Meshing (chapter 14 and [1])

Elisha Sacks

[1] Cheng, Dey, and Shewchuk. *Delaunay Mesh Generation*, Chapman and Hall, 2012.

Mesh Types

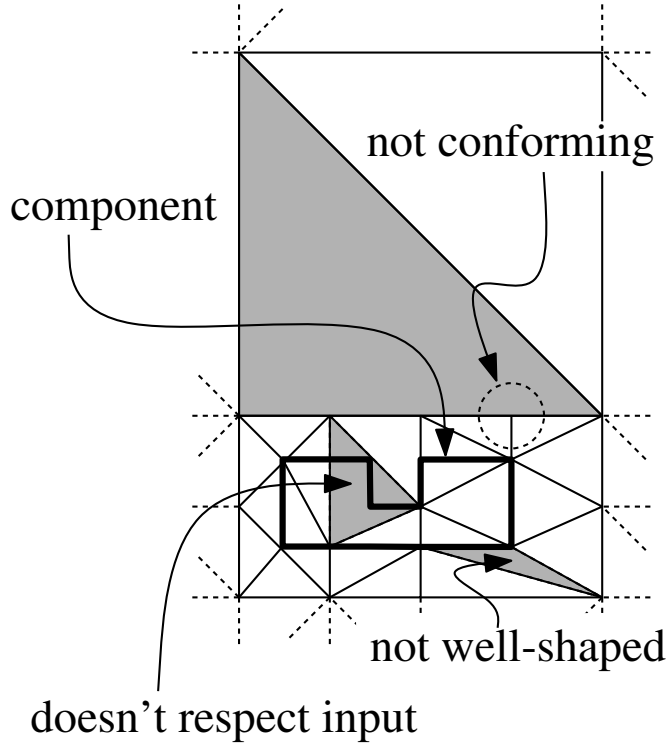
- Dimension: area in 2D, surface in 3D, volume in 3D.
- Domain: polygonal, polyhedral, curved.
- Element: triangle, tetrahedron, quadrilateral, hexahedron.
- Structured: regular grid with indexing.
- Unstructured: elements comprise a general subdivision.

Our Topic

Unstructured triangle meshing of polygonal domains

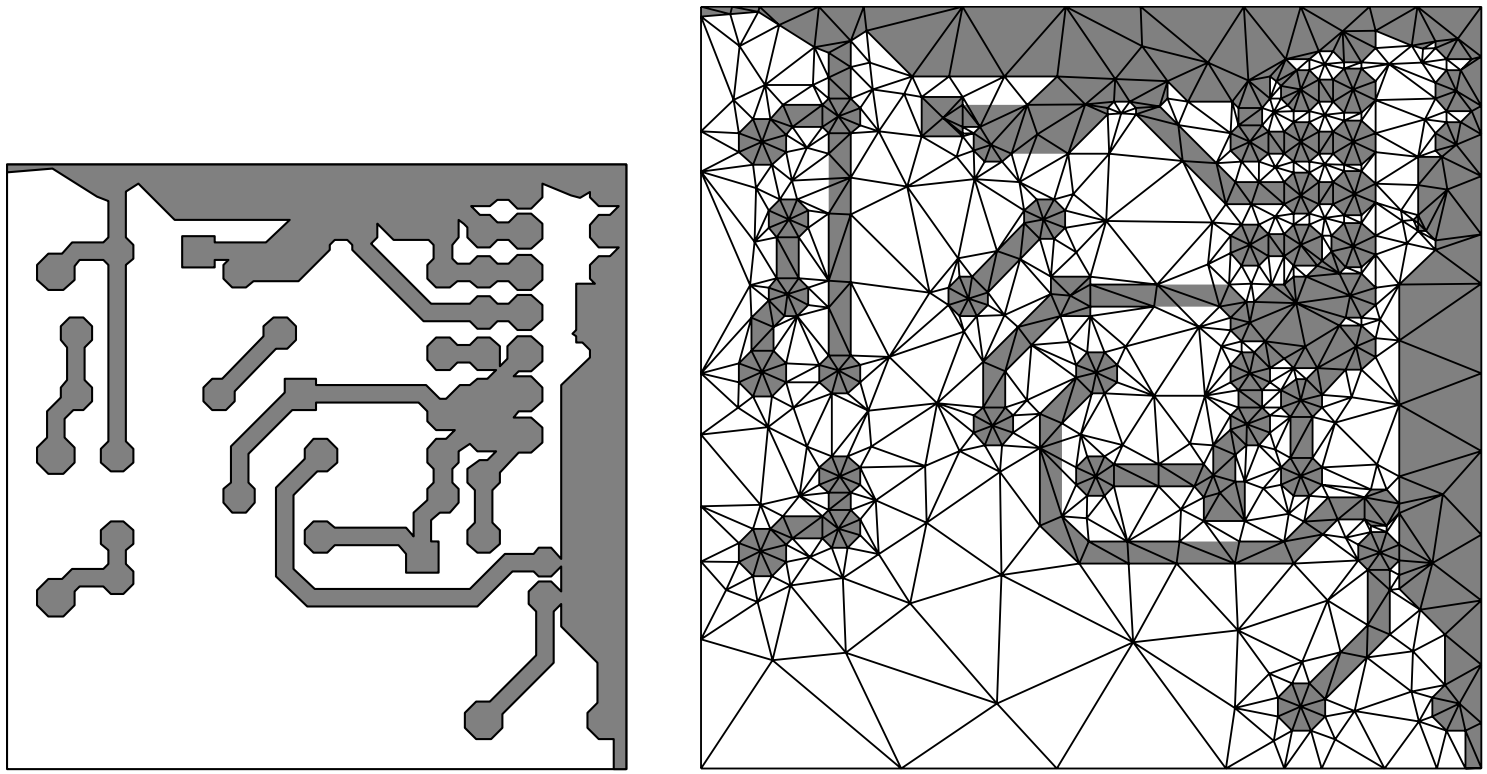
- Structured meshes are easy to construct and of limited use.
- Triangles are flexible and simple.
- Curved domains are usually approximated with polygons.
- 3D is very important, but beyond our scope.

Properties



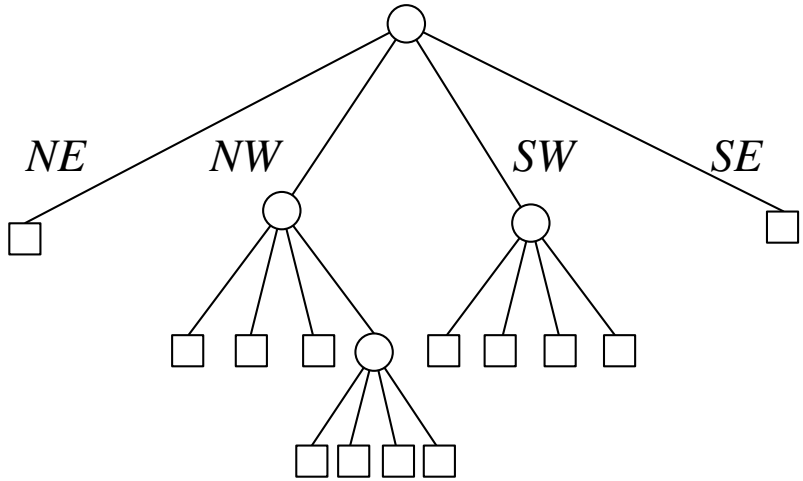
- Conformity: triangles meet solely at shared edges.
- Respects input: the input is a union of triangles.
- Well-shaped: no small or large angles.
- Graded: triangle size adapts to domain complexity.
- Fewest triangles while respecting maximum triangle size.

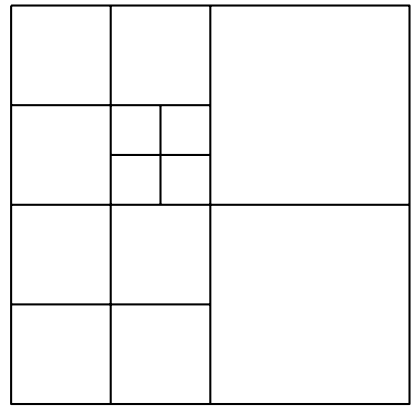
Quadtree Meshing (chapter 14)



- Domain: polygonal region with angles multiples of 45°.
- Grid: unstructured, graded triangle mesh.
- Good algorithm for limited domain.
- Demonstrates interesting quadtree algorithms.

