

# Computational Geometry · Lecture

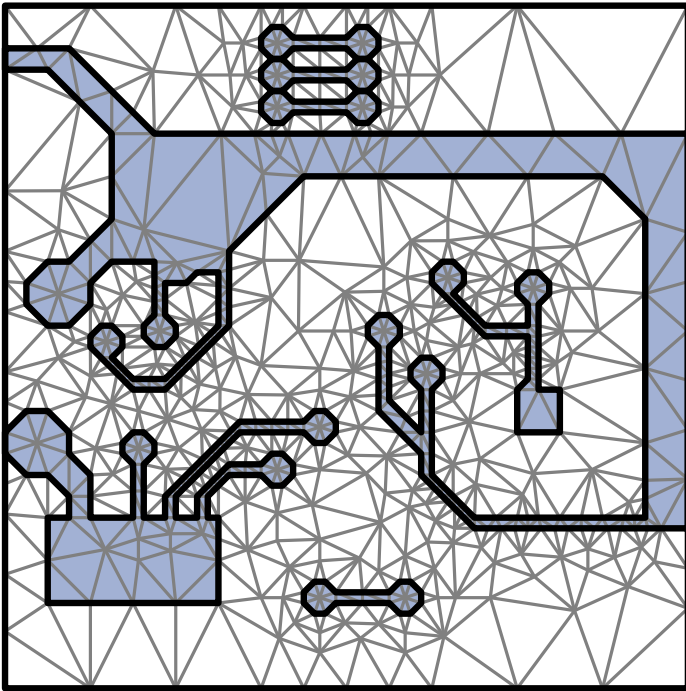
## Quadtrees and Meshing

INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Tamara Mchedlidze · Darren Strash  
14.12.2015



# Motivation: Meshing PC Board Layouts



To simulate the heat produced on boards we can use the *finite element method* (FEM):

- decompose the board in small homogeneous elements (e.g., triangles)  
→ mesh
- heat generation and impact on neighbors for each element known
- approximate numerically the entire heat generation of board

Quality properties of FEM:

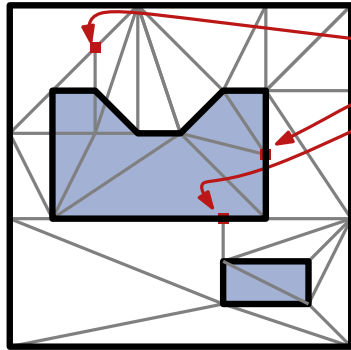
- the finer the mesh, the better the approximation
- the larger the mesh, the faster the calculation
- the more compact the elements, the faster the convergence

**Goal:**

- adaptive mesh size (small on materials, otherwise coarser)
- fat triangles (not too narrow)

# Adaptive Triangular Mesh

**Given:** Square  $Q = [0, U] \times [0, U]$  for power of two  $U = 2^j$  with *octilinear*, integer-coordinate polygons inside.



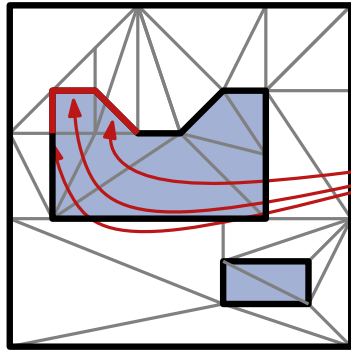
disallowed triangle vertices

**Goal:** Triangular mesh for  $Q$  with the following properties

- no triangle vertex in interior of triangular mesh

# Adaptive Triangular Mesh

**Given:** Square  $Q = [0, U] \times [0, U]$  for power of two  $U = 2^j$  with *octilinear*, integer-coordinate polygons inside.



