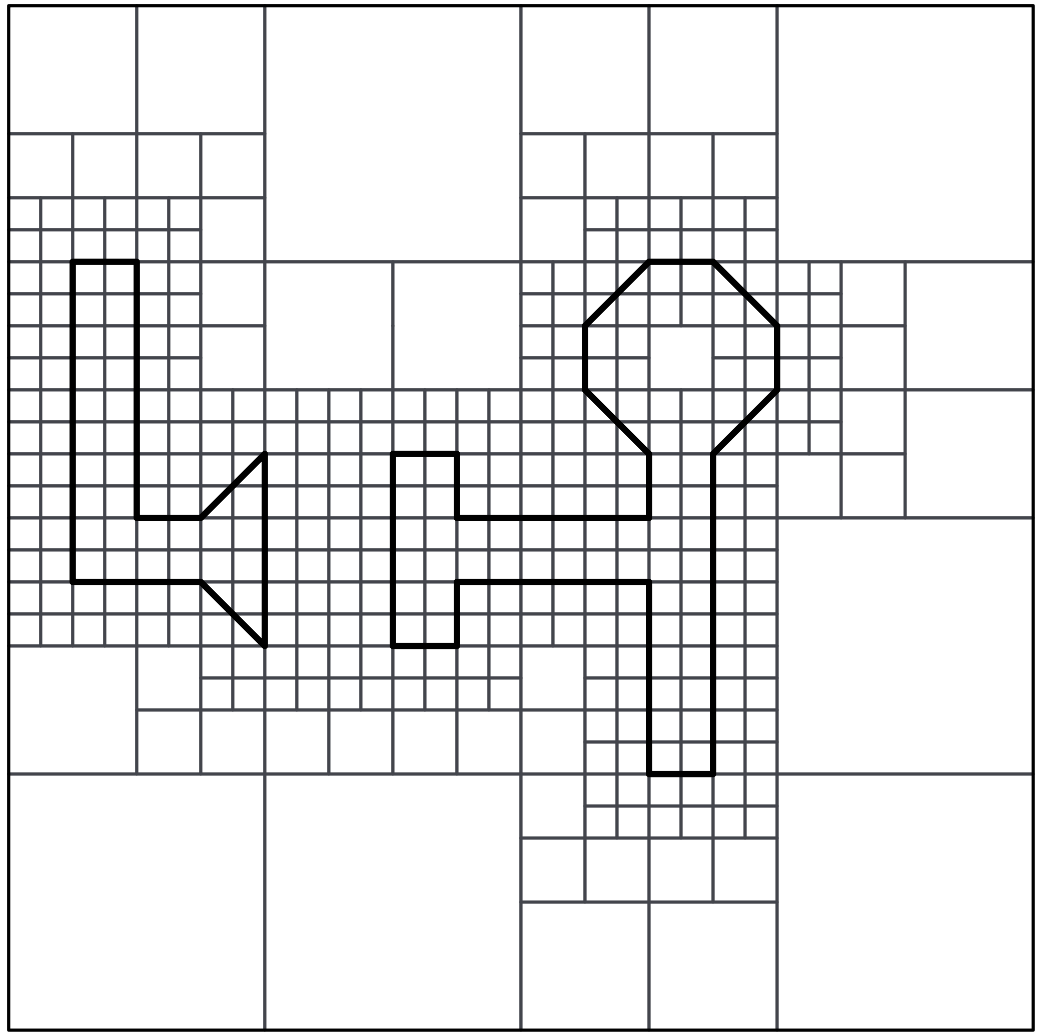
From quadtrees to meshes



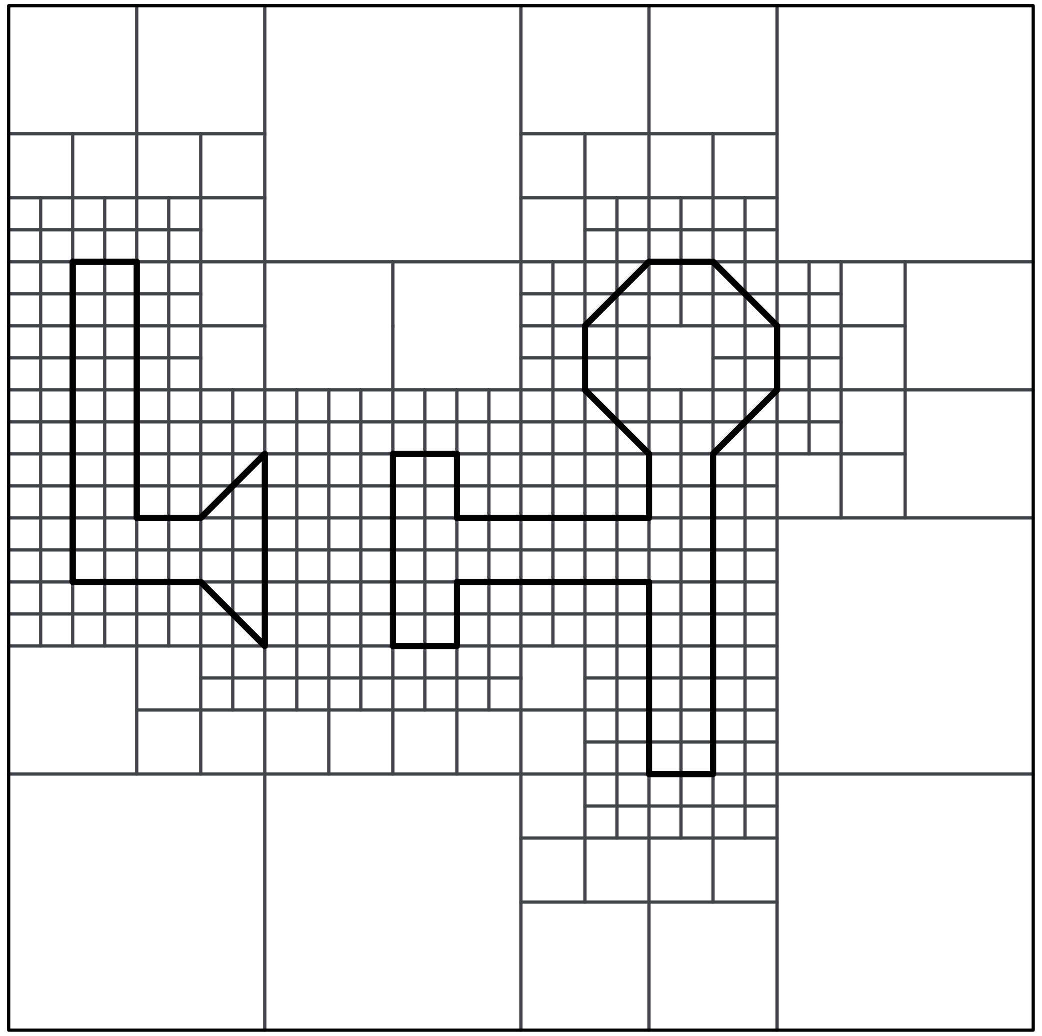
From quadtrees to meshes



From quadtrees to meshes

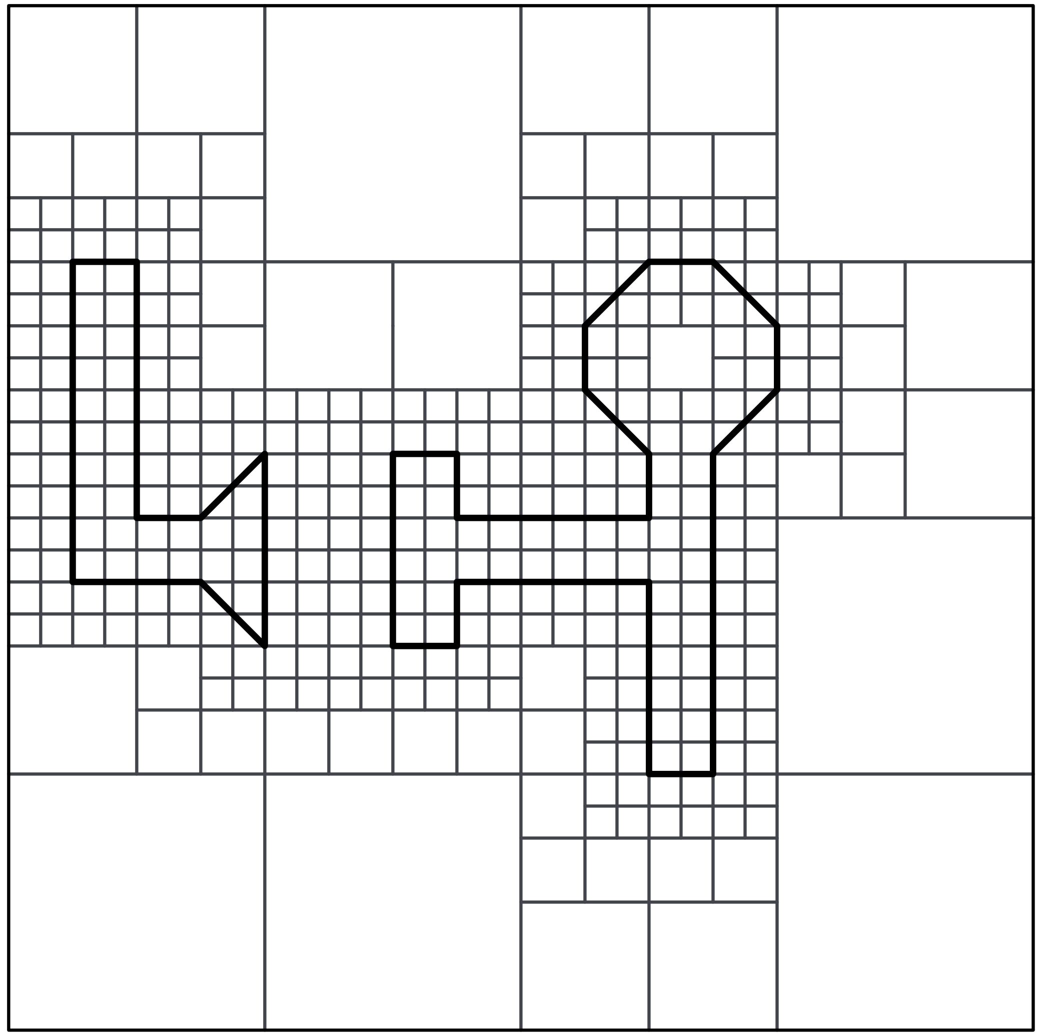


From quadtrees to meshes