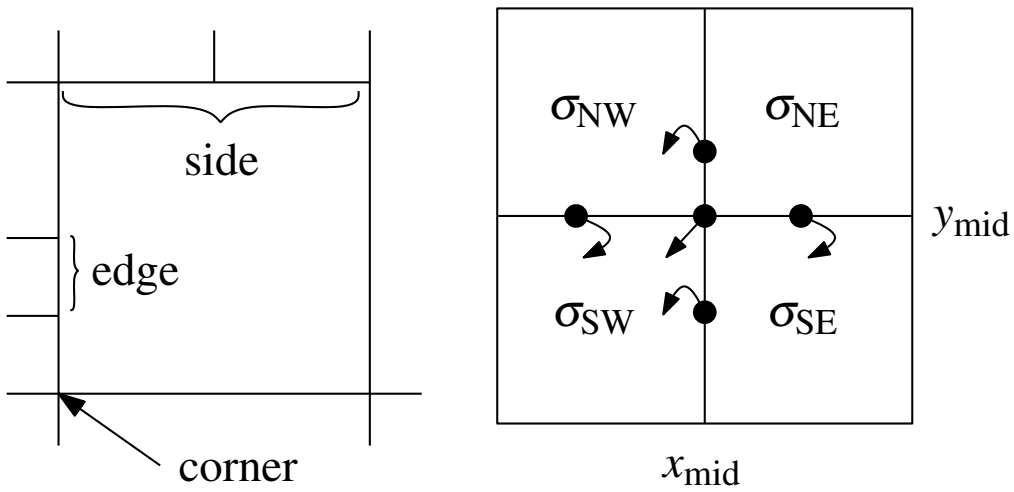
Quadtrees





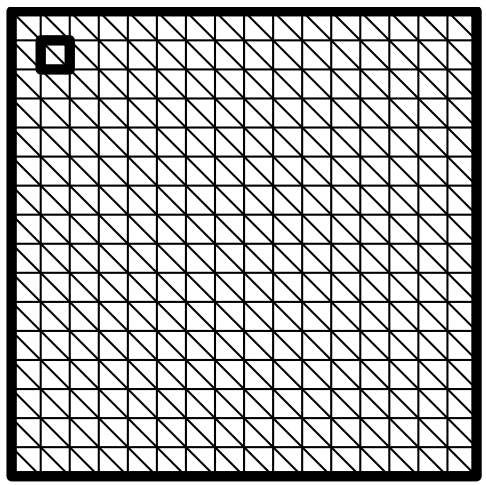
1. Start with a square that contains the elements.
2. Split the square in four at its middle x and y values.
3. Intersect the elements with the squares to obtain fragments.
4. Assign each fragment to its square.
5. Recurse while the square contains too many fragments.

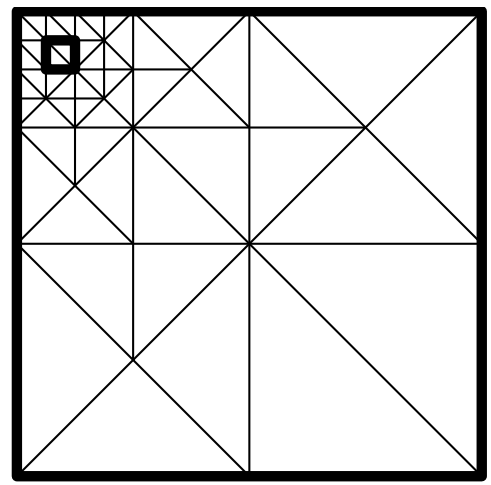
Terminology



- side: a horizontal or vertical side of a square.
- corner: a corner of a square.
- edge: a subset of a side bounded by corners of other squares.
- $\sigma(v)$: the square represented by quadtree node v.
- $\sigma_{NW}(v)$ etc: the squares of the children of v.

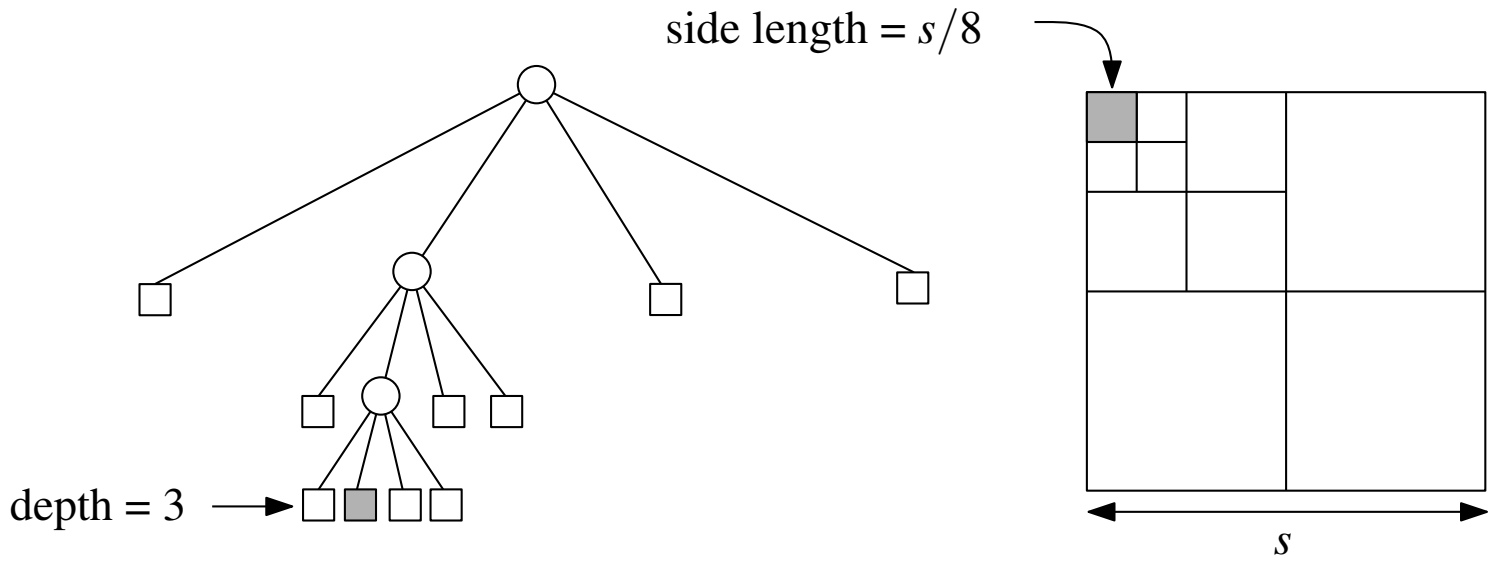
Quadtrees versus Uniform Grids





- Quadtrees store data compactly by adapting to the data.
- Uniform grids are larger, simpler, and sometimes faster.

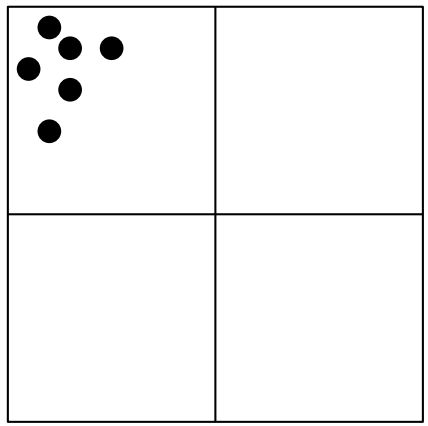
Quadtree Complexity



Lemma 14.1 The depth of a quadtree of points with a minimum separation of c and an s-by-s bounding box is $\log(s/c) + 1.5$.
Proof The squares at depth d have side length $s/2^d$.

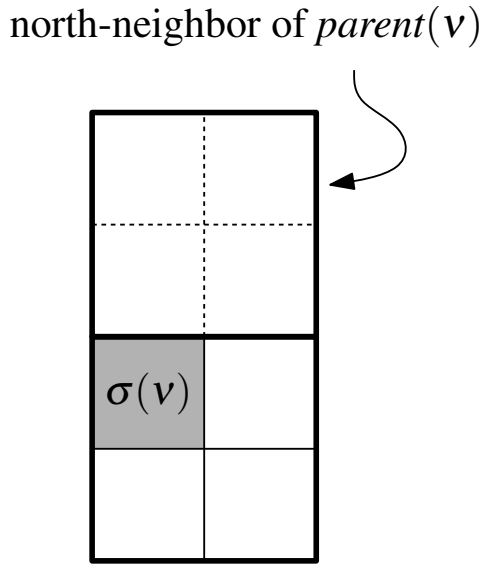
Theorem 14.2 The quadtree construction time for n points is $O(dn)$ with d its depth.
Proof There are n points at each level of the quadtree and the cost per point is constant.