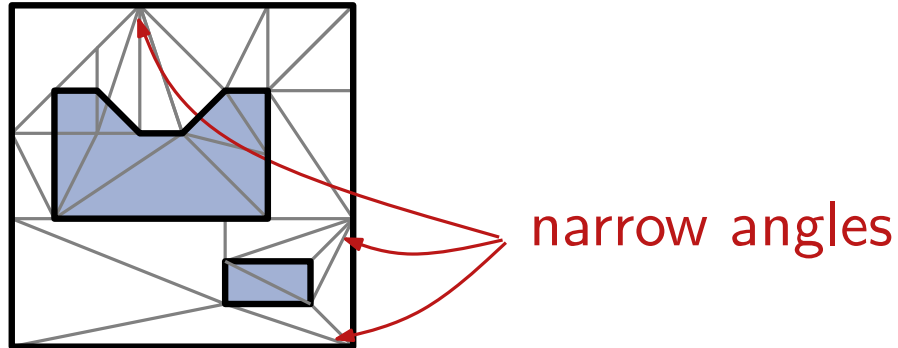
not part of the mesh

**Goal:** Triangular mesh for  $Q$  with the following properties

- no triangle vertex in interior of triangular mesh
- input edges must be part of the triangulation

# Adaptive Triangular Mesh

**Given:** Square  $Q = [0, U] \times [0, U]$  for power of two  $U = 2^j$  with *octilinear*, integer-coordinate polygons inside.

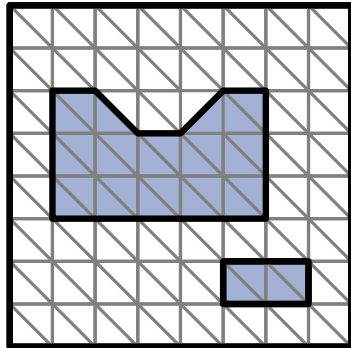


**Goal:** Triangular mesh for  $Q$  with the following properties

- valid {
- no triangle vertex in interior of triangular mesh
  - input edges must be part of the triangulation
  - triangle angle between  $45^\circ$  and  $90^\circ$

# Adaptive Triangular Mesh

**Given:** Square  $Q = [0, U] \times [0, U]$  for power of two  $U = 2^j$  with *octilinear*, integer-coordinate polygons inside.



uniform mesh

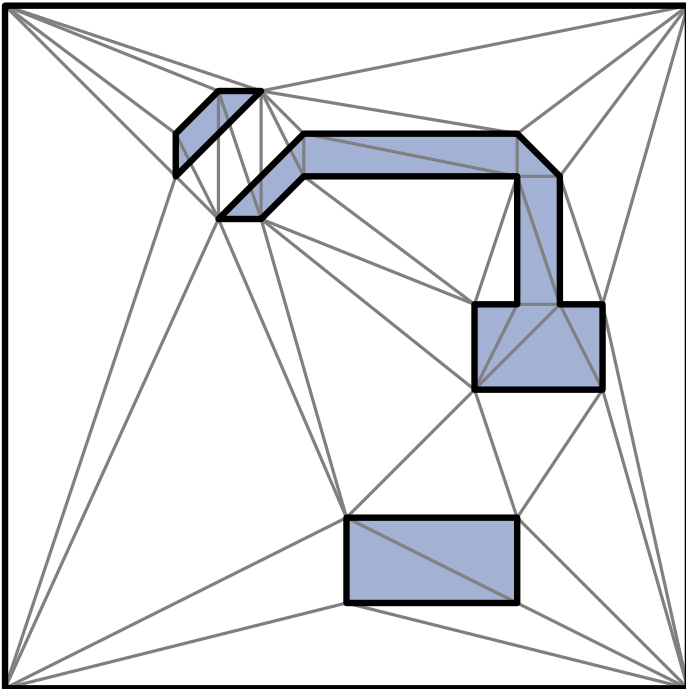
**Goal:** Triangular mesh for  $Q$  with the following properties

- valid {
- no triangle vertex in interior of triangular mesh
  - input edges must be part of the triangulation
  - triangle angle between  $45^\circ$  and  $90^\circ$
  - **adaptive** (i.e., fine at the polygon edges, otherwise coarser)

Do we already have meaningful triangulations of  $Q$ ?