Observation: the interior of a square in the quadtree can be intersected only by a diagonal

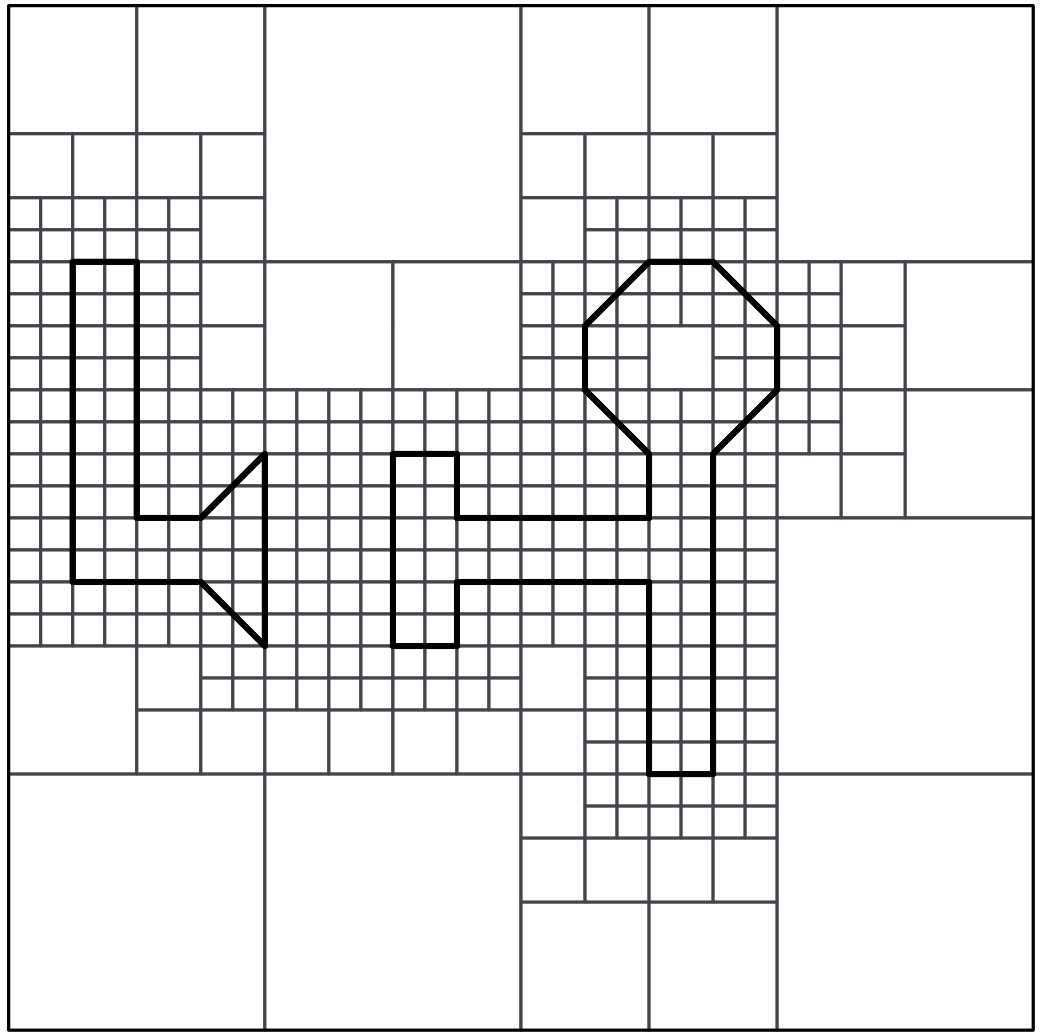
From quadtrees to meshes



Observation: the interior of a square in the quadtree can be intersected only by a diagonal

Question: How can we get a valid mesh?