Compressed Quadtrees



- The quadtree size depends on the distribution of the elements.
- When the elements are clustered, the quadtree contains many nodes with three empty children.
- Compressed quadtrees omit these nodes.
- Their depth for n points is $O(\log n)$.
- See the chapter 14 notes for references.

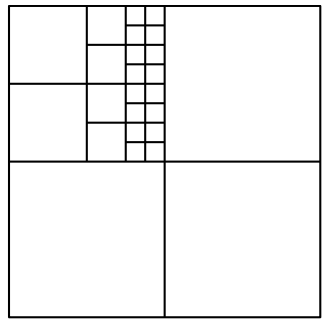
Neighbors



Algorithm NORTHNEIGHBOR$(v, \mathcal{T})$
Input. A node v in a quadtree $\mathcal{T}$.
Output. The deepest node v' whose depth is at most the depth of v such that $\sigma(v')$ is a north-neighbor of $\sigma(v)$, and **nil** if there is no such node.

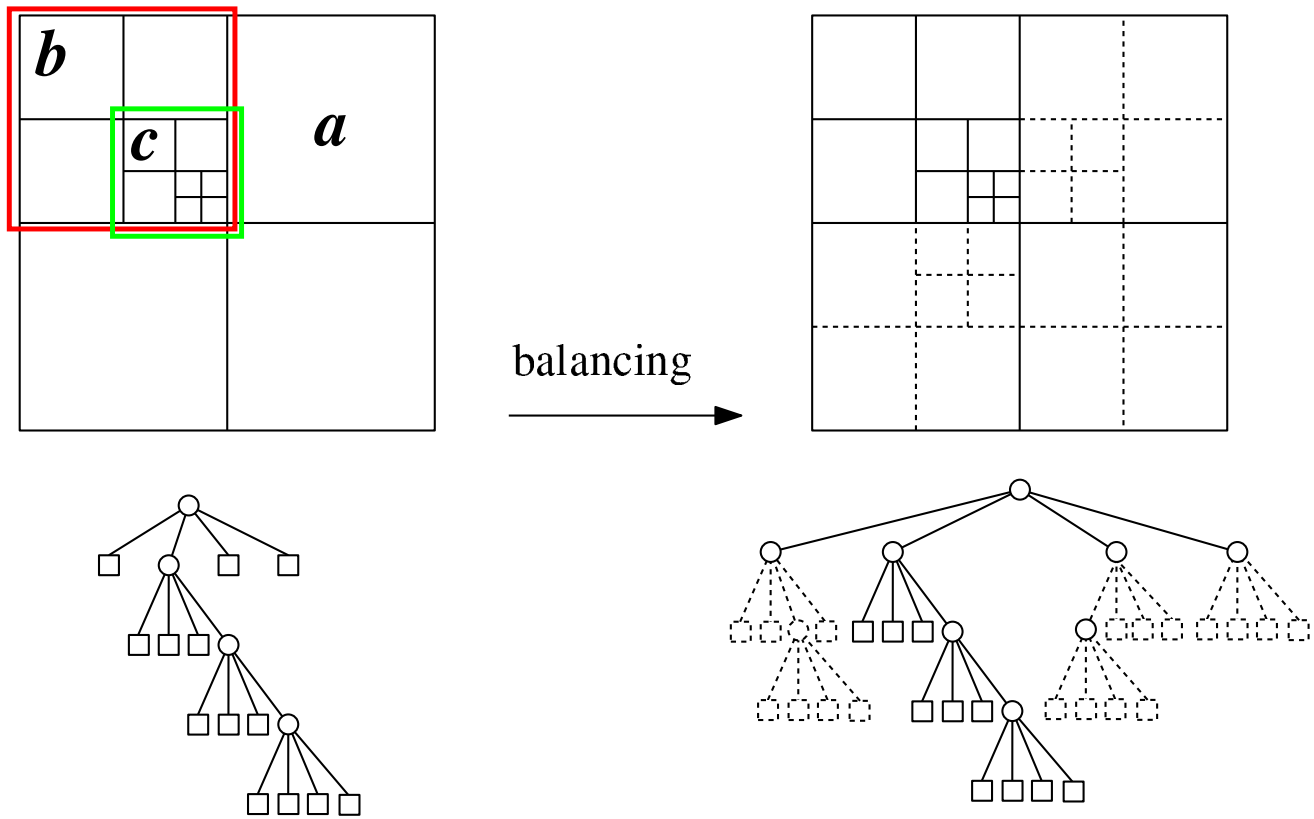
1. **if** $v = root(\mathcal{T})$ **then return nil**
2. **if** $v =$ SW-child of $parent(v)$ **then return** NW-child of $parent(v)$
3. **if** $v =$ SE-child of $parent(v)$ **then return** NE-child of $parent(v)$
4. $\mu \leftarrow$NORTHNEIGHBOR$(parent(v), \mathcal{T})$
5. **if** $\mu =$ **nil or** μ is a leaf
6. **then return** μ
7. **else if** $v =$ NW-child of $parent(v)$
8. **then return** SW-child of μ
9. **else return** SE-child of μ

Balance



- A quadtree for clustered points is unbalanced.
- Large and small squares share edges.
- This is undesirable in meshing and in other applications.
- The tree can be balanced efficiently.

Balancing Algorithm



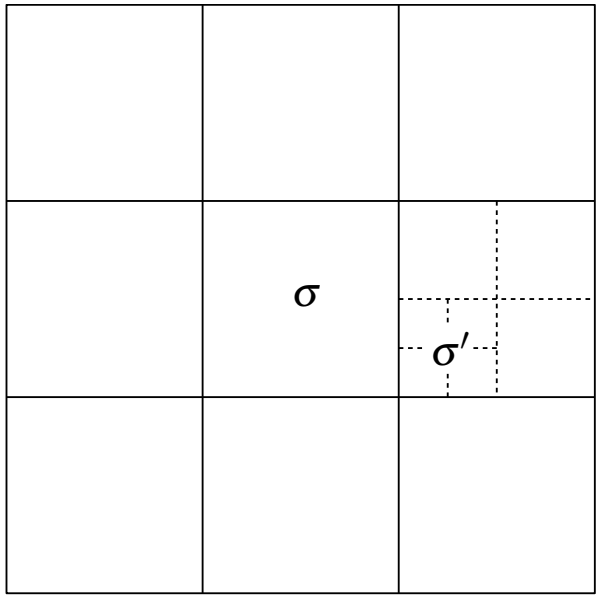
The balancing algorithm splits every leaf that is more than twice as large as one of its neighbors.

A leaf a has a west neighbor b with a northeast or southest child c that is not a leaf. Likewise for the other three sides of a.

Balancing Algorithm

1. Place the leaves of the quadtree in a list l.
2. While l is not empty
   - 2.1 Remove the first leaf μ from l.
   - 2.2 If μ needs to be split
     - 2.2.1 Split μ and move its point to the containing child.
     - 2.2.2 Add the children of μ and the neighbors of μ to l.

Balancing Complexity

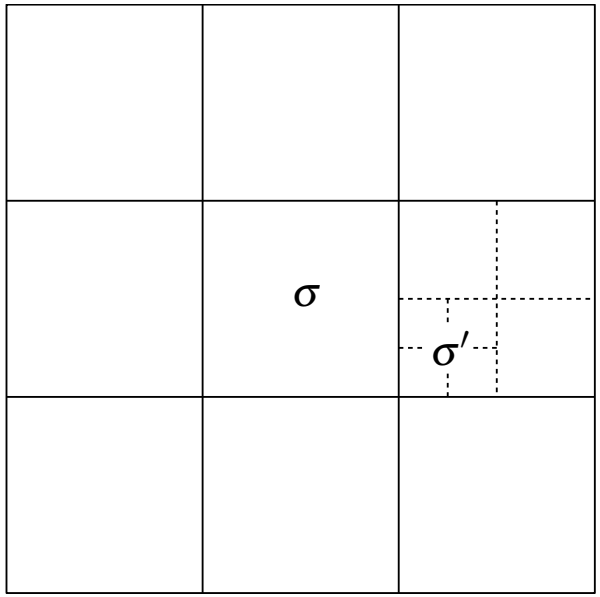


Theorem 14.4 A quadtree T with m nodes and depth d yields a balanced quadtree with $O(m)$ nodes in $O(dm)$ time.

Proof

- Claim: If a square is split, it has a same-size neighbor in T.
- There are m squares in T each with at most 8 neighbors.
- A split creates 4 nodes.
- There are at most $32m$ nodes in the balanced quadtree.
- The cost of creating a node is $O(d)$ for neighbor finding.

Proof of Claim by Contradiction



Let σ be the smallest square that is split such that all its same-size neighbors are new (result from splits).