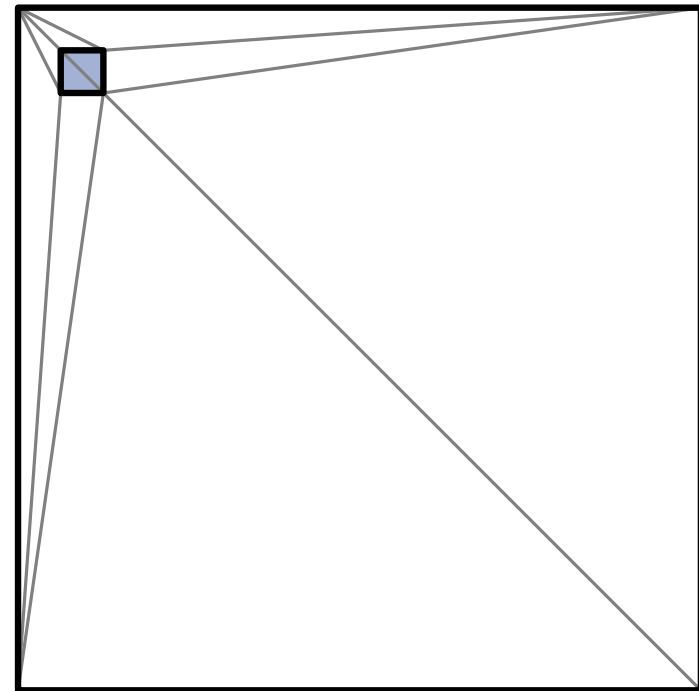
# Delaunay Triangulation?

- maximize smallest angle
- is defined for points and ignores existing edges



# Delaunay Triangulation?

- maximize smallest angle
- is defined for points and ignores existing edges
- can still produce very small angles

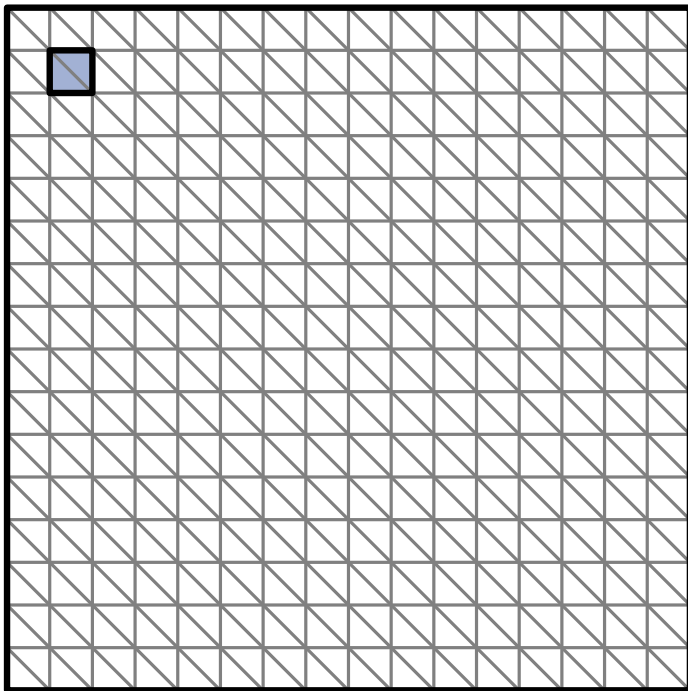




# Delaunay Triangulation?

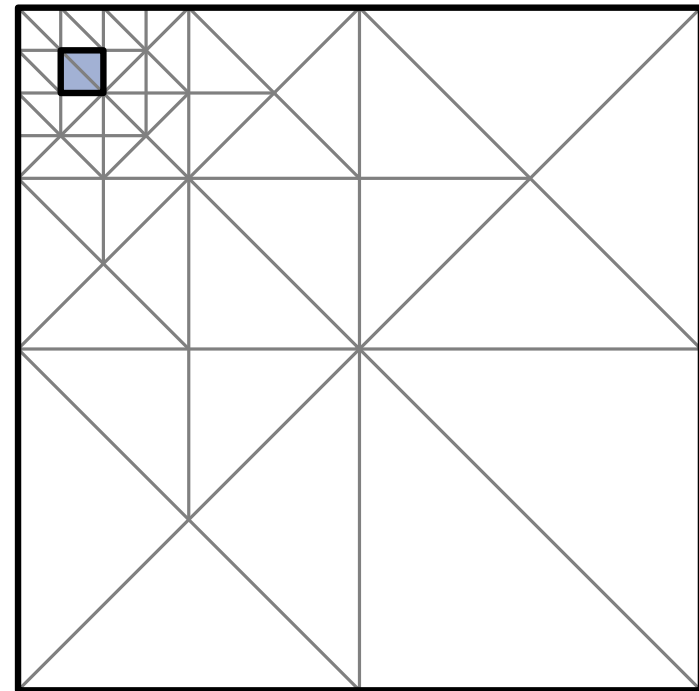
- maximize smallest angle
- is defined for points and ignores existing edges
- can still produce very small angles
- does not use additional *Steiner* points