From quadtrees to meshes

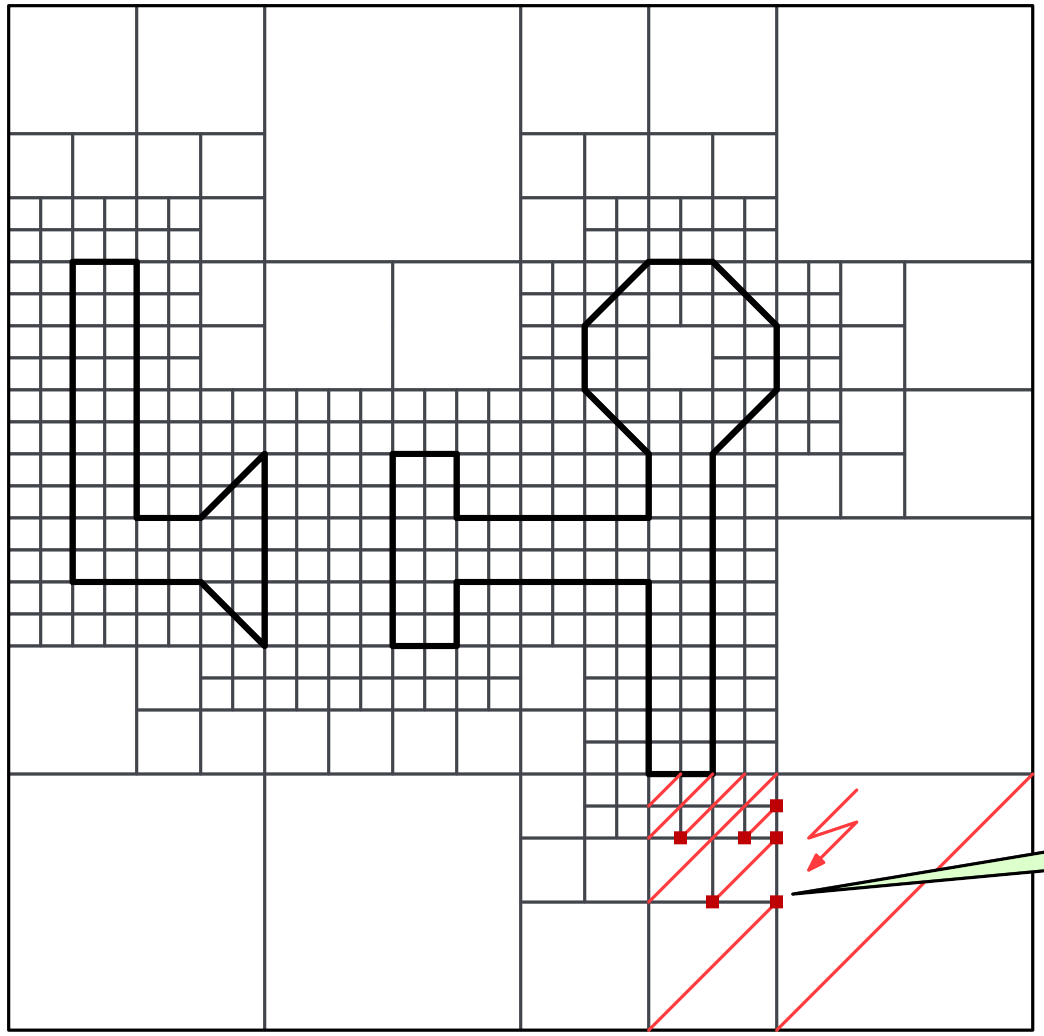


Observation: the interior of a square in the quadtree can be intersected only by a diagonal

Question: How can we get a valid mesh?

- Add diagonals for remaining squares?

From quadtrees to meshes



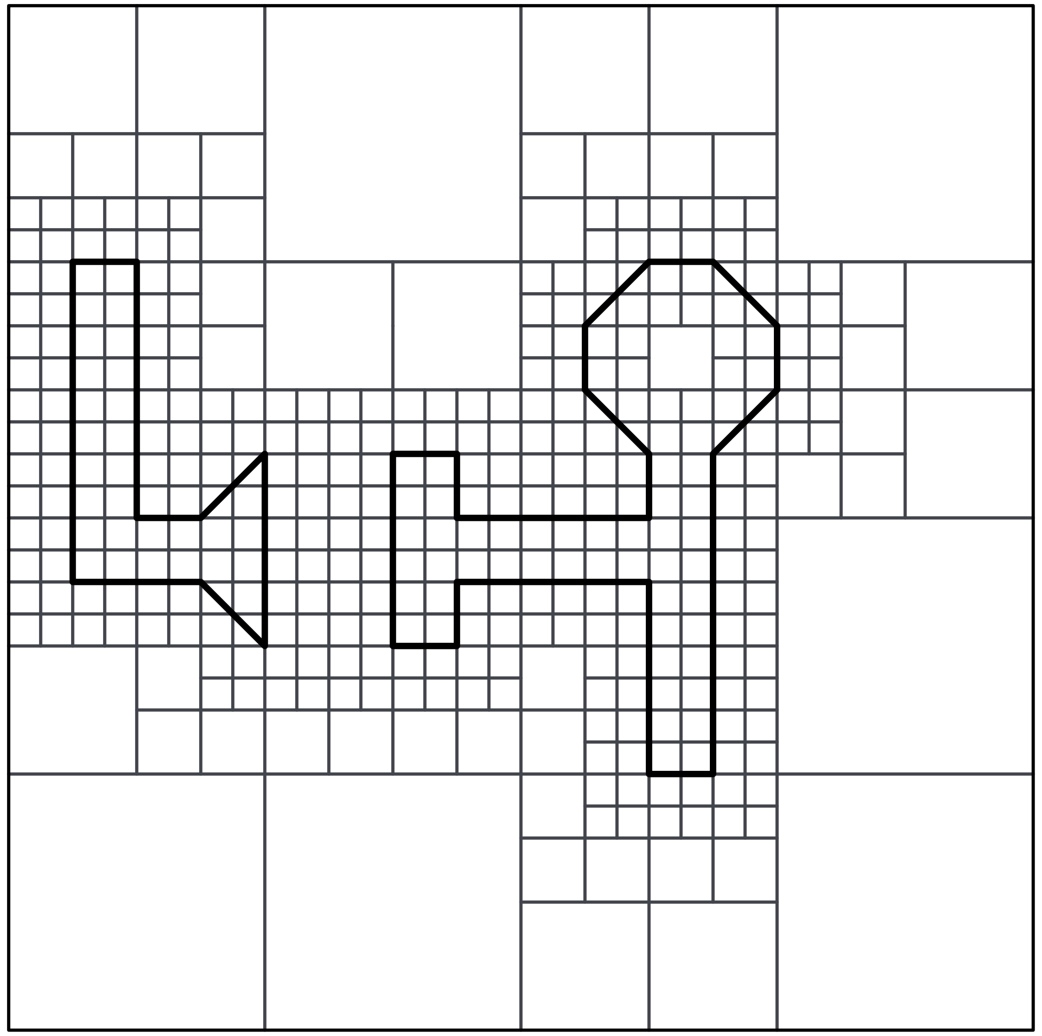
Observation: the interior of a square in the quadtree can be intersected only by a diagonal