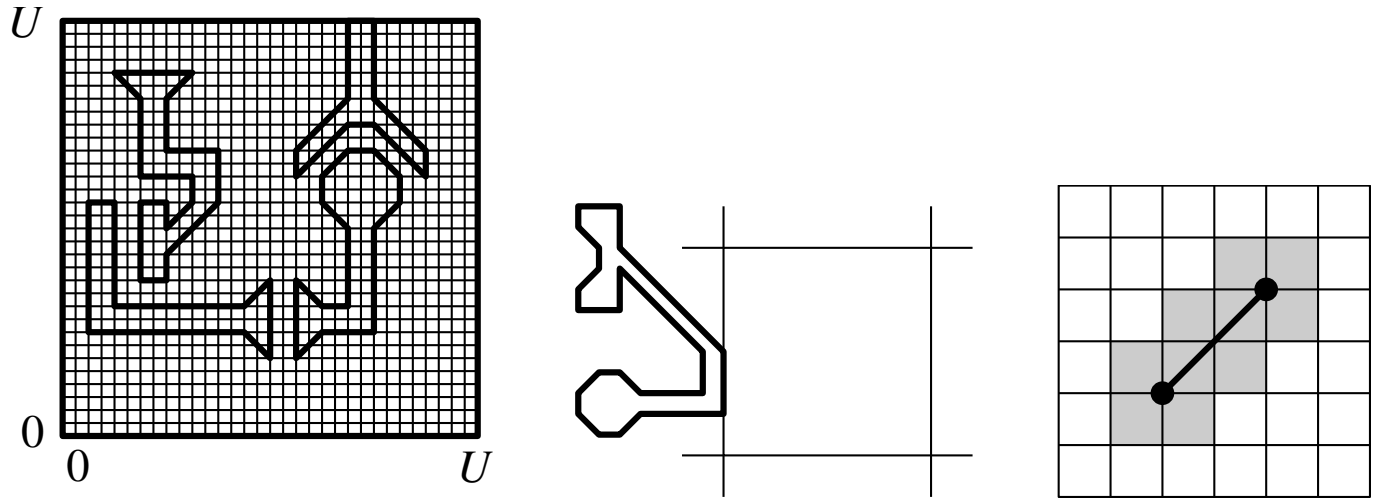
σ has a neighbor μ less than half its size.

Let σ' be the square that contains μ and is half the size of σ. σ' is new because it is contained in a same-size neighbor of σ.

The same-size neighbors of σ' are new because they are contained in σ or in the same-size neighbors of σ.

This contradicts the definition of σ.

Quadtree Meshing



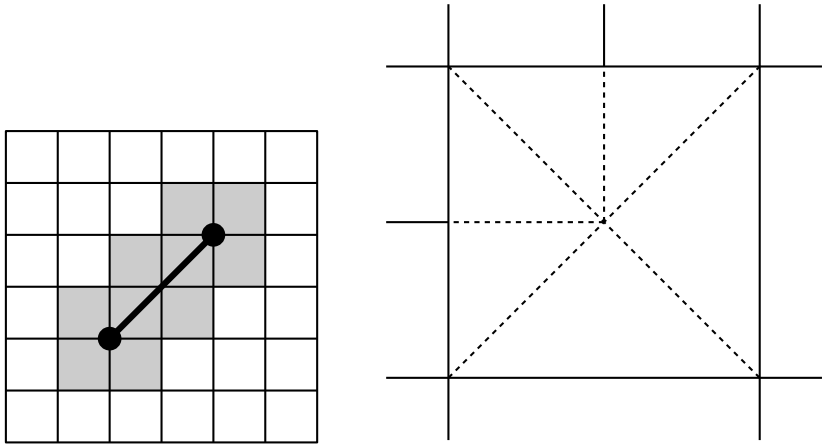
domain need to split done splitting

The input is a subdivision of a square $U = [0, 2^j] \times [0, 2^j]$. The vertices have integer coordinates. The edges have slopes of ± 1.

Algorithm

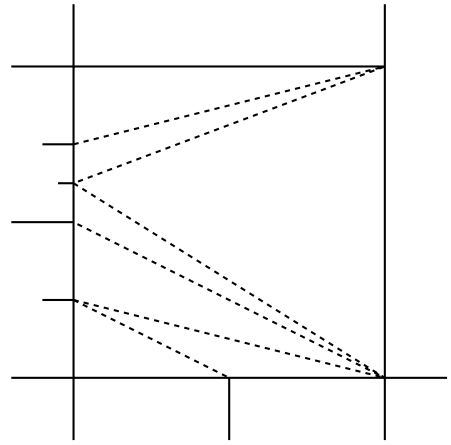
1. Construct a quadtree whose leaf squares are unit size or are disjoint from the input.
2. Balance the quadtree.
3. Split the leaf squares into triangles.

Square Triangulation



diagonal

good shape



bad shape

Cases for splitting a leaf into triangles.

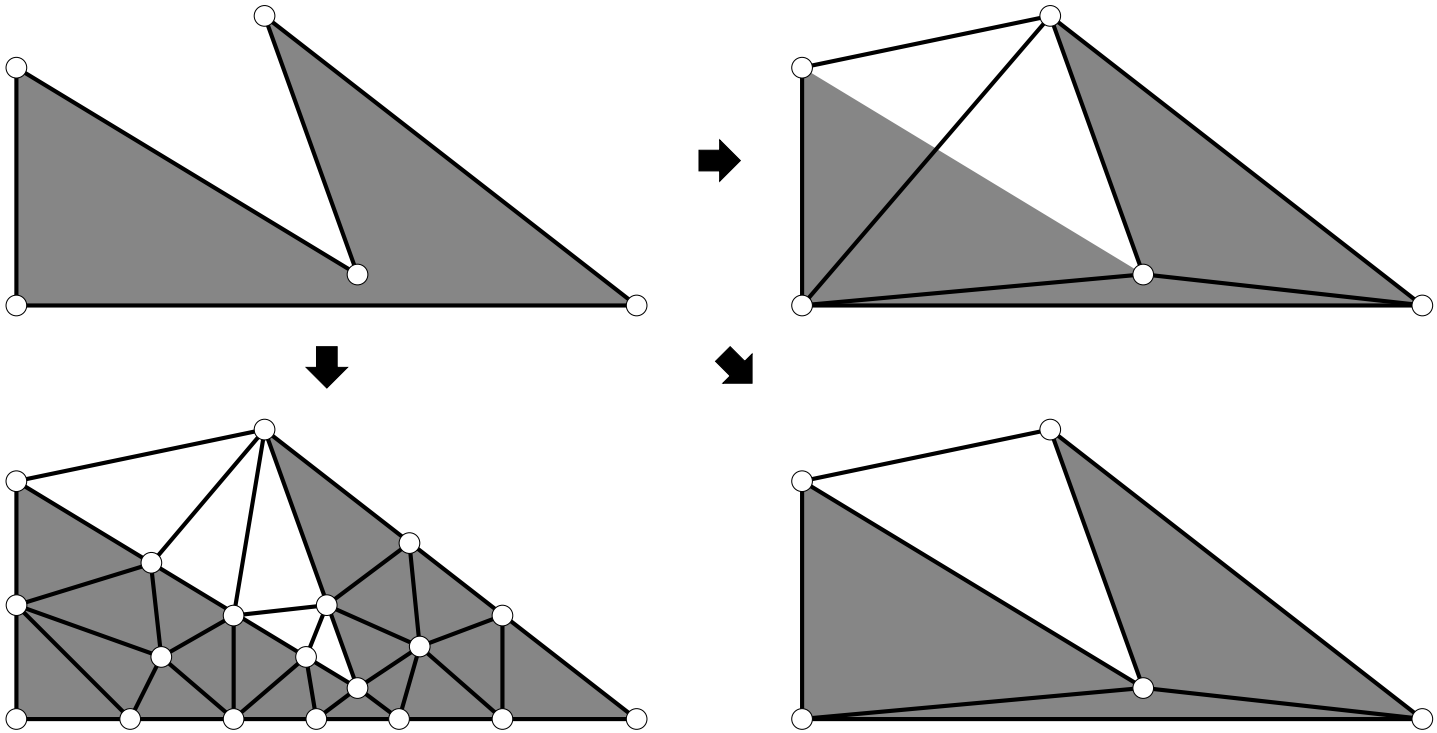
1. A diagonal is an edge.
   Split along the diagonal.
2. The interiors of the sides are free of vertices.
   Split along either diagonal.
3. The interiors of some sides contain vertices.
   Connect the side vertices to a new vertex at the center.
   The triangle shapes are good because of balancing.