Allowed angles, but uniform



512 triangles

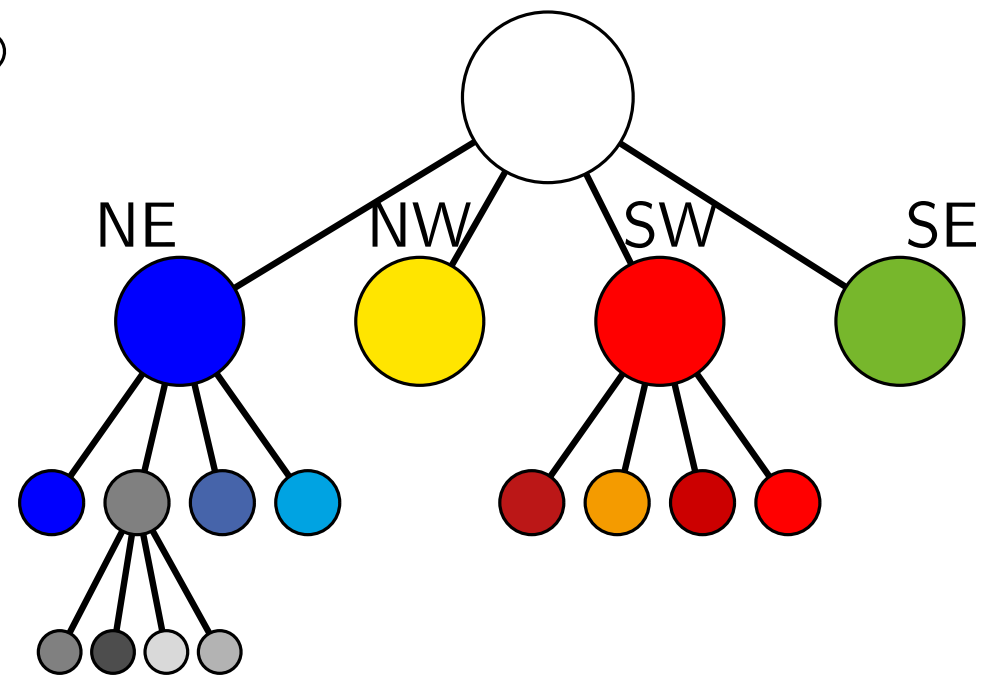
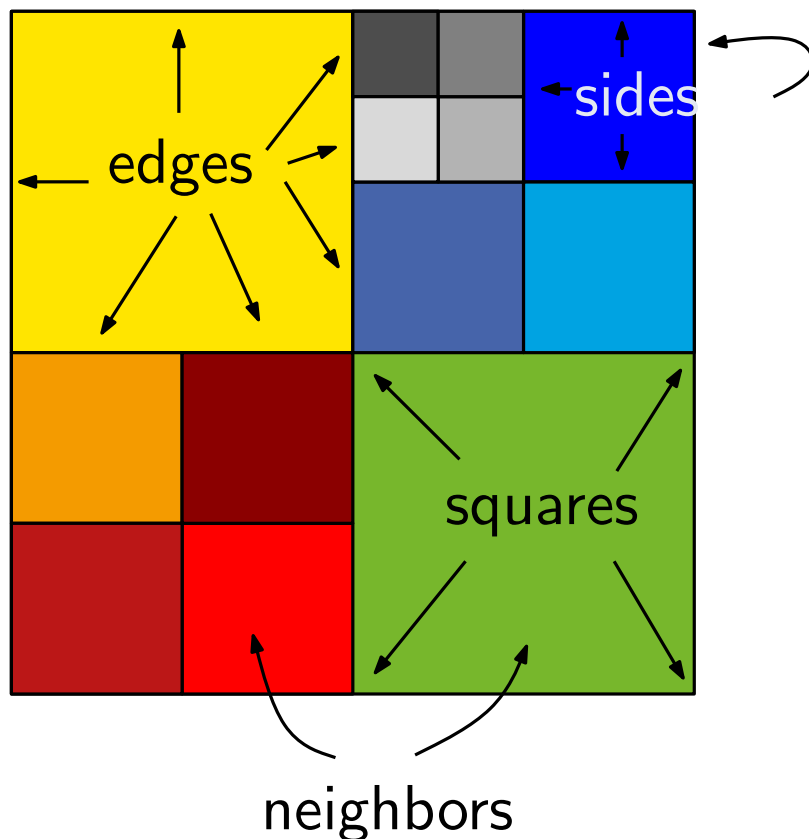
Allowed angles and adaptive



