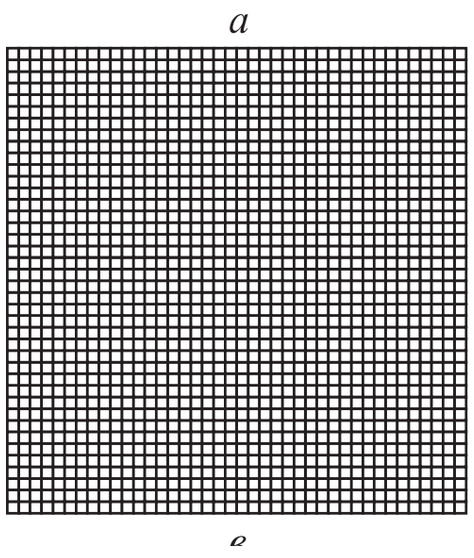
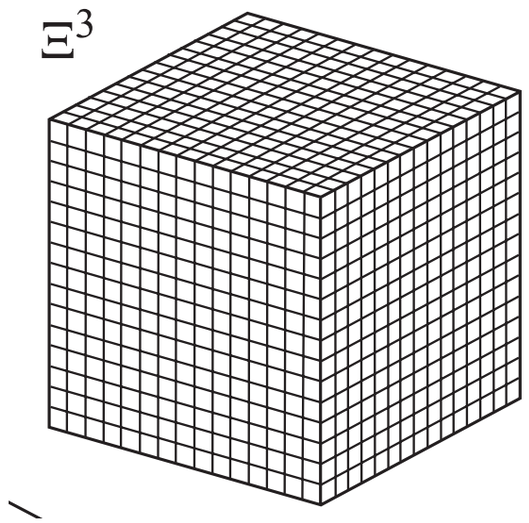
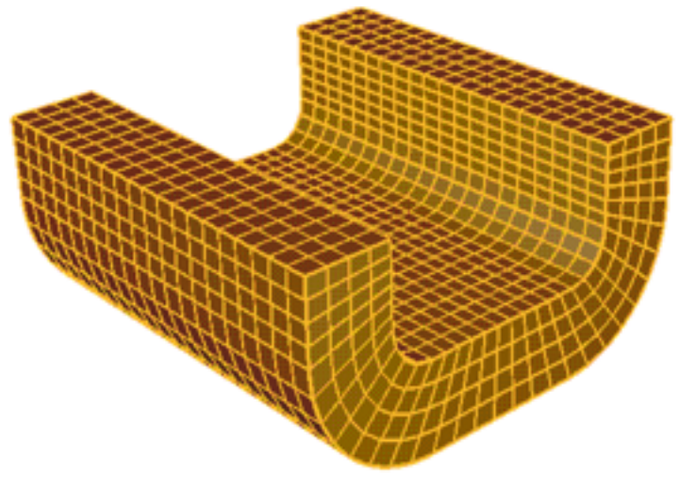
Overview of Unstructured Mesh Generation Methods

Structured Meshes

- local mesh points and cells do not depend on their position but are defined by a general rule.
- Lead to very efficient algorithms and storage.
- High quality for numerical methods.

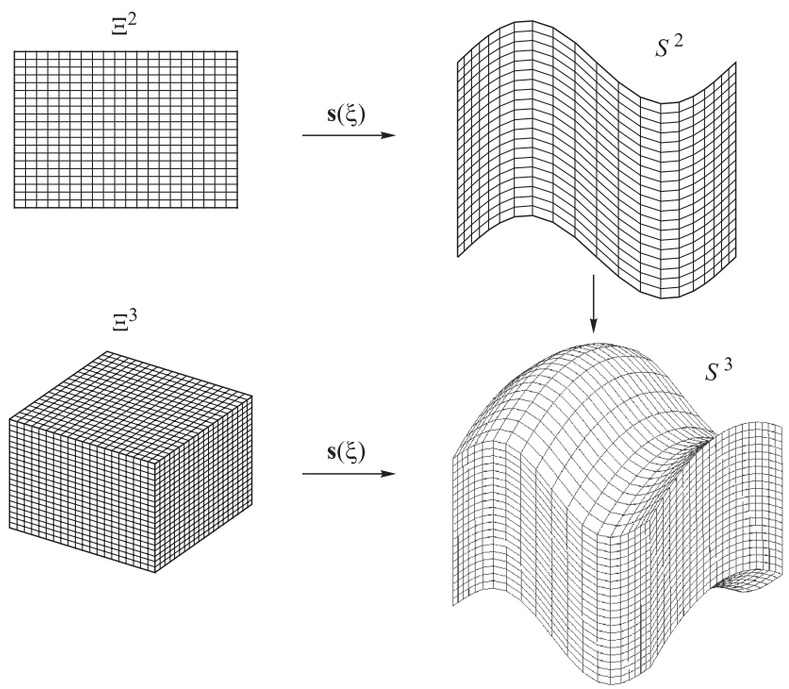






Structured Grid Generation Methods

- Conformal mapping
- Transfinite interpolation
- Solving PDEs — Elliptic, Parabolic/Hyperbolic
- Multi-block structured methods



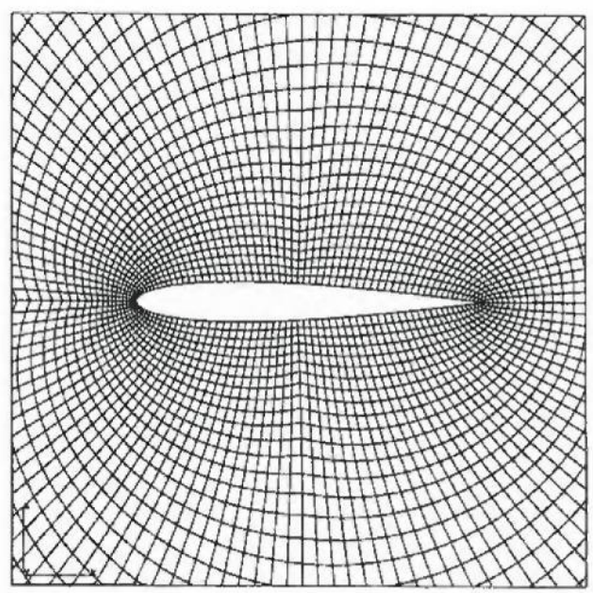


FIGURE 4.6 Laplace grid. Grid control map is the identity map.

Limitations of Structured Meshes

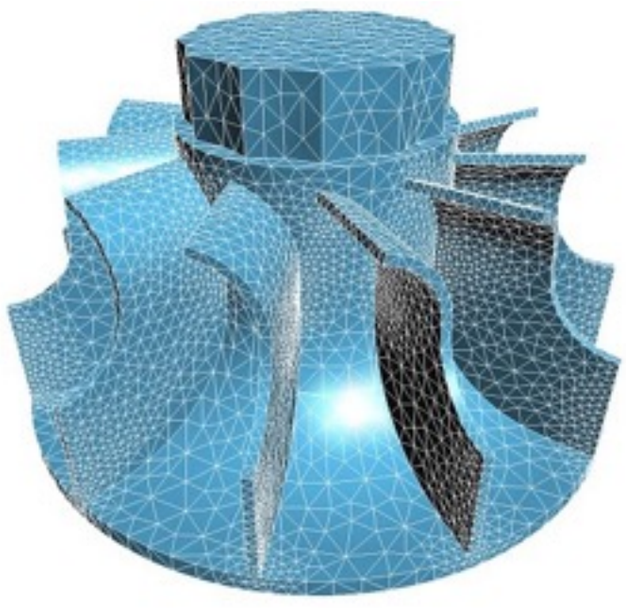
- Only suitable for domain with regular shapes.
- The block-structured (or hybrid) mesh is formed by a number of structured meshes combined in an unstructured pattern.
- Mesh adaptation is difficult.