Meshing Complexity

Theorem 14.5 The mesh has $O(P \log U)$ triangles with P the sum of the input edge lengths. The construction time is $O(P \log^2 U)$.

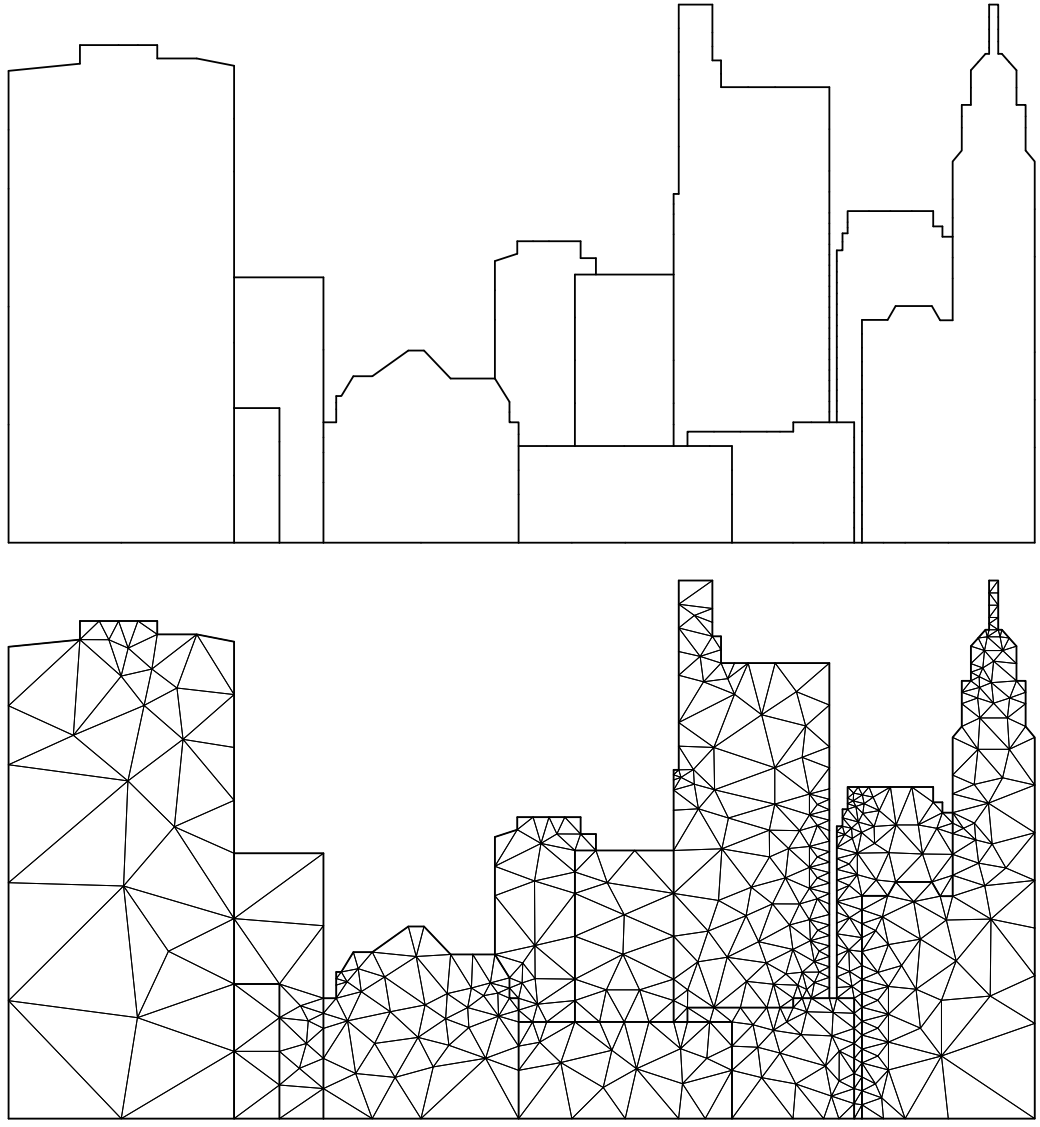
The proof is straightforward.

Delaunay Refinement



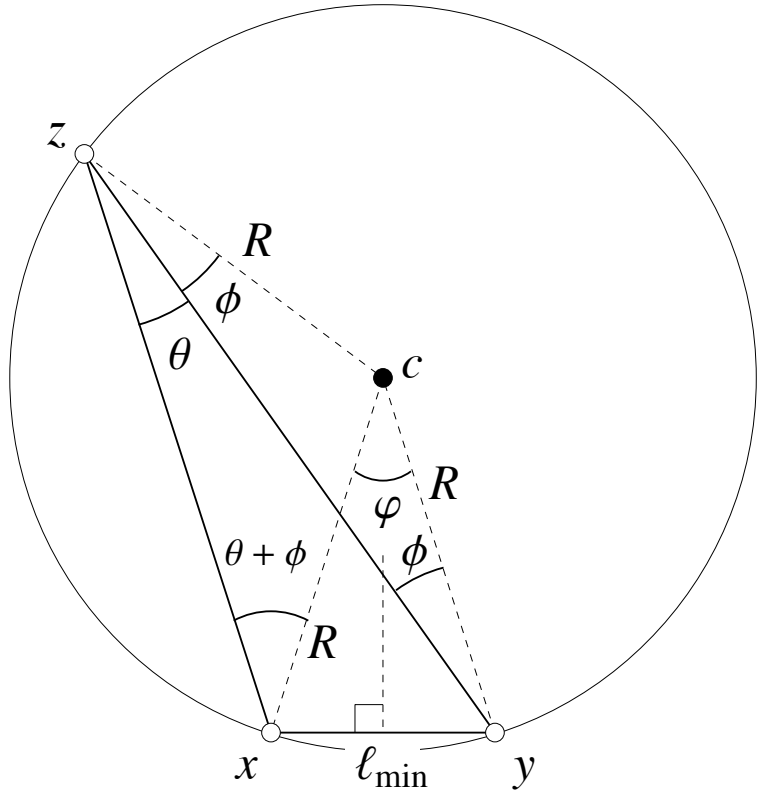
- Goal: construct a Steiner Delaunay triangulation of a domain.
- We will discuss Ruppert's algorithm for a polygonal domain.
- Cheng provides more information on Delaunay refinement.

Example Mesh



The input is the bounded faces of a subdivision with linear edges.

Quality Metric



- The triangle quality metric is $\rho = R/\ell_{\min}$ with R the radius of the circumcircle and $\ell_{\min}$ the length of the shortest edge.
- The smallest angle is $\theta = \arcsin(1/2\rho)$.
  - $\angle ycz = 180 - 2\phi$ because ycz is isoceles.
  - $\angle xcz = 180 - 2(\theta + \phi)$ because xcz is isoceles.
  - $\varphi = \angle ycx = \angle ycz - \angle xcz = 2\theta$
  - $\sin\theta = \sin\varphi/2 = \ell_{\min}/2R = 1/2\rho$

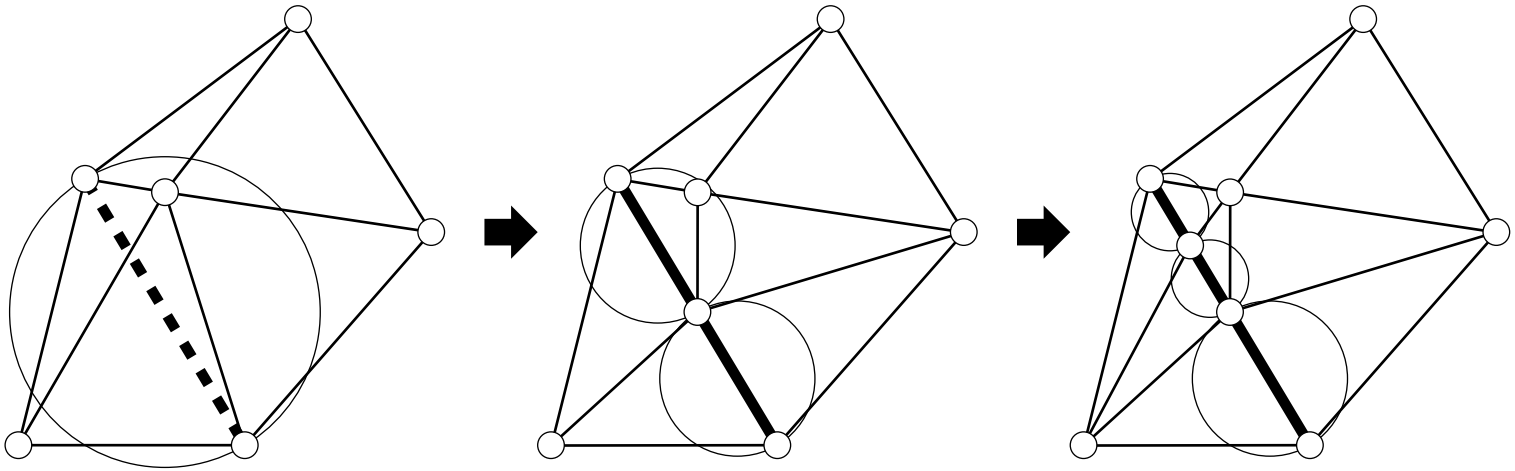
Quality of Ruppert's Algorithm

- Ruppert's algorithm splits triangles with $\rho > \overline{\rho}$.
- It provably converges for $\overline{\rho} \geq \sqrt{2}$.
- This yields minimum and maximum angles of 21° and 138°.
- Angles of 33° and 134° are usually achievable in practice.