52 triangles

# 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.:** 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

$$P_{NE} := \{p \in P \mid p_x > x_{\text{mid}} \text{ and } p_y > y_{\text{mid}}\}$$

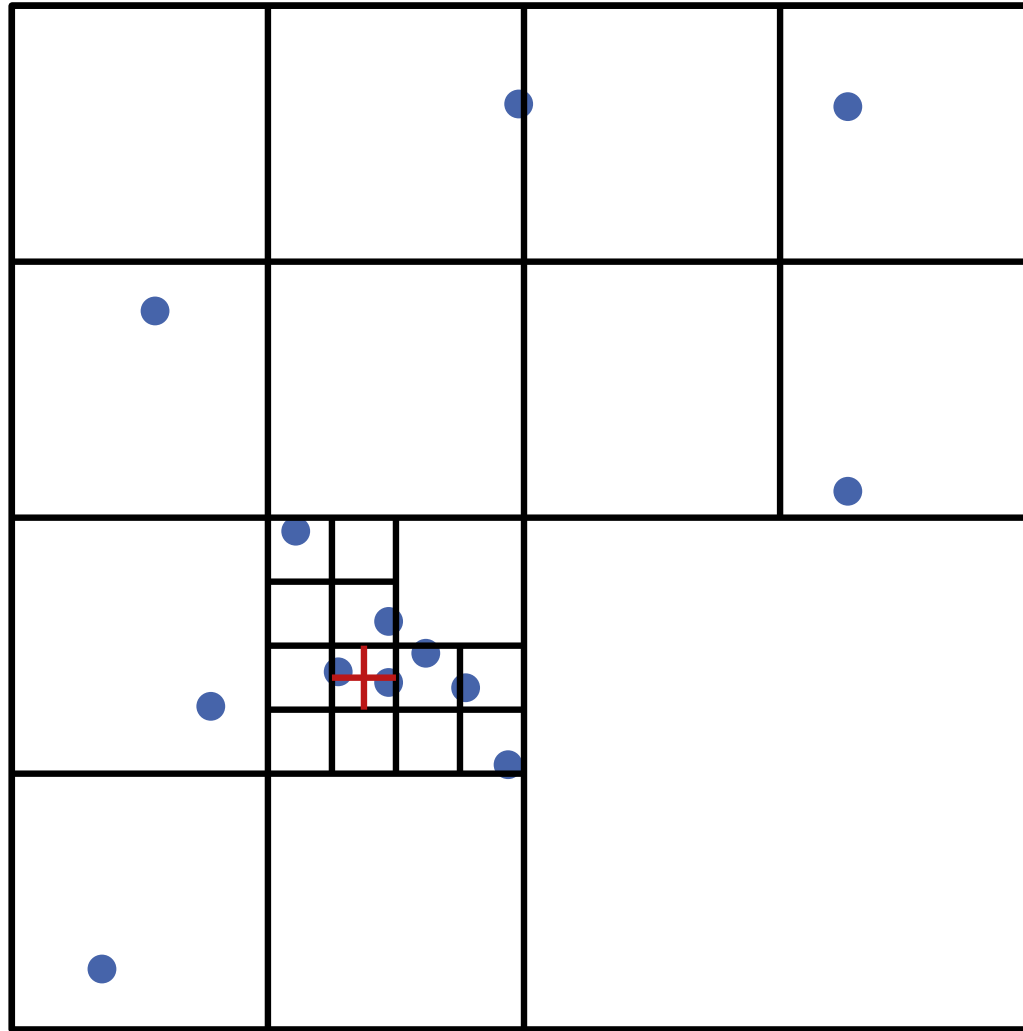
$$P_{NW} := \{p \in P \mid p_x \leq x_{\text{mid}} \text{ and } p_y > y_{\text{mid}}\}$$

$$P_{SW} := \{p \in P \mid p_x \leq x_{\text{mid}} \text{ and } p_y \leq y_{\text{mid}}\}$$

$$P_{SE} := \{p \in P \mid p_x > x_{\text{mid}} \text{ and } p_y \leq y_{\text{mid}}\}$$