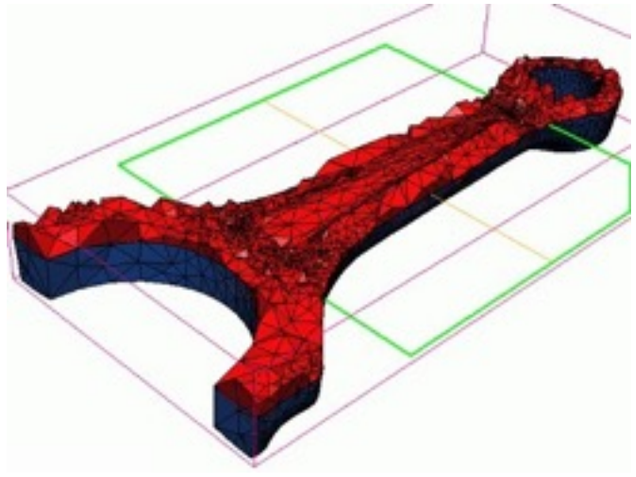
Unstructured Meshes

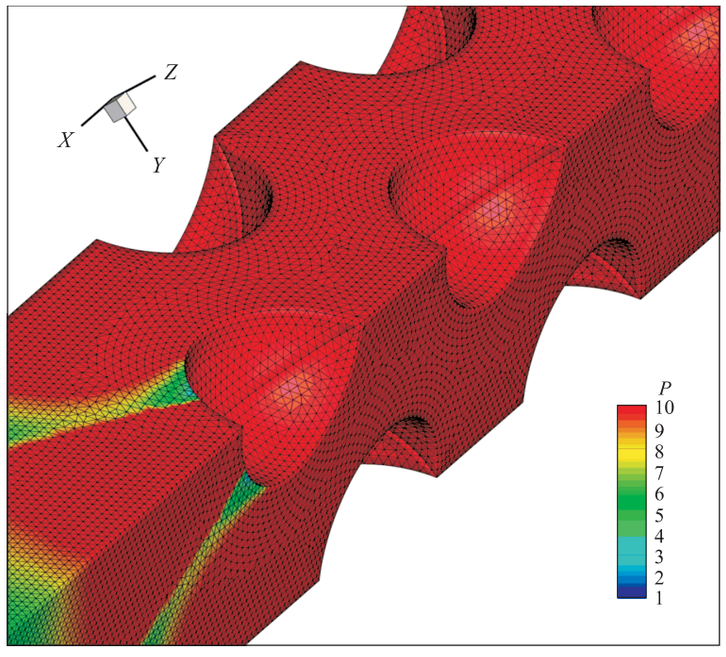
- An unstructured mesh is one which vertices have arbitrary varying local neighbours.

- Such mesh are more easy to fit in complex domain.

- Such mesh are easy for mesh adaptation.







- Quadtree-Octree based methods
- Advancing-front methods
- Delaunay-based methods

Quadtree-Octree based Methods

Introduction

- It is one of the spatial tree data structures used widely in geometric computation.

- Quadtree-Octree based methods for mesh generation has been a topic of research since 1980s, pioneered by [Thacker-1980, Yerry,Shephard-1983].

- This type of methods is fully automatic for arbitrary complicated domains.

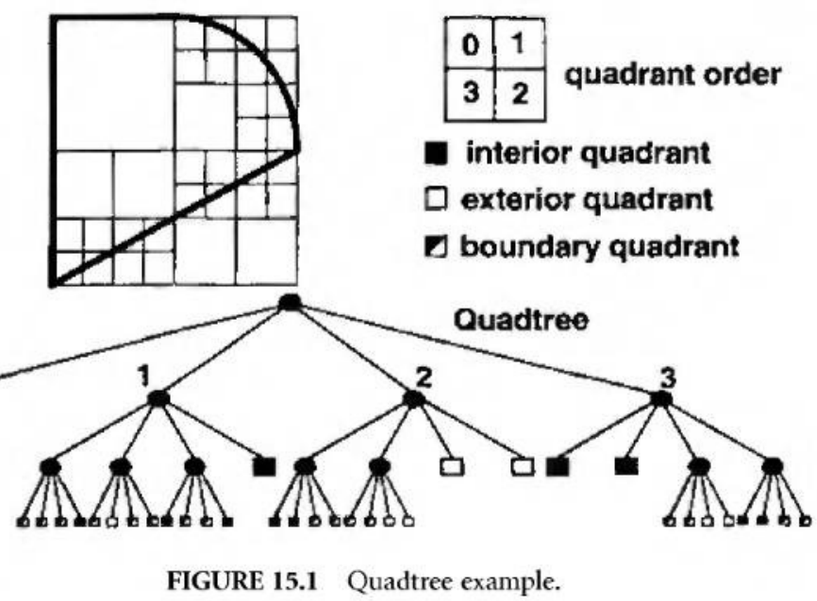


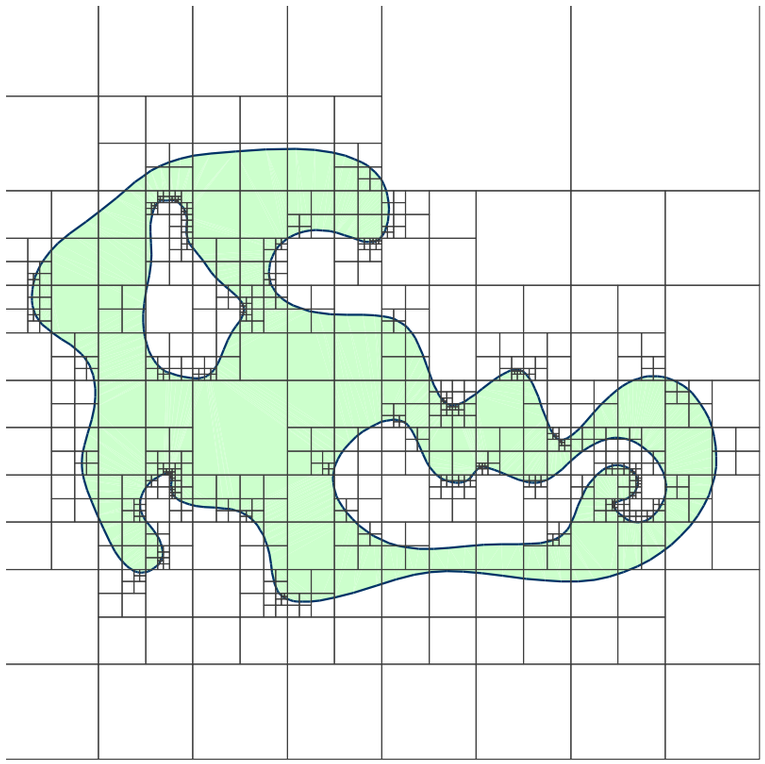
FIGURE 15.1 Quadtree example.

[HandbookGG, Chap 15, Shephard et al]

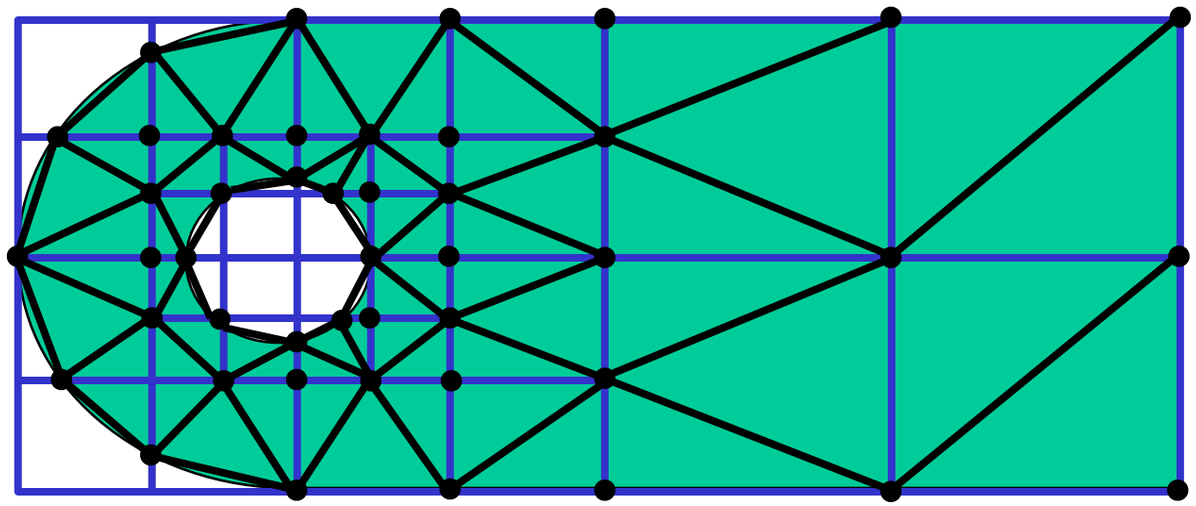
General Principles

- The mesh domain is first approximate by a union of disjoint and viable sized cells. These cells are obtained by recursive refinement with a set of subdivision rules and stopping criteria.