Question: How can we get a valid mesh?

- Add diagonals for remaining squares? no!

non-conforming

From quadtrees to meshes

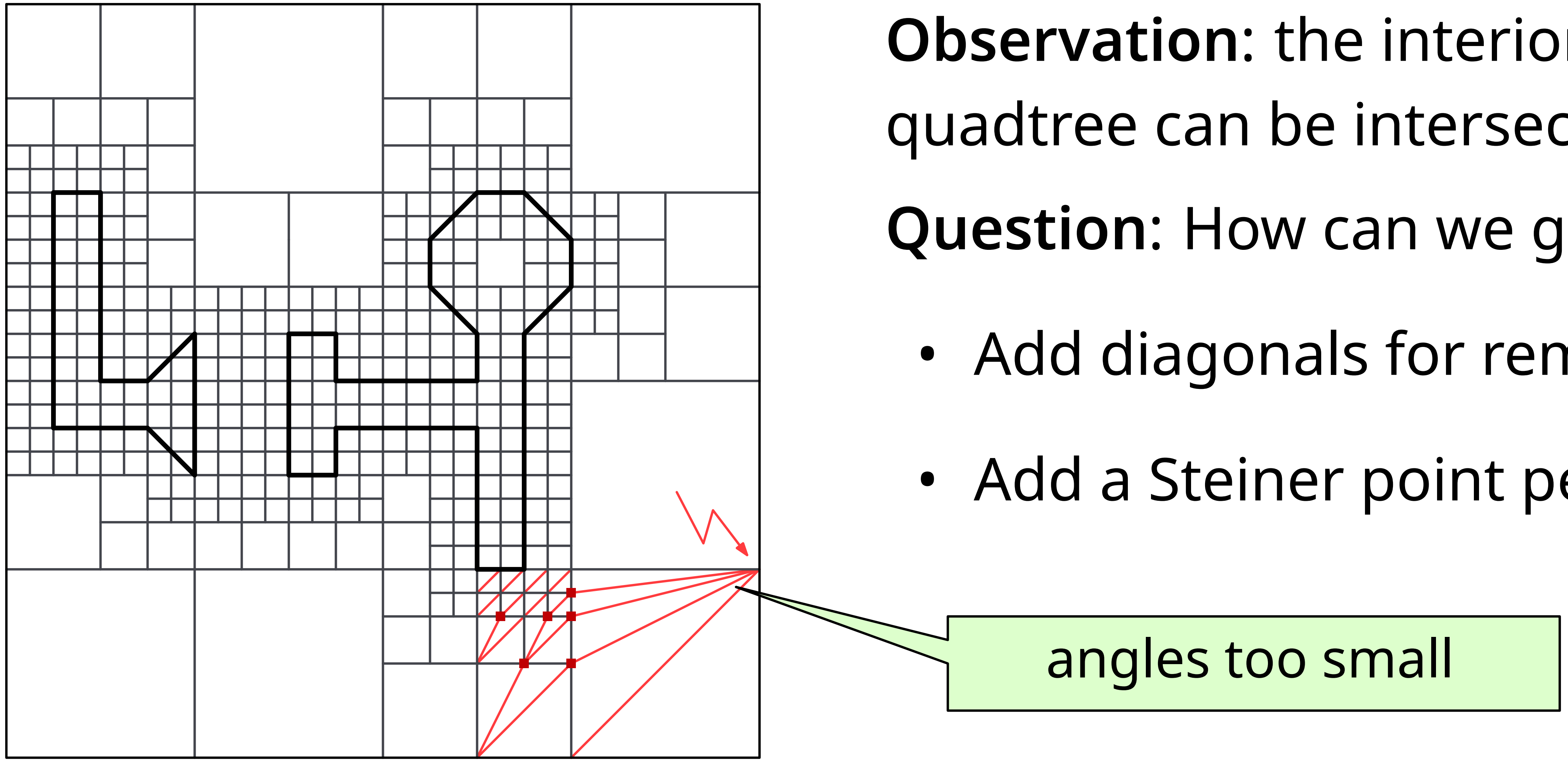


Observation: the interior of a square in the quadtree can be intersected only by a diagonal

Question: How can we get a valid mesh?

- Add diagonals for remaining squares? no!
- Add a Steiner point per cell?