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

# Example



# 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.

# 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**  $v = SW-/SE$ -child of  $\pi$  **then return**  $NW-/NE$ -child of  $\pi$

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

**if**  $\mu = \text{nil}$  or  $\mu$  leaf **then**

**return**  $\mu$

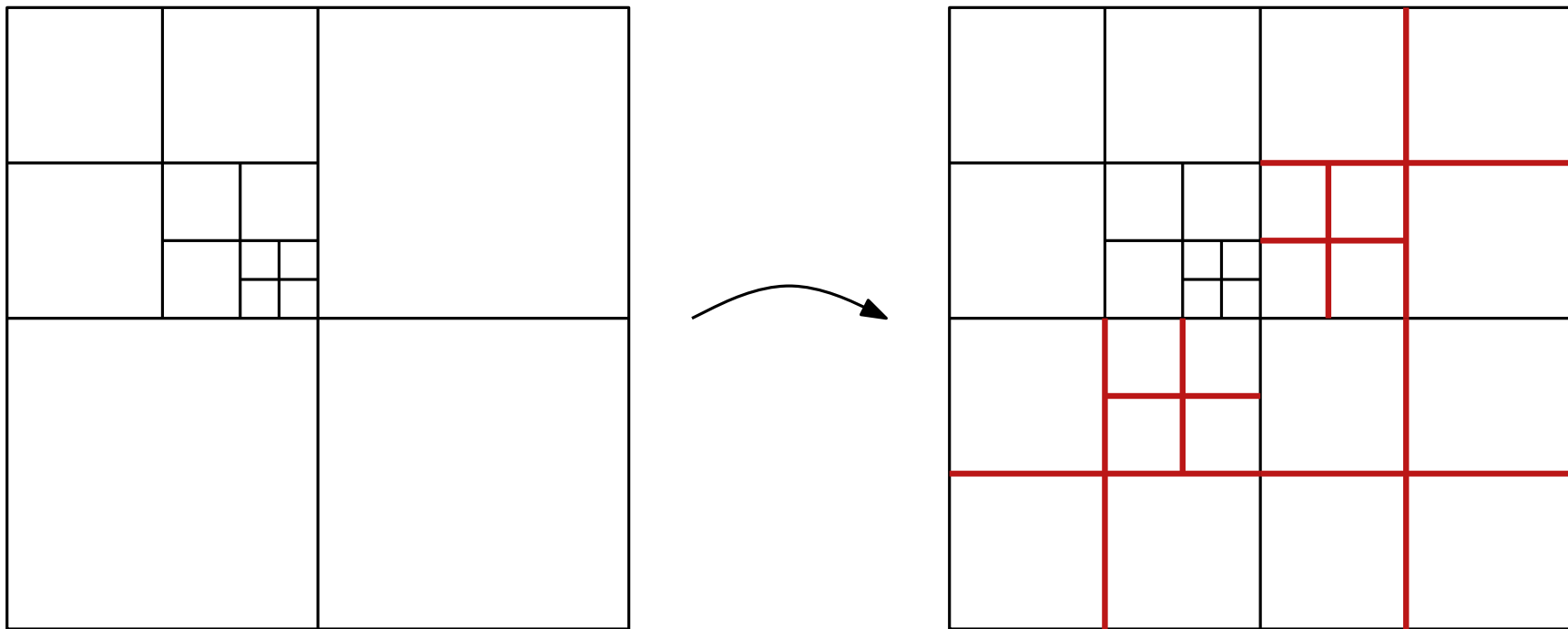
**else**

**if**  $v = NW-/NE$ -child of  $\pi$  **then return**  $SW-/SE$ -child of  $\mu$

**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 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.



# Balancing Quadrees

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**  $\mathcal{T}$

**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.



## Recall:

**Given:** Square  $Q = [0, U] \times [0, U]$  for power of two  $U = 2^j$  with *octilinear*, integer-coordinate polygons inside.