- Each cell is decomposed into a set of mesh elements by a set of templates.



Picture of Persson's Mesh Course 2006



- Define intial bounding box (*root* of quadtree)
- Recursively break into 4 *leaves* per *root* to resolve geometry
- Find intersections of leaves with geometry boundary
- Mesh each *leaf* using corners, side nodes and intersections with geometry
- Delete Outside
- (Yerry and Shephard, 84), (Shepherd and Georges, 91)

Detail: Tree Construction

The tree construction is an iterative procedure that builds the covering tree of the domain

A. Selection of a boundary entity, in ascending order (point, edges, faces).

B. Identification of the cell in the current tree that contains this entity.

C. Analysis of the cell, if it already contains an entity of the same dimension then refine the cell (with 4 equally sized cells), otherwise, back to (B)

D. Insertion of the entity in the cell and back to (A).

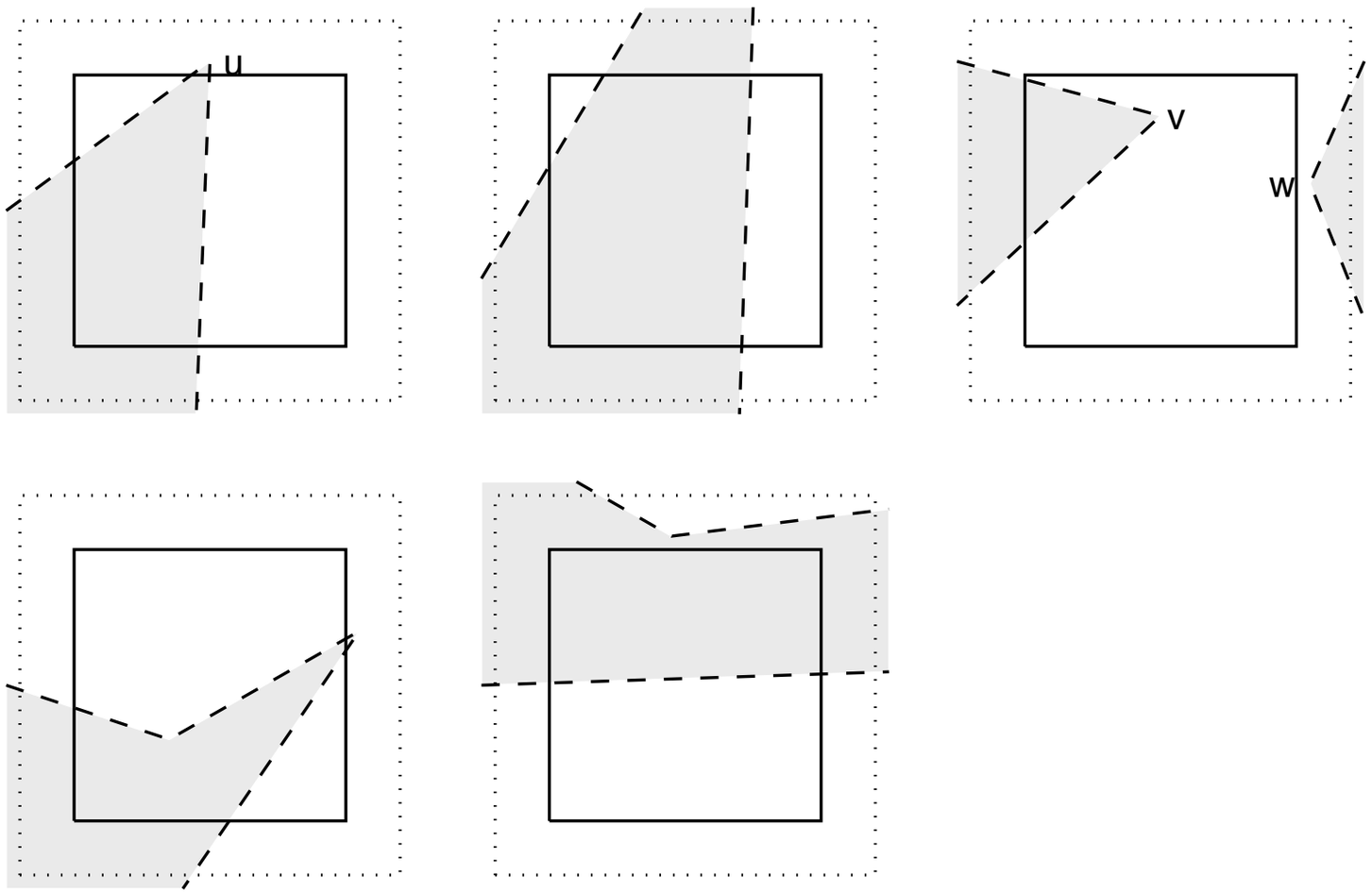


FIG. 5.1. *Examples of crowdedness: solid lines indicate boxes, dotted lines indicate* $\mathrm{ex}(B)$ *for these boxes, dashed lines indicate the boundary of* P*, and shading represents the interior of* P*. Suppose we are in the separation stage of the phase* 0 *in the case* $d = 2$*. All boxes in the top row are uncrowded. The first box would be placed into* $O_{\boldsymbol{u}}$*. The second box would be placed into* I_1*. The third box in the top row must be duplicated, and then one duplicate would go into* $O_{\boldsymbol{v}}$ *and the other into* $O_{\boldsymbol{w}}$*. Both boxes in the bottom row are crowded and must be split.*

Cell refinement examples [Mitschell & Vavasis 2000]

Detail: Tree Balancing

- The tree resulting from the tree construction procedure can be rather unbalanced. It is not efficient for future mesh generation:
  - The size variation between neighbour cells will increase the searching cost.
  - It complicates the mesh template design
- The **2-to-1 Rule**: A tree subdivision is balanced if every side (edge) of a terminal cell contains at most one corner (hanging node) .