Ruppert's Algorithm

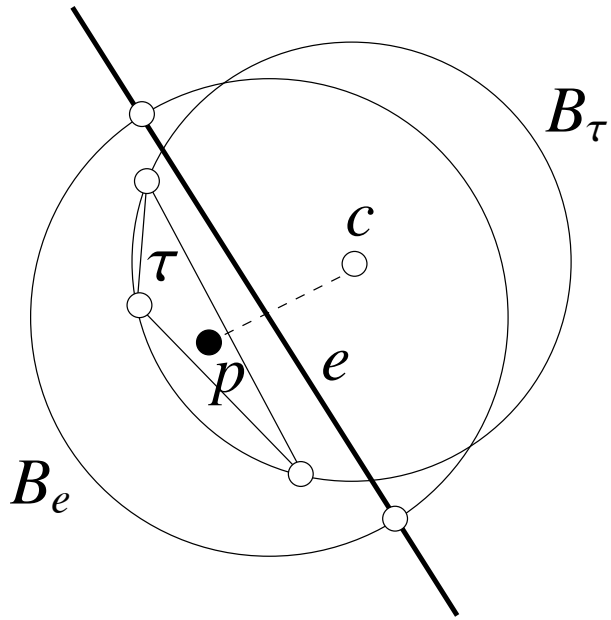
1. Construct the Delaunay triangulation of the vertices.
2. Split the domain edges that are missing from the triangulation.
3. Split the bad triangles in the domain.

Step 2: Split Encroached Edges



- A vertex in the diametric circle of an edge *encroaches* on it.
- If vertices a and b have an empty circle, ab is Delaunay.
- Step 2 bisects the encroached input edges until none remain.
- The resulting Delaunay triangulation contains the subedges.

Circumcenters in Domain

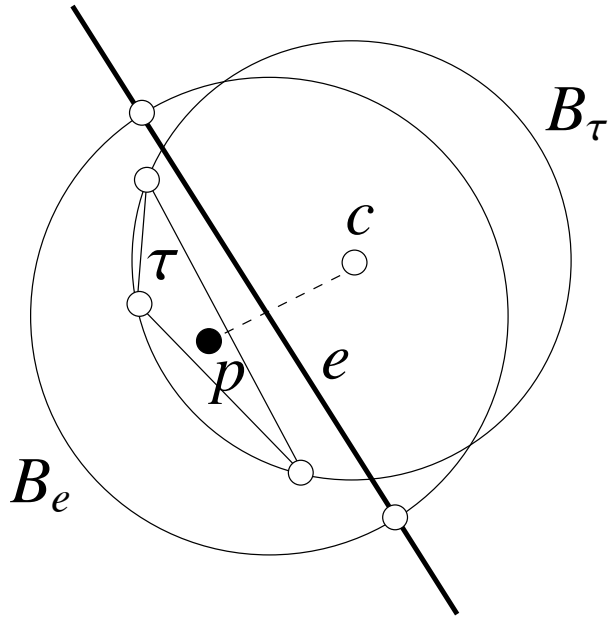


Proposition 6.2 If no edge is encroached, the circumcenter of every triangle is in the domain D.

Proof Suppose a triangle τ has a circumcenter c outside D. Let p be a point in τ. An edge e on the boundary of D intersects pc.

The interior of the circumdisk B_τ of τ intersects e, since pc is in B_τ, but excludes its endpoints, since τ is Delaunay.

Circumcenters in Domain (continued)

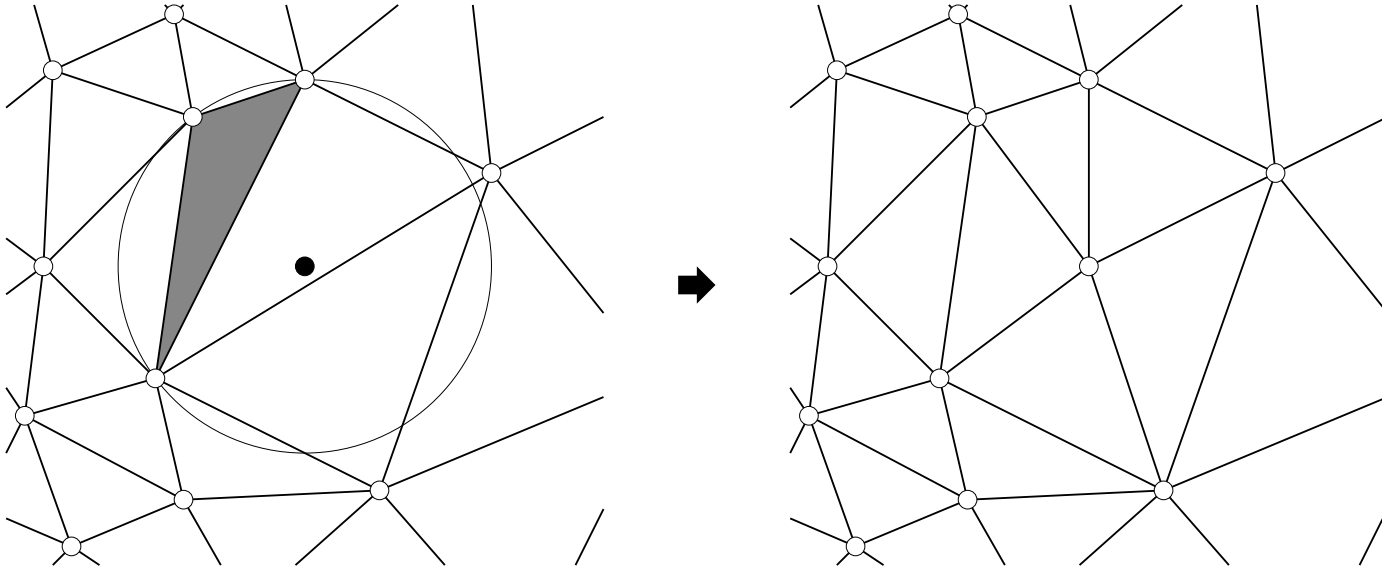


Let H be the half plane of e that contains p. Since B_τ is empty, e splits it into two regions with $\tau \subset B_\tau \cap H \subset D$.

Let B_e be the diametric disk of e. Since c is outside H, $B_\tau \cap H \subset B_e$.

Hence, $\tau \subset B_e$. At most two vertices are incident on e, so one vertex encroaches on it.

Step 3: Split Bad Triangles



- A triangle with $\rho > \overline{\rho}$ (or a large triangle) is bad.
- Step 2 ensures that the circumcenter o is in the domain.
- If o encroaches on an edge, the edge is split.
- Otherwise, o is added to the vertex set.
- The bad triangle vanishes because its circumcircle contains o.
- The new edges are longer than the shortest edge length $\ell_{\min}$. Their length is at least the circumradius $R = \rho\ell_{\min} > \ell_{\min}$.