From quadtrees to meshes

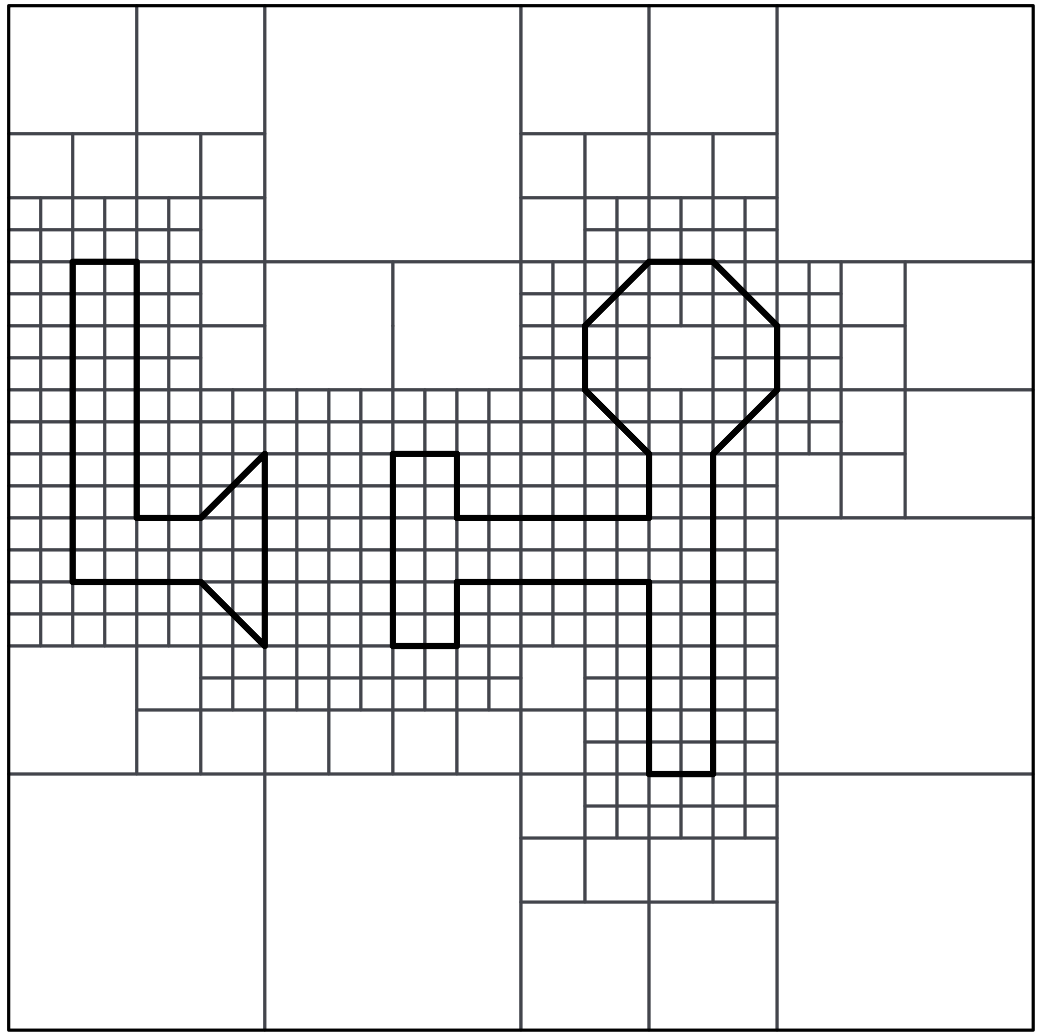


Observation: the interior of a square in the quadtree can be intersected only by a diagonal

Question: How can we get a valid mesh?

- Add diagonals for remaining squares? no!
- Add a Steiner point per cell? no!

From quadtrees to meshes

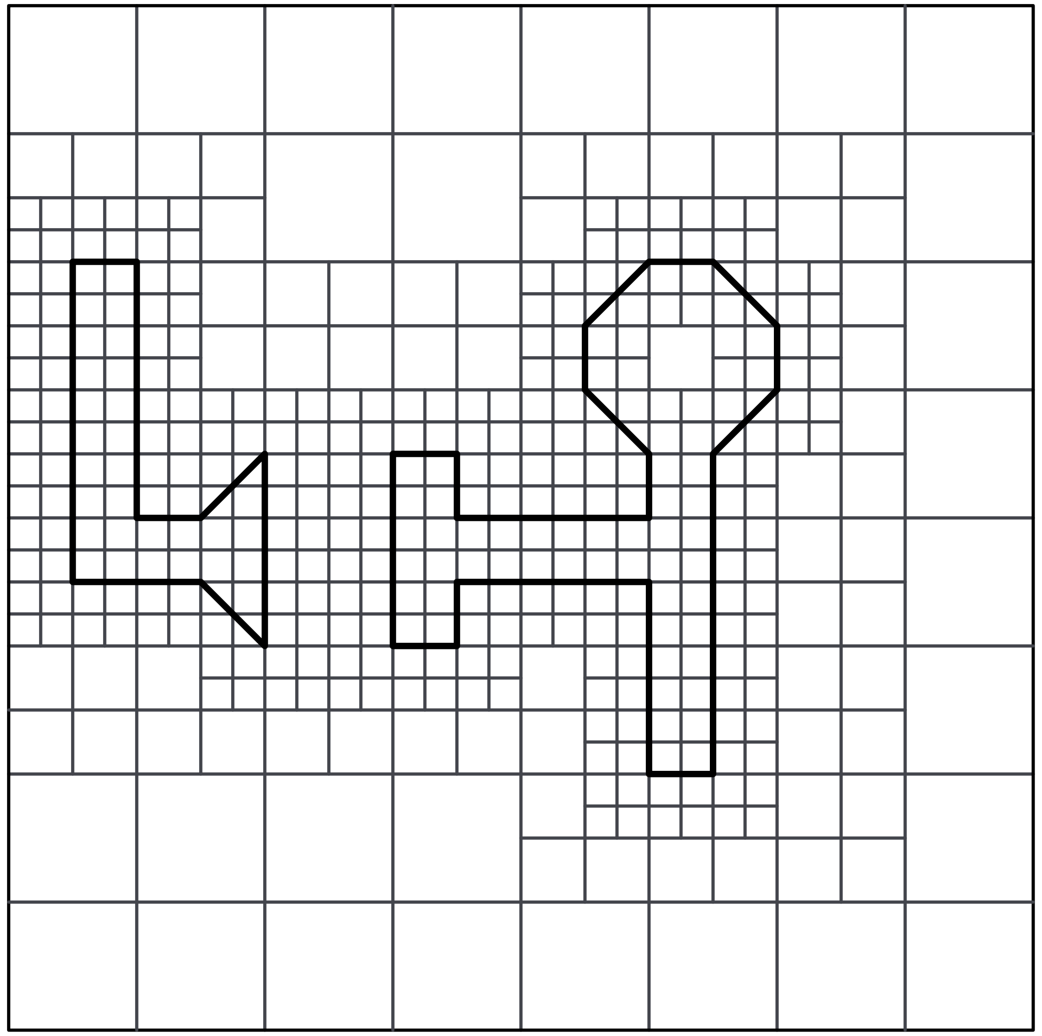


Observation: the interior of a square in the quadtree can be intersected only by a diagonal

Question: How can we get a valid mesh?

- Add diagonals for remaining squares? no!
- Add a Steiner point per cell? no!
- Balanced quadtree and add Steiner points if necessary!