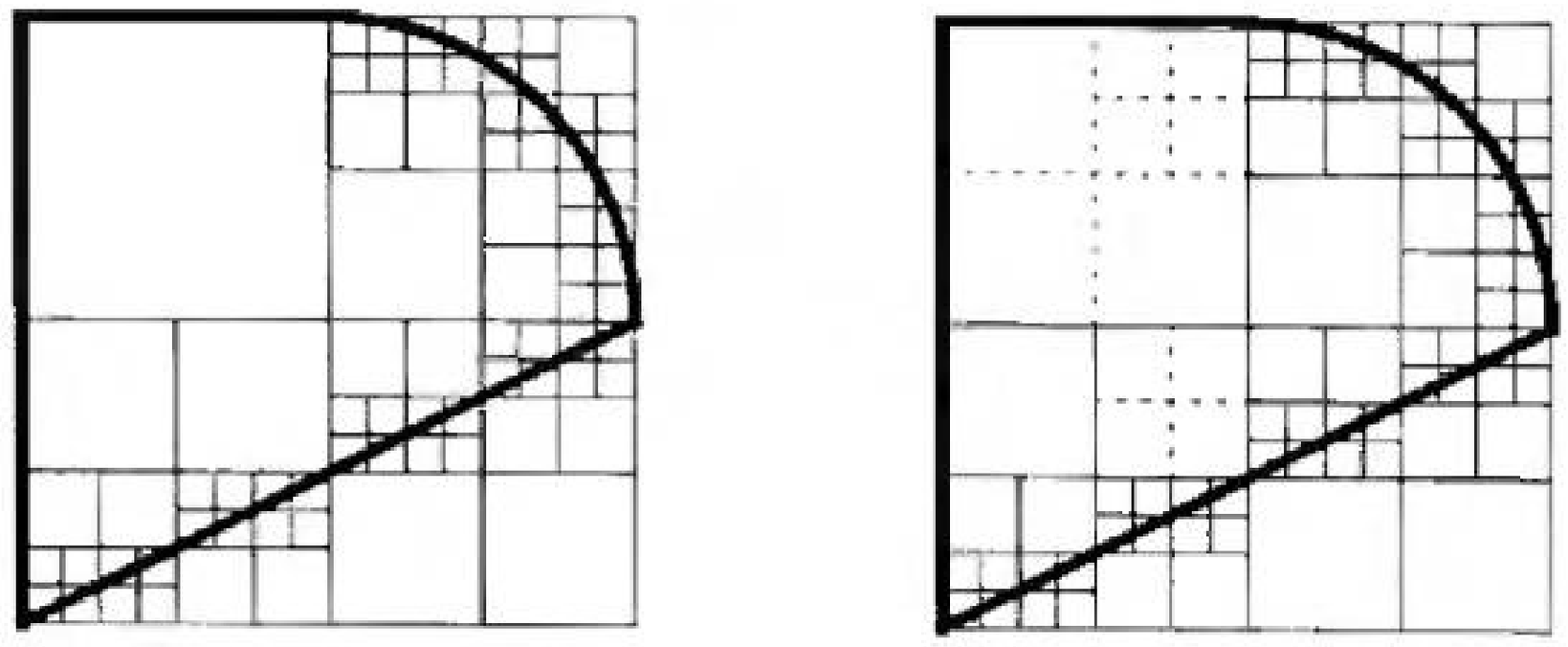


FIGURE 15.7 Quadtree example before (left) and after (right) one-level difference enforcement.

An example of tree balancing

Detail: Boundary Intersection

- Filtering of the intersecting points
- This step could be very tedious and time consuming.

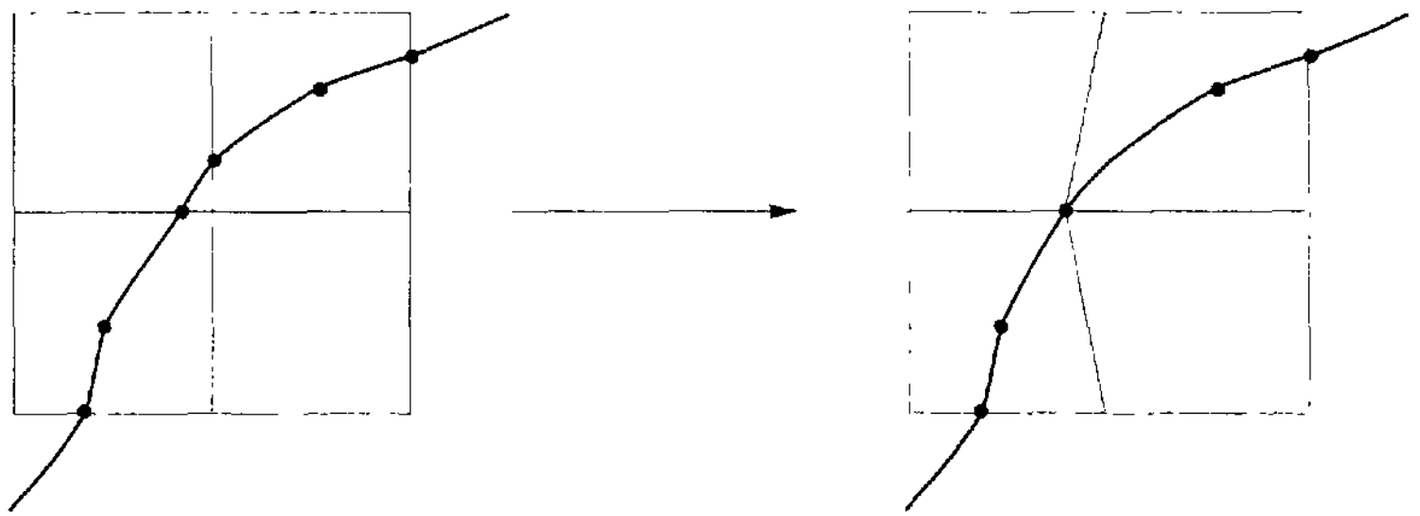


Figure 5.10: *Insertion of the boundary edges in the tree. Left, two intersection points are close to the quadrant corner. Right, the points have been merged together into a single one, the quadrant corner then being moved toward the resulting point.*

[Frey & George 2000]

- Inserting boundary edges and tree balancing.

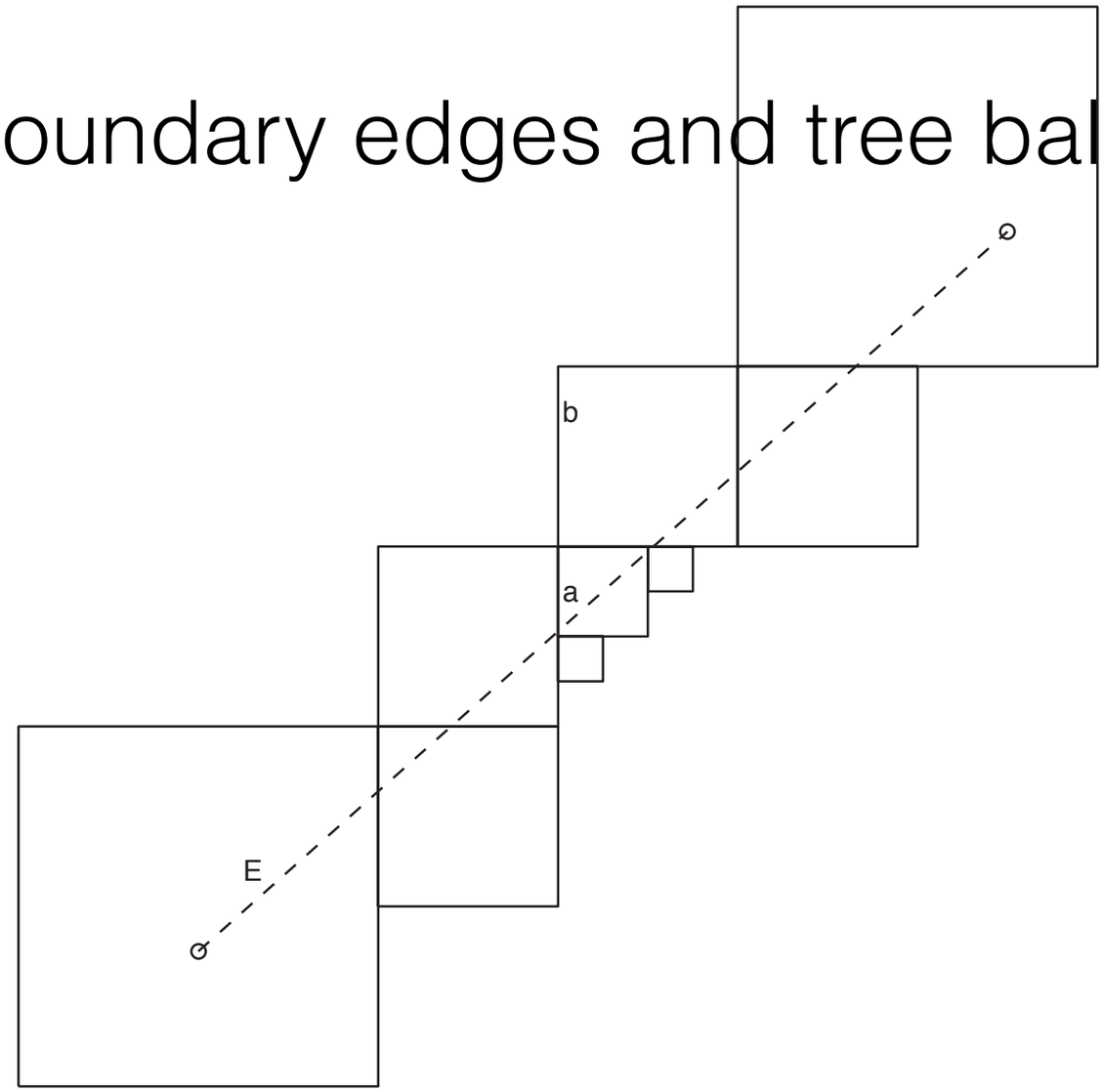


FIG. 6.1. *The alignment condition in the case $d = 2, k = 1$: the boxes in this figure are the extended orbit of a P-edge E, which is the dashed line. The two large boxes at the ends are protected boxes for the endpoints of E, protected from phase 0. In this figure, box a must be split because the alignment condition does not hold for this box. Its close subface, which could be either its lower left-hand corner or upper right-hand corner, is contained by another box smaller than a. All other boxes satisfy the alignment condition. For example, box b does not have to be split; its close subface could be either its bottom edge or right edge. The right edge will have higher priority, since the alignment condition holds for that edge.*

[Mitschell & Vavasis 2000]

Detail: Mesh Generation

The mesh vertices consist of cell corners and intersecting points of cell sides with boundary edges. Templates are designed to quickly triangulate the cells.