Meshing a Hollow Rectangle

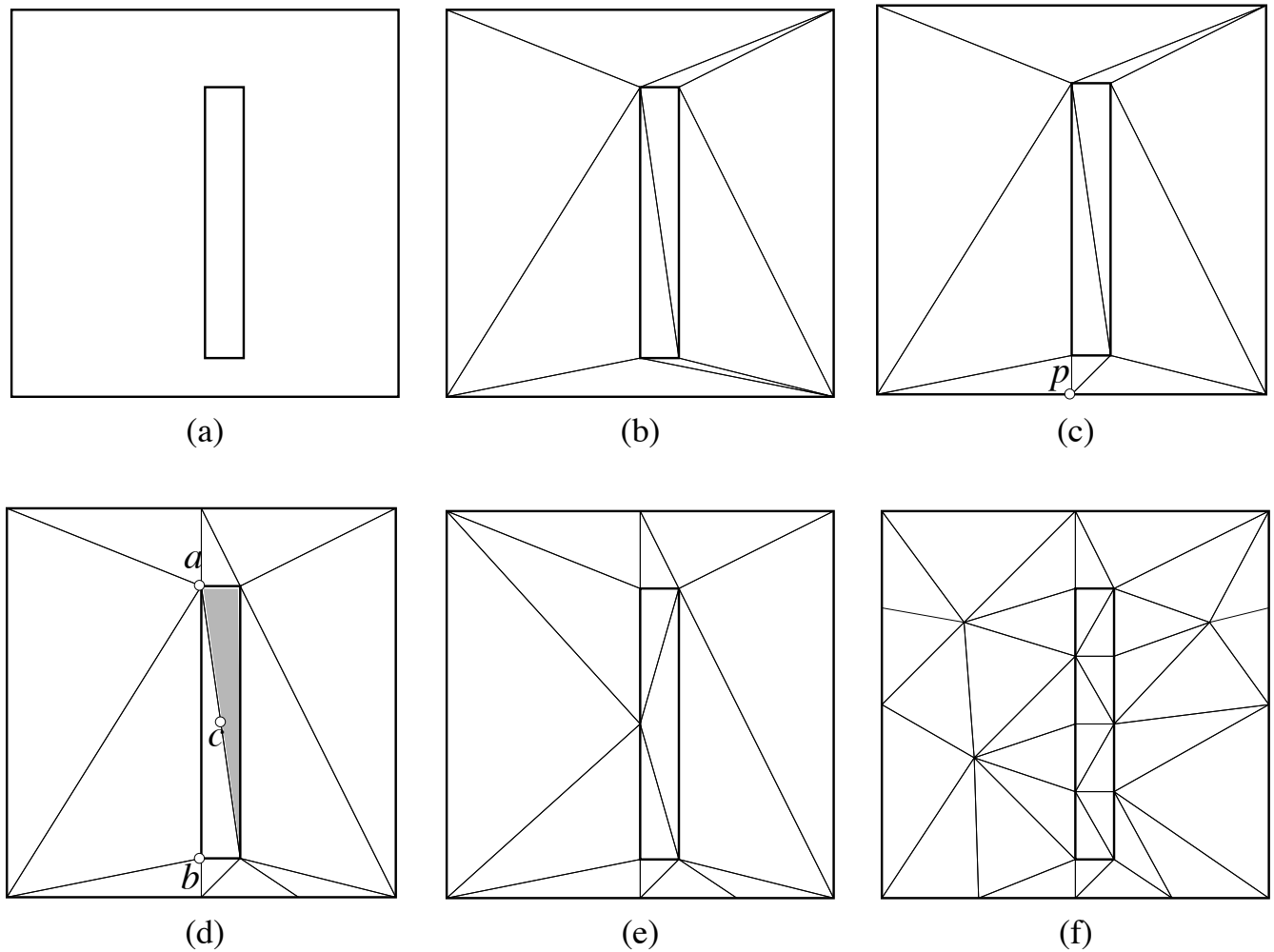
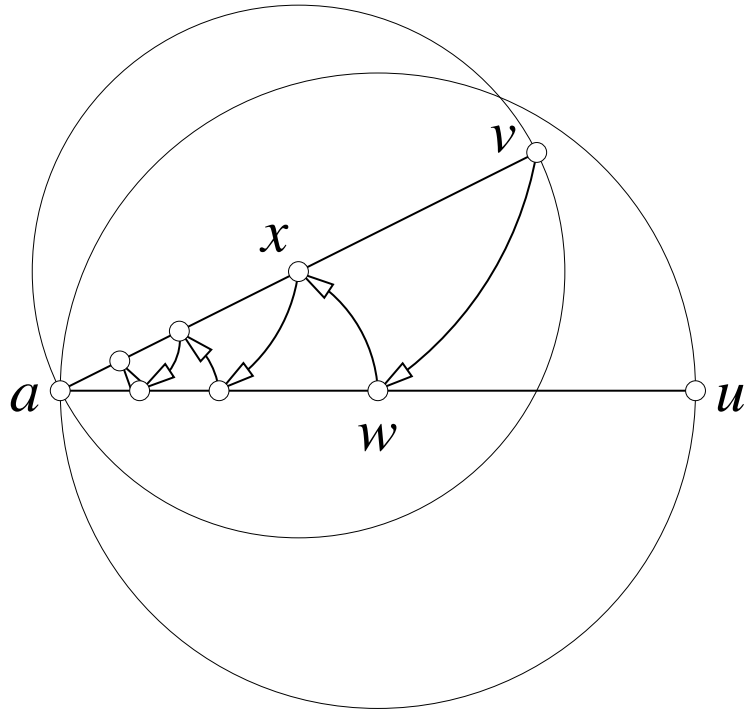


Figure 6.4: The algorithm DelTriPLC in action. (a) The input PLC. (b) The Delaunay triangulation of the input vertices. (c) The bottom edge is encroached, so it is split at its midpoint p. (d) After all the encroached subsegments are split, the shaded triangle has poor quality, so the algorithm considers inserting its circumcenter c. (e) Because c encroaches upon the segment ab, the algorithm does not insert c; instead it splits ab. (f) The final mesh.

Small Angles

- Ruppert's original algorithm forbids acute interior angles.
- Convergence fails because of ping-pong encroachment and seditious edges.
- These are handled by two simple extensions.
- The maximal angle is as before.
- The minimum is as before, except at small input angles.

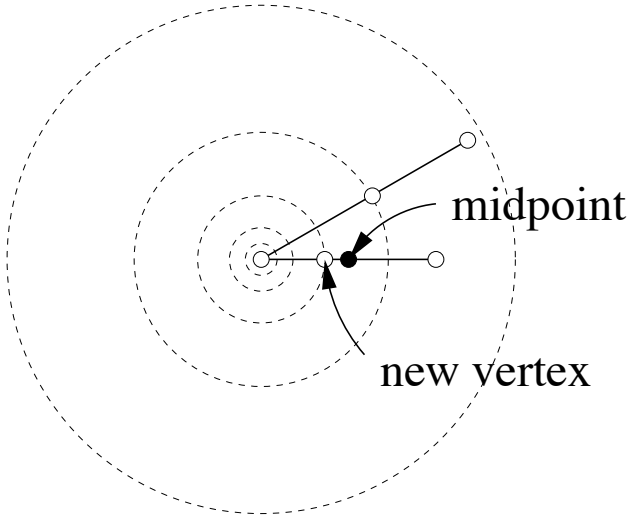
Ping-Pong Encroachment

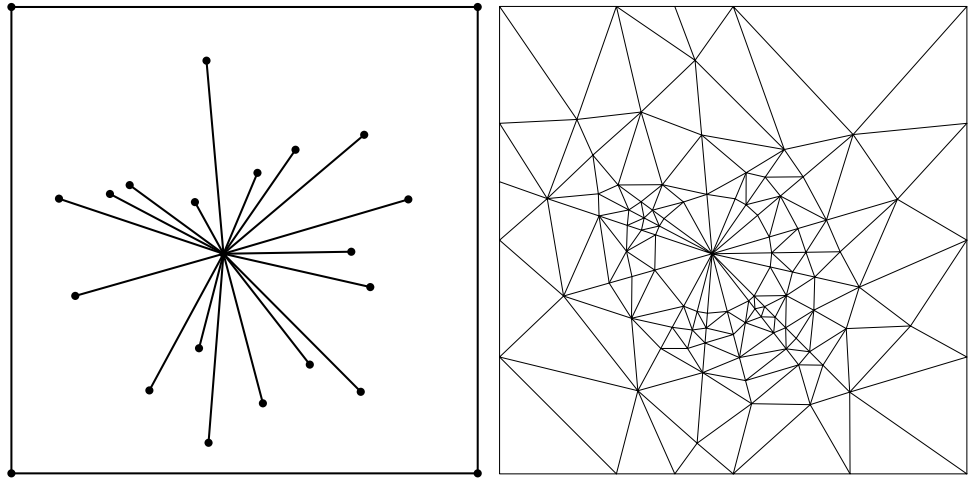


1. v encroaches on au.
2. au is split at w.
3. w encroaches on av.
4. av is split at x.
5. x encroaches on aw.

The encroachment cycle continues forever.