From quadtrees to meshes

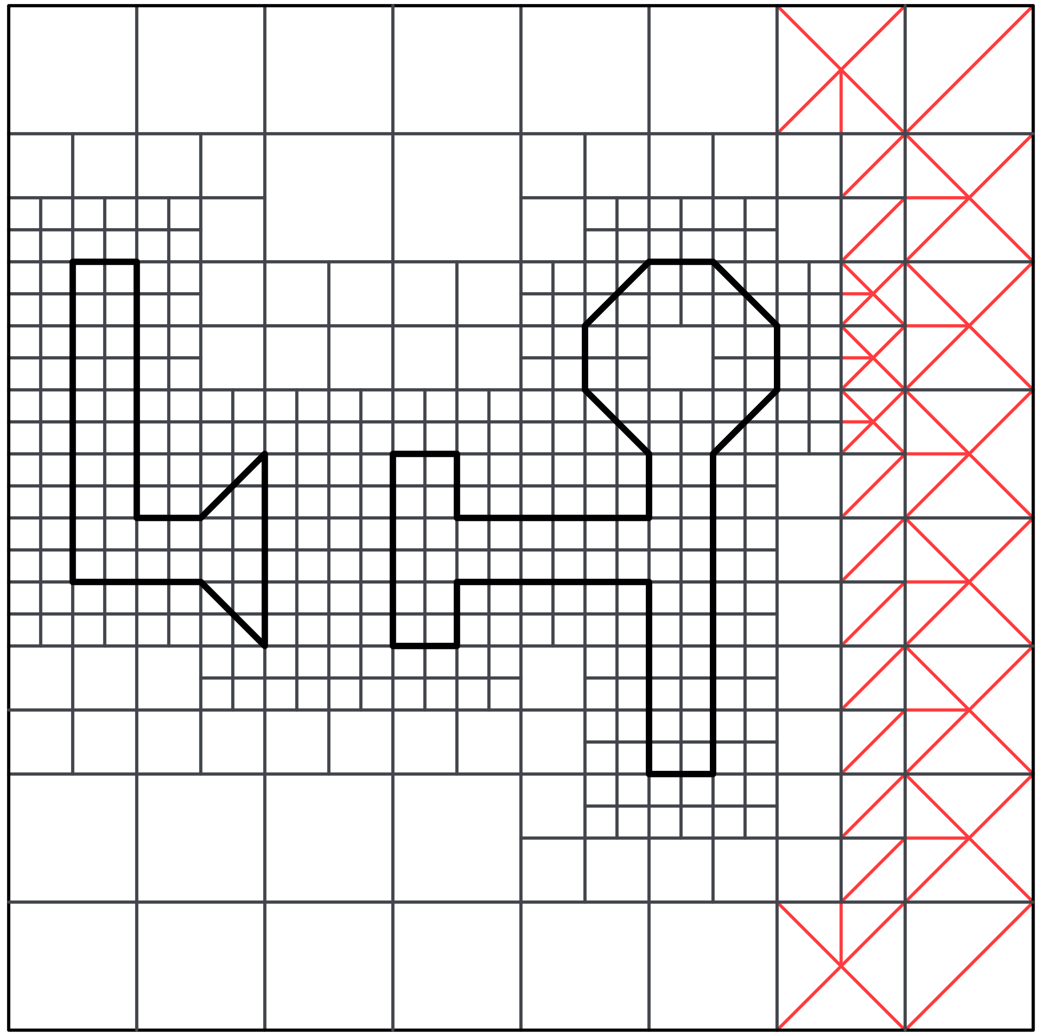


Observation: the interior of a square in the quadtree can be intersected only by a diagonal

Question: How can we get a valid mesh?

- Add diagonals for remaining squares? no!
- Add a Steiner point per cell? no!
- Balanced quadtree and add Steiner points if necessary!

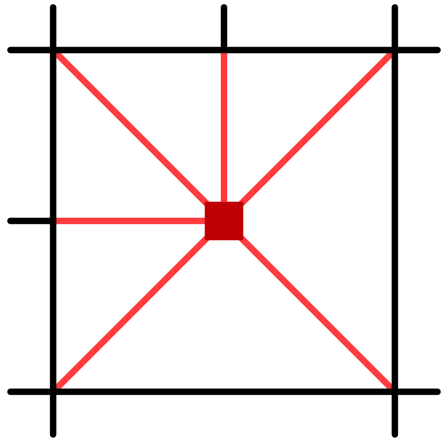
From quadtrees to meshes



Observation: the interior of a square in the quadtree can be intersected only by a diagonal

Question: How can we get a valid mesh?

- Add diagonals for remaining squares? no!
- Add a Steiner point per cell? no!
- Balanced quadtree and add Steiner points if necessary!



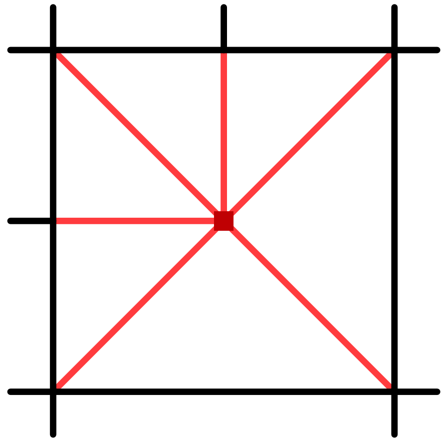
Triangulating quadtrees

TRIANGULATEQUADTREE($\mathcal{T}$)

Input: quadtree $\mathcal{T}$

Output: triangulation of $\mathcal{T}$

$\mathcal{D} \leftarrow$ DCEL for partition of Q by $\mathcal{T}$
for each facet f in $\mathcal{D}$ **do**
    if int(f) is intersected by a polygon **then**
        add corresponding diagonal in f to $\mathcal{D}$
    else
        if vertices only add corners of f **then**
            add a diagonal in f to $\mathcal{D}$
        else
            create Steiner point in the middle of f and
            connect in $\mathcal{D}$ to all vertices on ∂f
return $\mathcal{D}$



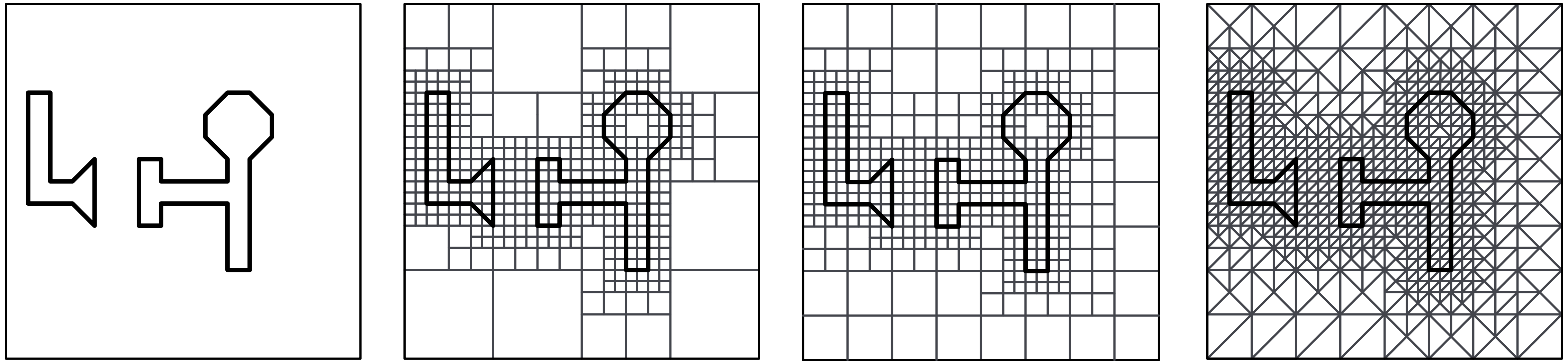
Algorithm

CREATEMESH(S)