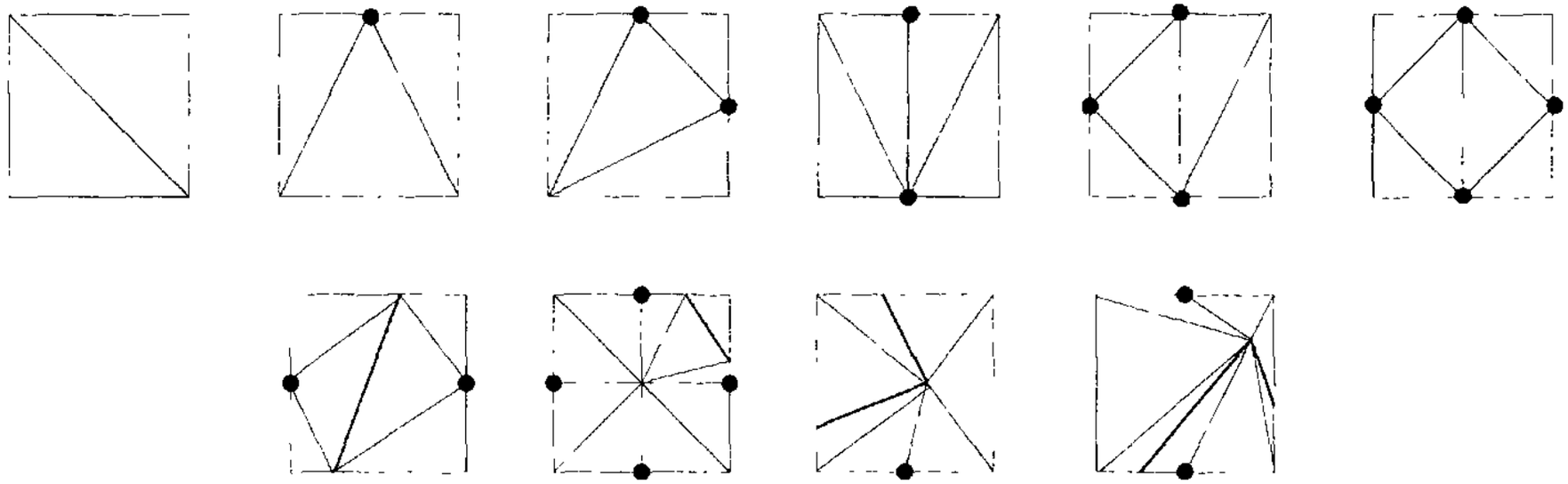
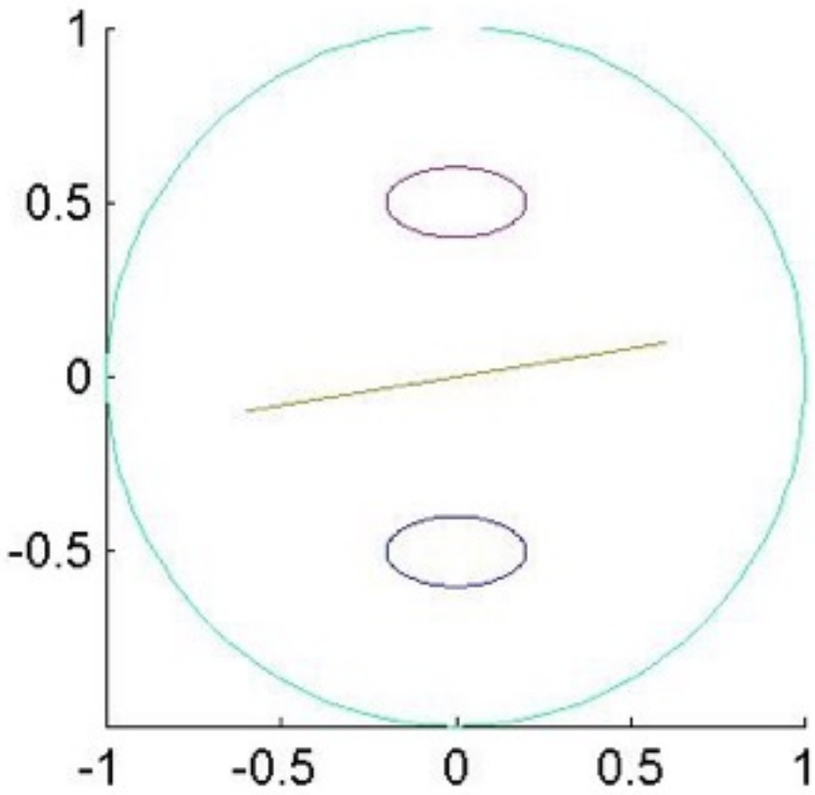
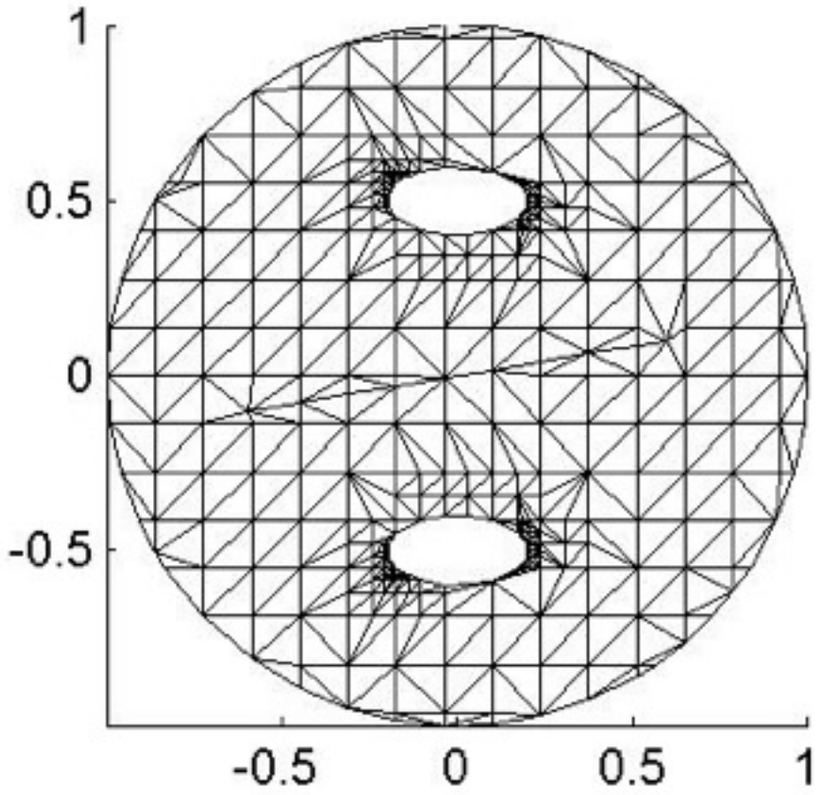


Figure 5.12: *A set of six plausible patterns to triangulate the internal quadrants (the other patterns can be retrieved using the rotational symmetry properties) and four patterns used to mesh boundary quadrants (in the last two patterns, the intersection point sees the other quadrant points).*





Software: QMG, Cornell University

Main Difficulties

- Specific consideration must be given to the interaction of the cells of the tree and the geometric domain.