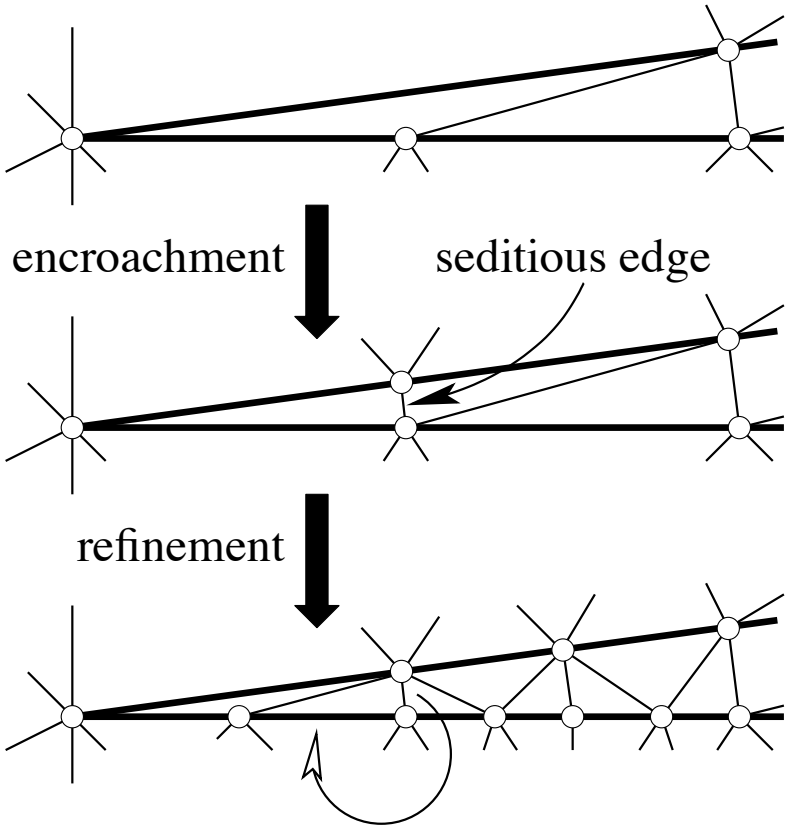
Solution: Concentric Circles

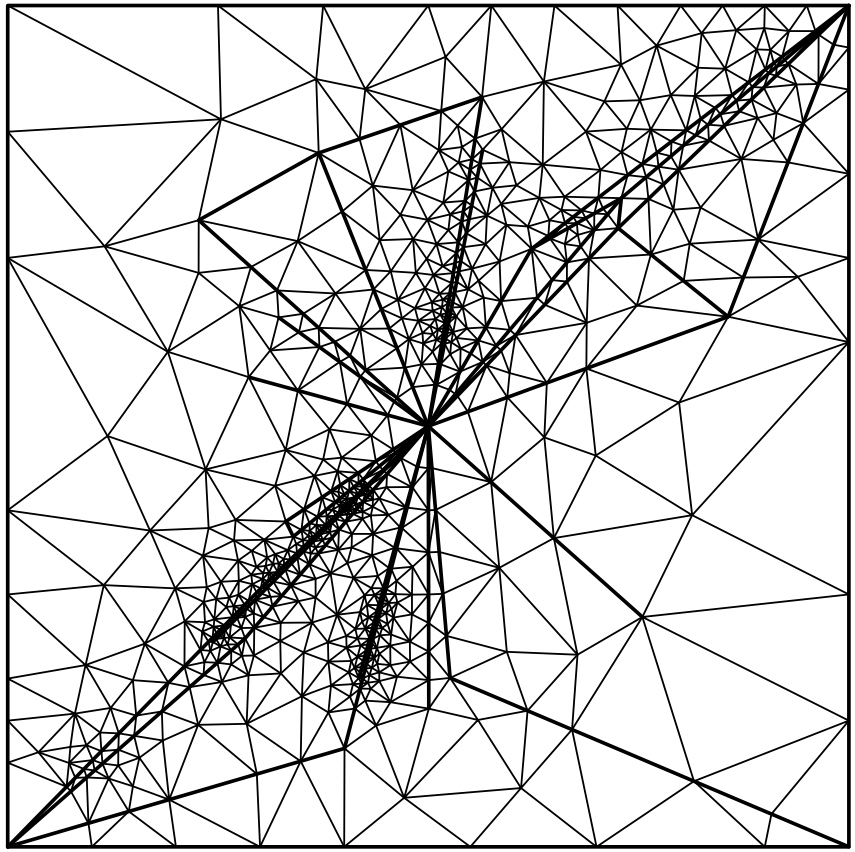




- Edges au and av form an acute angle and v encroaches on au.
- au is split at its intersection with a circle centered at a.
- The radius is $r = 2^i$ with i the integer for which the split point is as close as possible to the midpoint of au.
- If u is a circle point, au is split at its midpoint.
- au and av with u and v on the same circle do not encroach.

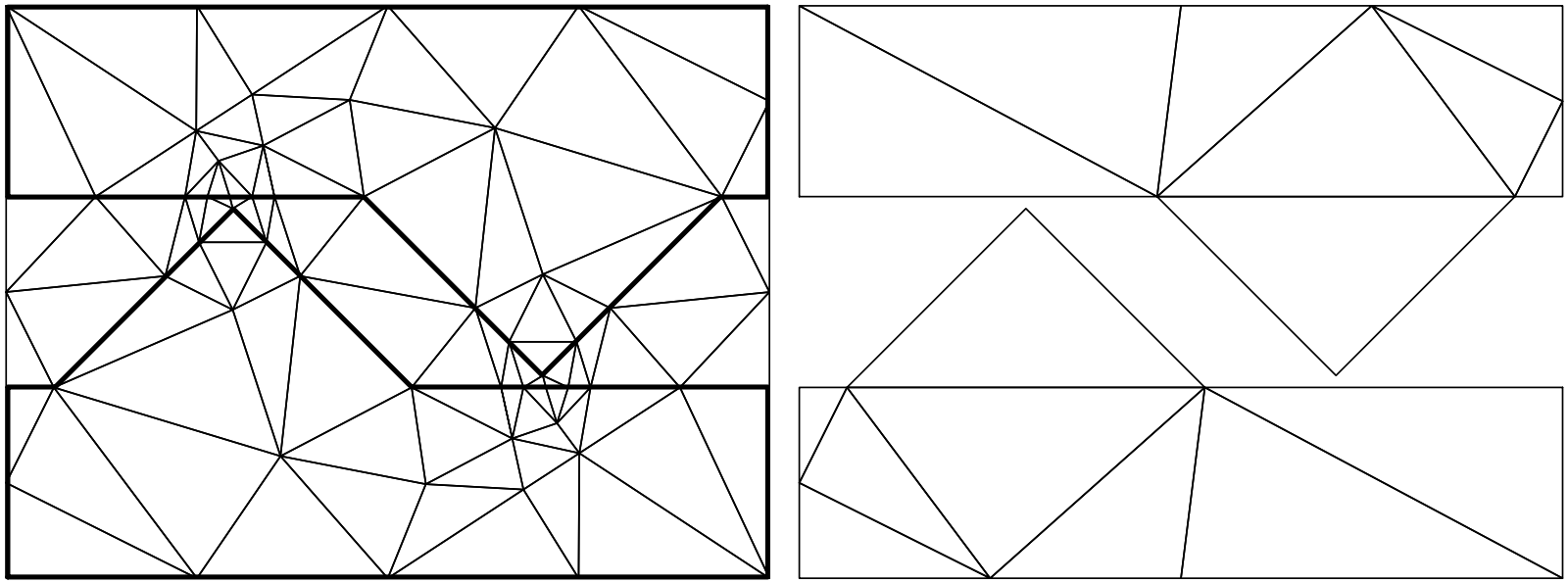
Seditious Edges





- An edge uv is seditious when $\angle uav < 60°$, u and v are on the same circle centered at a, and they are midpoints.
- It causes an edge split that leads to another seditious edge.
- Fix: do not split a triangle whose shortest edge is seditious.

Constrained Delaunay Triangulation



- Step 1 can construct a constrained Delaunay triangulation.
- Triangles are not created outside the domain.
- This saves memory and avoids the domain test in step 3.
- Encroached input edges are in the triangulation.
- This avoids triangle location in step 2.
- Close, unconnected elements no longer force mesh refinement.

3D Delaunay Refinement

- Ruppert's original algorithm generalizes to 3D.
  - Delaunay triangulation is fast, but worst case $O(n^2)$.
  - Splitting encroached edges is a bit more complicated.
  - Encroached facets must also be split.
  - The resulting Delaunay triangulation conforms to the input.
  - Splitting bad tetrahedrons is like splitting bad triangles in 2D.
- The extension to acute angles is much more complicated.
- Constrained Delaunay triangulation maintains its advantages.
- The quality metric generalizes, but misses sliver tetrahedrons.