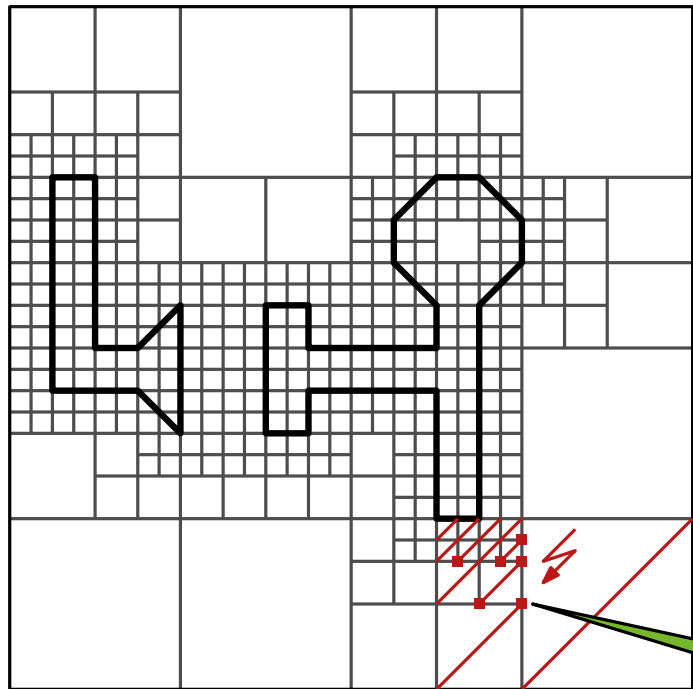
**Goal:** Triangular mesh for  $Q$  with the following properties

- valid {
- no triangle vertex in interior of triangular mesh
  - input edges must be part of the triangulation
  - triangle angle between  $45^\circ$  and  $90^\circ$
  - **adaptive** (i.e., fine at the polygon edges, otherwise coarser)
- Squares and edges are finished

**Idea:** Use a quadtree as the basis for the triangular mesh!

**Adaptation:** Split squares until they are no longer cut by a polygon or have size 1

# From Quadtree to Triangular Mesh



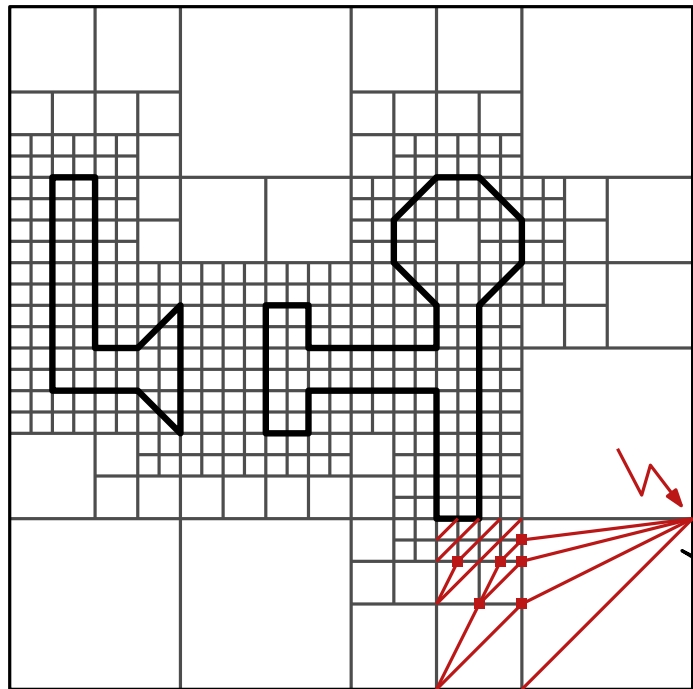
**Obs.:** when the interior of a square in a quadtree is intersected by an edge, then it is a diagonal.

How do you get a valid triangular mesh?

nodes on interior edges

- Diagonals for all remaining squares?