- Determine the intersections of the cells of the tree represents the most complex aspect of the method.

- 3D cases are much more complicated.

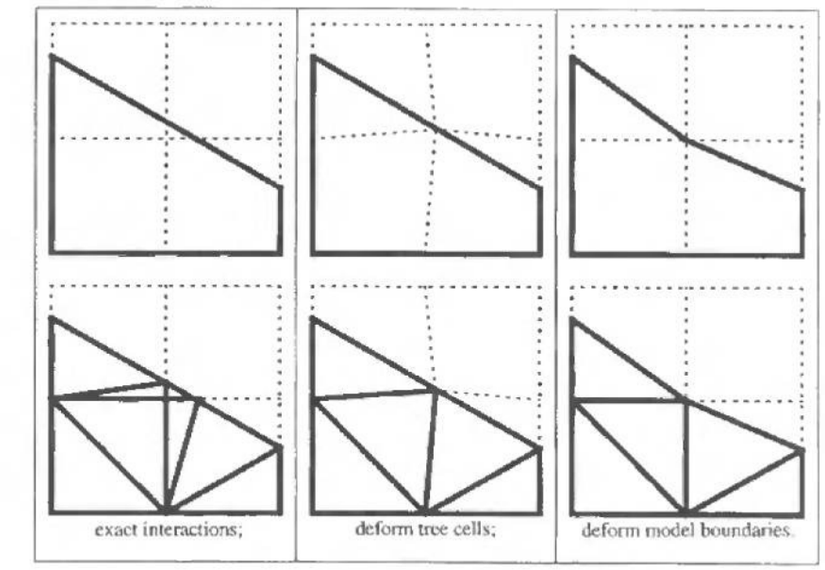
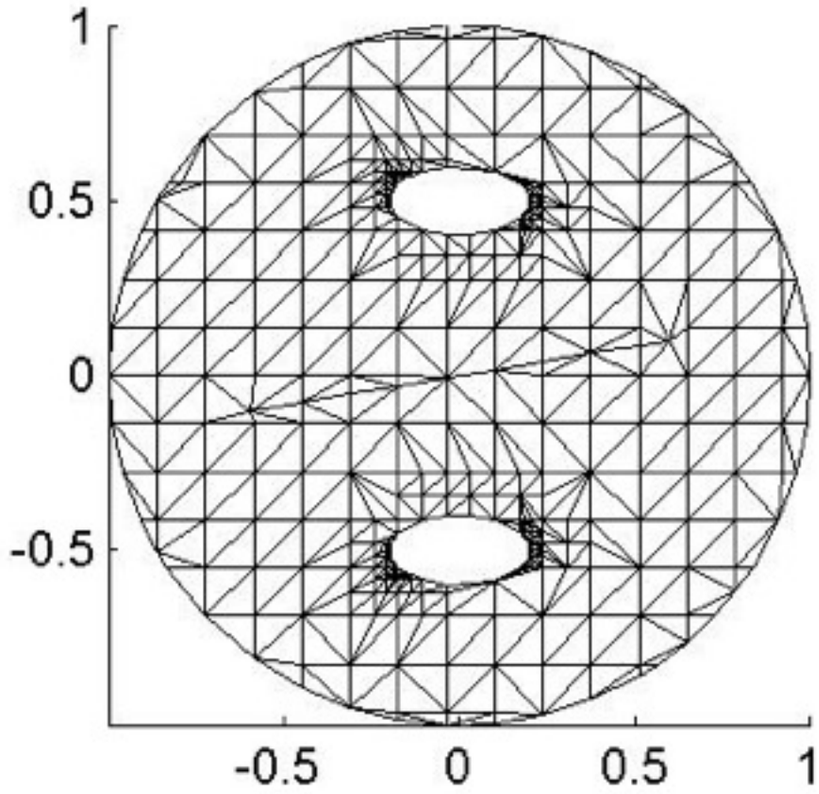


FIGURE 15.4 Options for the interactions of the model boundary with the boundary of the tree cells.

- Mesh quality are degraded at domain boundary.

- Mesh improvement is necessary.



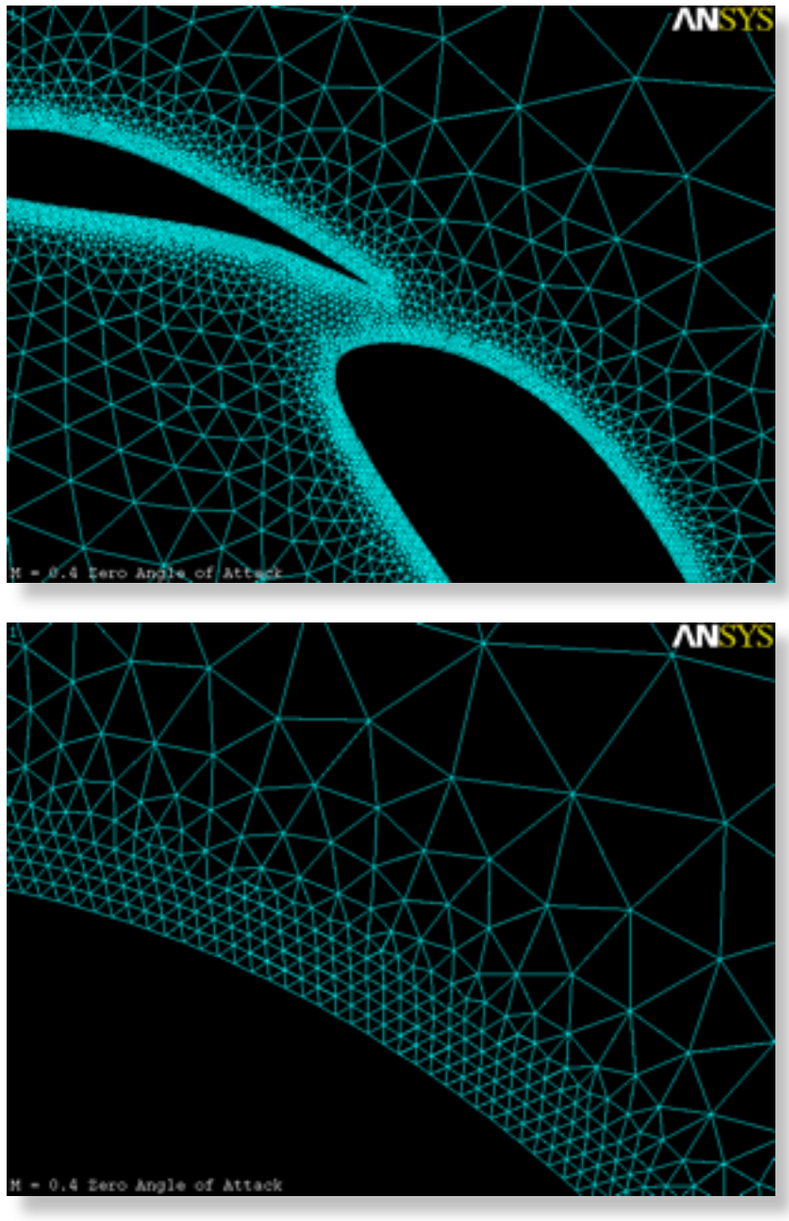
Advancing Front

Introduction

- Advancing Front methods for mesh generation has been a topic of research since 1980s, pioneered by [Lo-1985, Perrier et al-1987, Löhner & Parikh-1988].

- This type of methods is fully automatic for arbitrary complicated domains.

- It is now a very powerful and mature technique for generating high-quality unstructured meshes.



Ansys

Introduction (cont'd)

- This method allows for the generation of high quality (aspect ratio) mesh elements that fit domain boundary.

- Mesh sizing control is easily adapted by using mesh spacing control functions. Resulting nicely graded meshes.

- On the other hand, convergence problems can occur, especially in 3d, as it is not always guarantee how to cover the entire domain.

General Principles

- Assume the domain boundary has been proper subdivided (into) the right element shape and size.

- Form the initial front of all boundary edges (faces).