Input: set S of octilinear polygons with integer coordinates in $Q = [0, 2^j] \times [0, 2^j]$

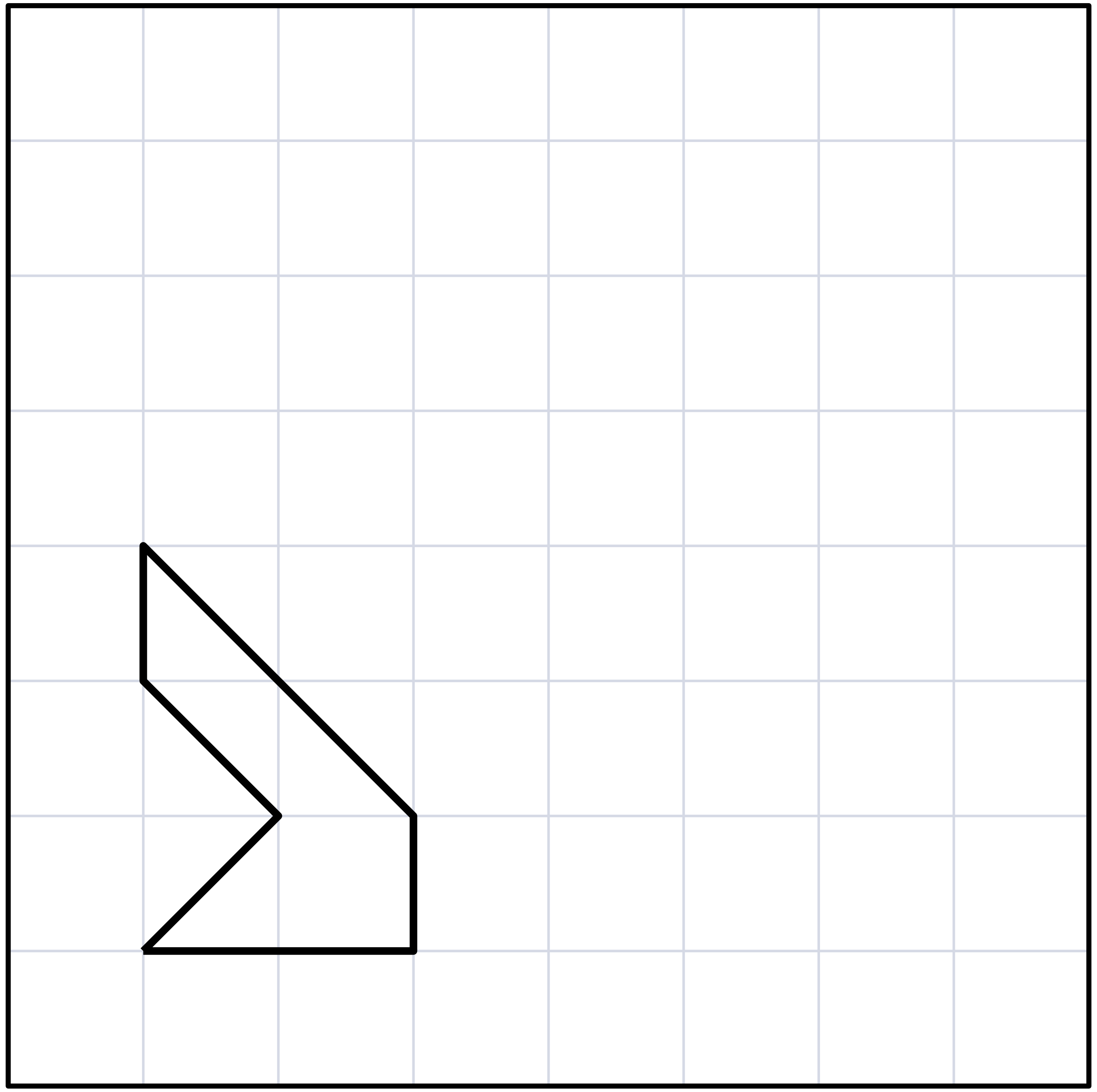
Output: valid, non-uniform triangular mesh S

1: $\mathcal{T} \leftarrow$ CREATEQUADTREE
2: $\mathcal{T} \leftarrow$ BALANCEQUADTREE$(\mathcal{T})$
3: $\mathcal{D} \leftarrow$ TRIANGULATEQUADTREE$(\mathcal{T})$
4: **return** $\mathcal{D}$