# From Quadtree to Triangular Mesh



**Obs.:** when the interior of a square in a quadtree is intersected by an edge, then it is a diagonal.

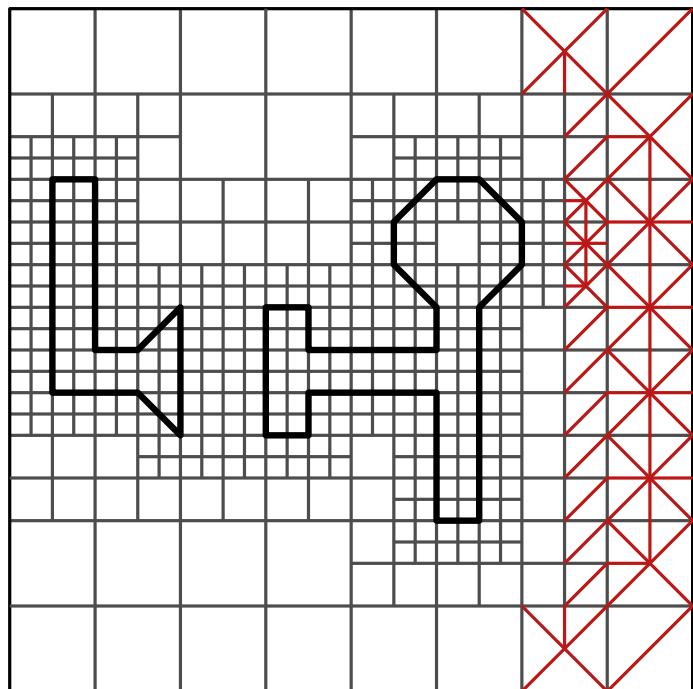
How do you get a valid triangular mesh?

angle is too small

- Diagonals for all remaining squares?
- Use subdivision nodes in triangulation?

no!

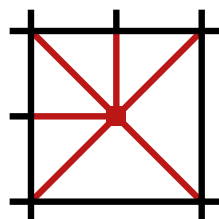
# From Quadtree to Triangular Mesh



**Obs.:** when the interior of a square in a quadtree is intersected by an edge, then it is a diagonal.

How do you get a valid triangular mesh?

- Diagonals for all remaining squares? no!
- Use subdivision nodes in triangulation? no!
- Balance quadtree and, if necessary, add a Steiner vertex!



# Algorithm

GenerateMesh( $S$ )

**Input:** Set  $S$  octilinear, integer-coordinate polygons in  
 $Q = [0, 2^j] \times [0, 2^j]$

**Output:** valid, adaptive triangular mesh for  $S$

$\mathcal{T} \leftarrow$  CreateQuadtree

$\mathcal{T} \leftarrow$  BalanceQuadtree( $\mathcal{T}$ )

$\mathcal{D} \leftarrow$  DCEL for subdivisions of  $Q$  by  $\mathcal{T}$

**foreach** Face  $f$  in  $\mathcal{D}$  **do**

**if**  $\text{int}(f) \cap S \neq \emptyset$  **then**

        | add appropriate diagonals in  $f$  to  $\mathcal{D}$

**else**

**if** Nodes only on the corners of  $f$  **then**

            | add a diagonal in  $f$  to  $\mathcal{D}$

**else**

            | generate Steiner point in the middle of  $f$  and connect it to  
            | all nodes in  $\partial f$  of  $\mathcal{D}$

**return**  $\mathcal{D}$

**Theorem 4:** For a set  $S$  of disjoint octilinear, integer-coordinate polygons with total perimeter  $p(S)$  in a square  $Q = [0, U] \times [0, U]$  we can compute in  $O(p(S) \log^2 U)$  time a valid adaptive triangular mesh with  $O(p(S) \log U)$  triangles.

## Are there quadtree variants with space linear in $n$ ?

Yes, if you contract internal nodes with only one non-empty child you get a so-called *compressed quadtree* (see exercise); a more advanced data structure is the *skip quadtree* with  $O(n)$  space and insert, remove, and search in  $O(\log n)$  time. [Eppstein et al., '05]

## What other applications are there?

Quadtrees are frequently used in practice in Computer Graphics, Image Processing, GIS and others, for range queries, although they are theoretically worse than range trees or  $kd$ -trees. (see exercise)

## As always: higher dimensions?

Quadtrees can be easily generalized to higher dimensions. Then they are also called octrees.