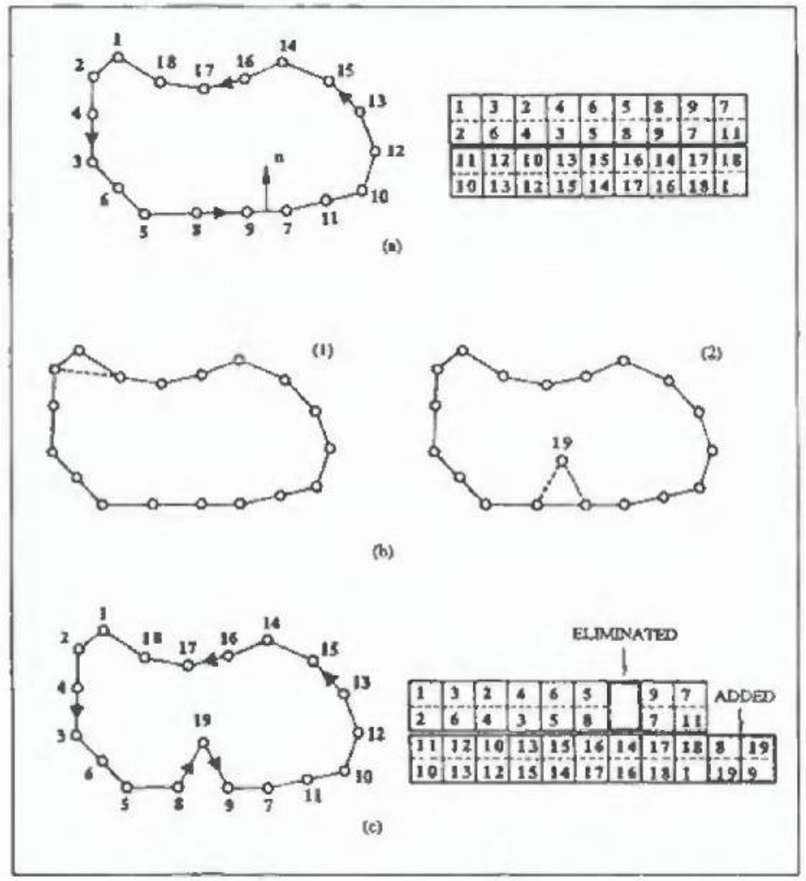


FIGURE 17.4 The front updating procedure in two dimensions. (a) The initial generation front. (b) Creation of a new element: (1) no new point is created; (2) the new point 19 is created. (c) The updating of the front for the case (b) (2).

[Peraire et al1998]

General Principles

1. Generate new mesh elements from current front.

2. Update the front.

3. Repeat 1 and 2 until the front is empty.

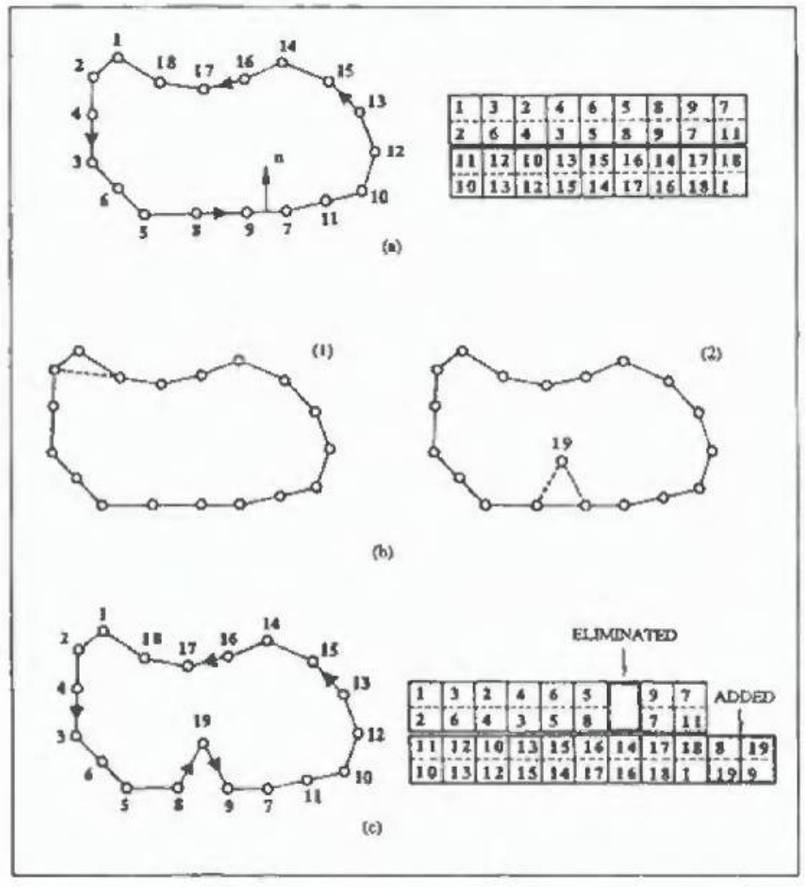
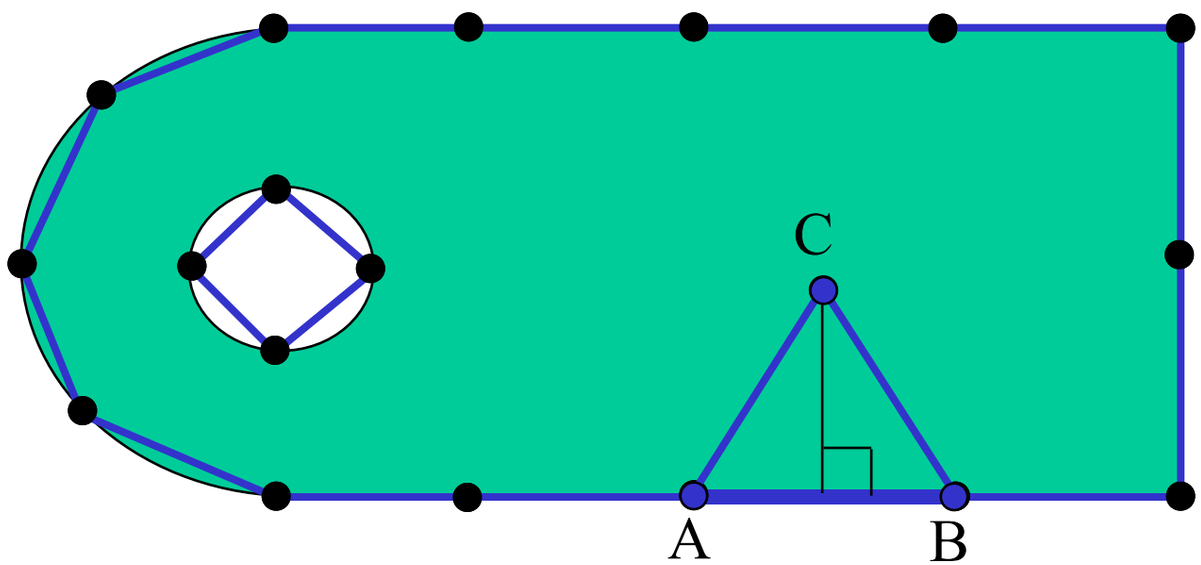