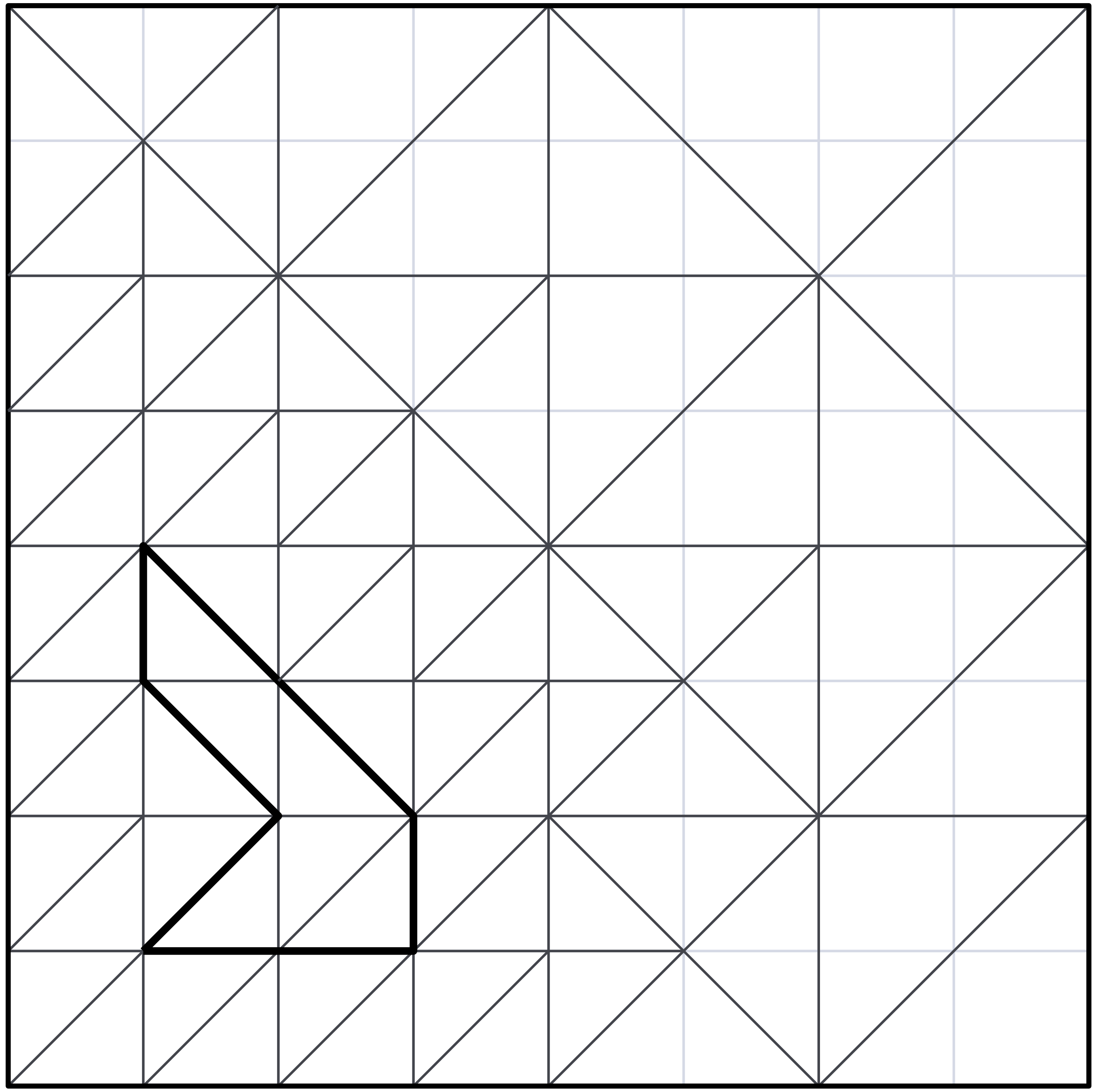
Exercise

Exercise:



Exercise

Exercise:



Summary

Theorem 4: Let S be a set of disjoint polygonal objects with vertices on a (integer) grid $[0, U] \times [0, U]$. Then

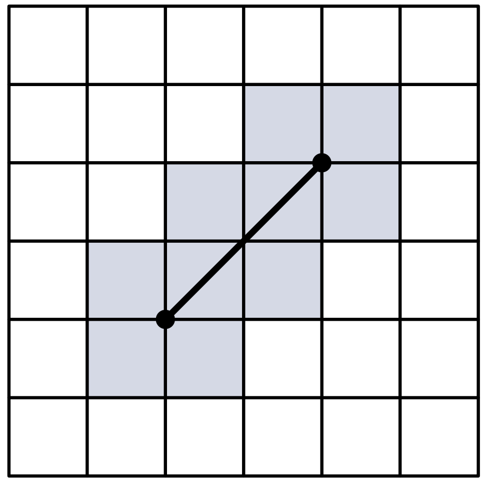
- there exists a non-uniform triangular mesh for S that is conforming, well-shaped and respects the input
- the number of triangles is $O(p(S) \log U)$, where $p(S)$ is the sum of (lengths of) perimeters of the objects
- the mesh can be constructed in $O(p(S) \log^2 U)$ time

Summary

Theorem 4: Let S be a set of disjoint polygonal objects with vertices on a (integer) grid $[0, U] \times [0, U]$. Then

- there exists a non-uniform triangular mesh for S that is conforming, well-shaped and respects the input
- the number of triangles is $O(p(S) \log U)$, where $p(S)$ is the sum of (lengths of) perimeters of the objects
- the mesh can be constructed in $O(p(S) \log^2 U)$ time

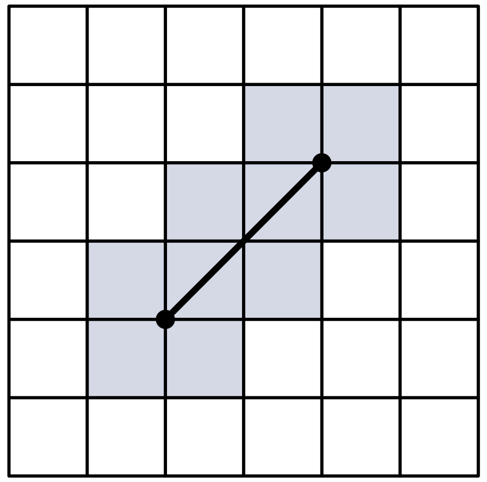
Proof: size of mesh: # squares intersected in a layer is $O(p(S))$, depth is $O(\log U)$



Summary

Theorem 4: Let S be a set of disjoint polygonal objects with vertices on a (integer) grid $[0, U] \times [0, U]$. Then

- there exists a non-uniform triangular mesh for S that is conforming, well-shaped and respects the input
- the number of triangles is $O(p(S) \log U)$, where $p(S)$ is the sum of (lengths of) perimeters of the objects
- the mesh can be constructed in $O(p(S) \log^2 U)$ time



Proof: size of mesh: # squares intersected in a layer is $O(p(S))$, depth is $O(\log U)$

construction time:

1. quadtree: linear in size
2. balancing: extra log-factor (by Thm 3)
3. triangulating: linear in size

Discussion

Can we compute/update compressed quadtrees efficiently?

Discussion

Can we compute/update compressed quadtrees efficiently?

Yes, skip quadtrees have complexity $O(n)$ and we can insert, delete and search in $O(\log n)$ time in a suitable model of computation

[Eppstein et al., '05]

Discussion

Can we compute/update compressed quadtrees efficiently?

Yes, skip quadtrees have complexity $O(n)$ and we can insert, delete and search in $O(\log n)$ time in a suitable model of computation

[Eppstein et al., '05]

Other applications?

Discussion

Can we compute/update compressed quadtrees efficiently?

Yes, skip quadtrees have complexity $O(n)$ and we can insert, delete and search in $O(\log n)$ time in a suitable model of computation

[Eppstein et al., '05]

Other applications?

Quadtrees are used in many applications including computer graphics, image processing, GIS etc. In geometry used for approximation algorithms, but also connections to Delaunay triangulations.

Discussion

Can we compute/update compressed quadtrees efficiently?

Yes, skip quadtrees have complexity $O(n)$ and we can insert, delete and search in $O(\log n)$ time in a suitable model of computation

[Eppstein et al., '05]

Other applications?

Quadtrees are used in many applications including computer graphics, image processing, GIS etc. In geometry used for approximation algorithms, but also connections to Delaunay triangulations.

As always: higher dimensions?

Discussion

Can we compute/update compressed quadtrees efficiently?

Yes, skip quadtrees have complexity $O(n)$ and we can insert, delete and search in $O(\log n)$ time in a suitable model of computation

[Eppstein et al., '05]

Other applications?

Quadtrees are used in many applications including computer graphics, image processing, GIS etc. In geometry used for approximation algorithms, but also connections to Delaunay triangulations.

As always: higher dimensions?

Directly generalize. In 3D quadtrees $\rightarrow$ octtrees