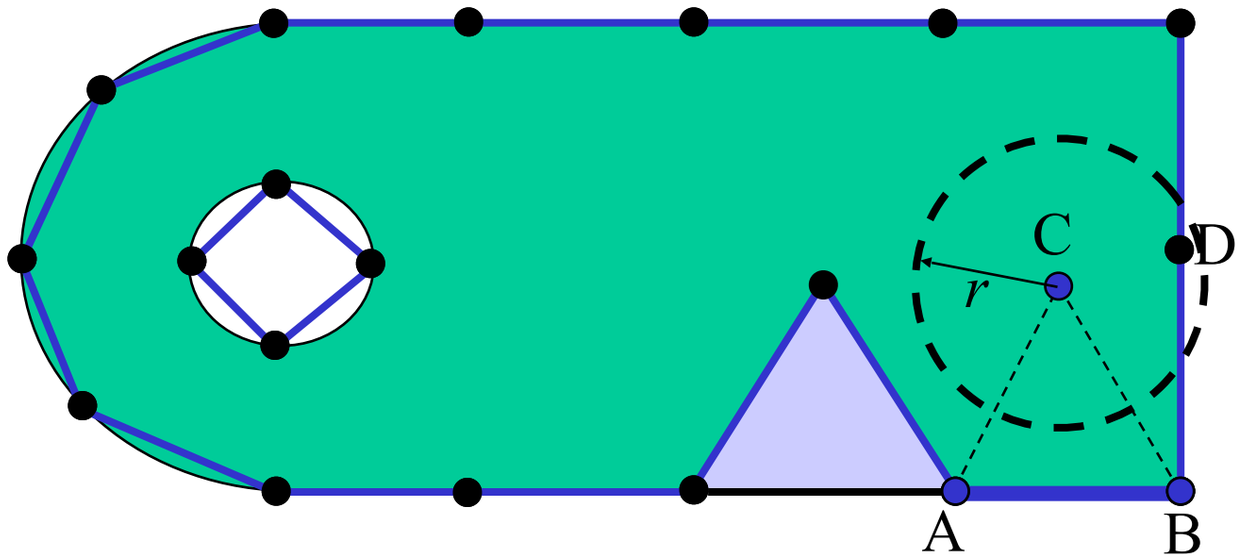
FIGURE 17.4 The front updating procedure in two dimensions. (a) The initial generation front. (b) Creation of a new element: (1) no new point is created; (2) the new point 19 is created. (c) The updating of the front for the case (b) (2).

Advancing Front



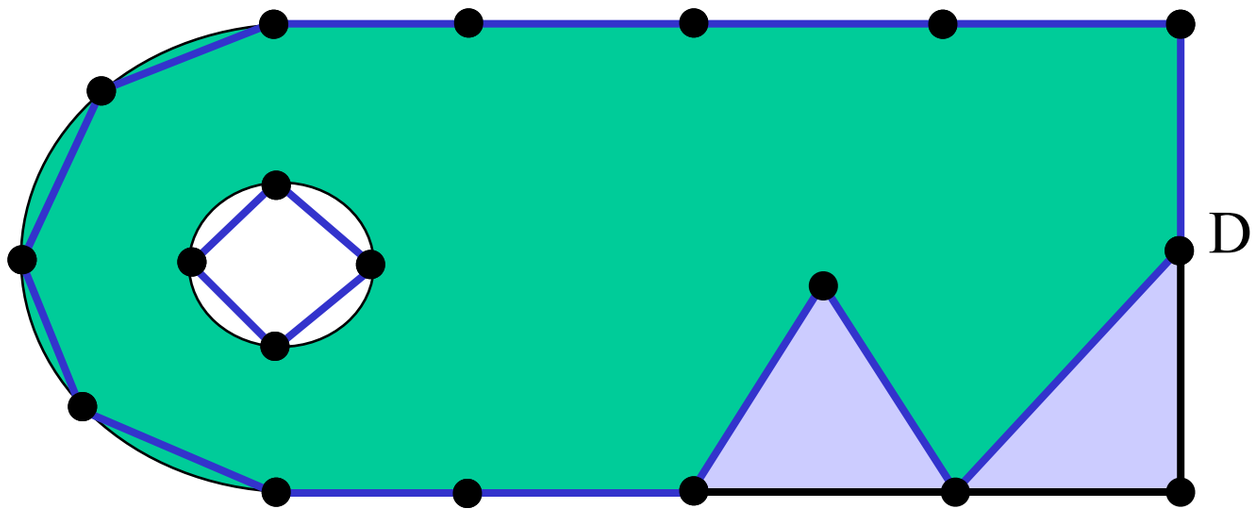
•Begin with boundary mesh - define as initial *front*
•For each edge (face) on front, locate ideal node C based on front AB

Advancing Front



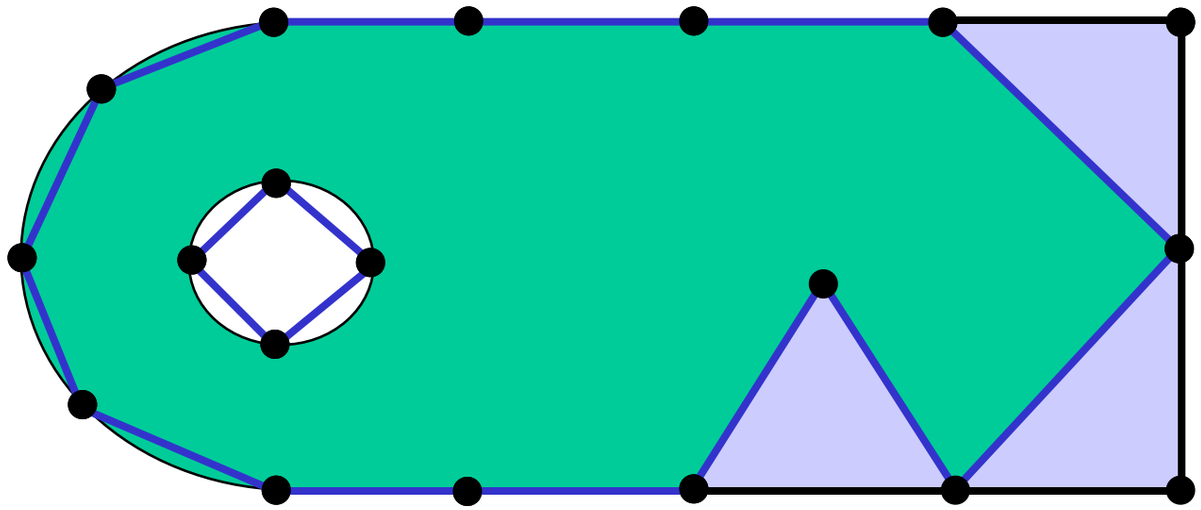
•Determine if any other nodes on current front are within search radius *r* of ideal location C (Choose D instead of C)

Advancing Front



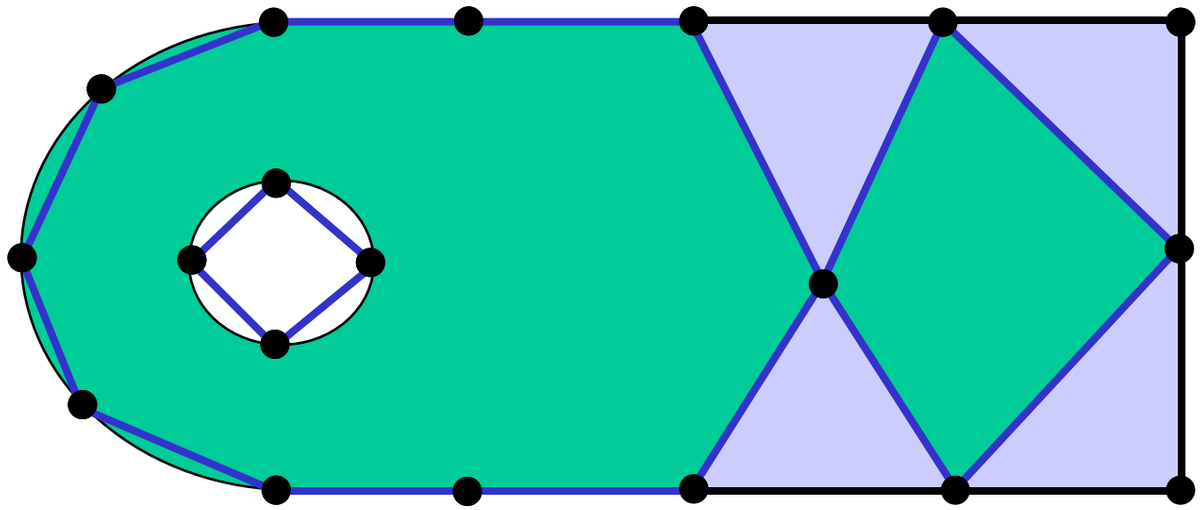
•Book-Keeping: New *front edges* added and deleted from *front* as triangles are formed
•Continue until no *front edges* remain on *front*

Advancing Front



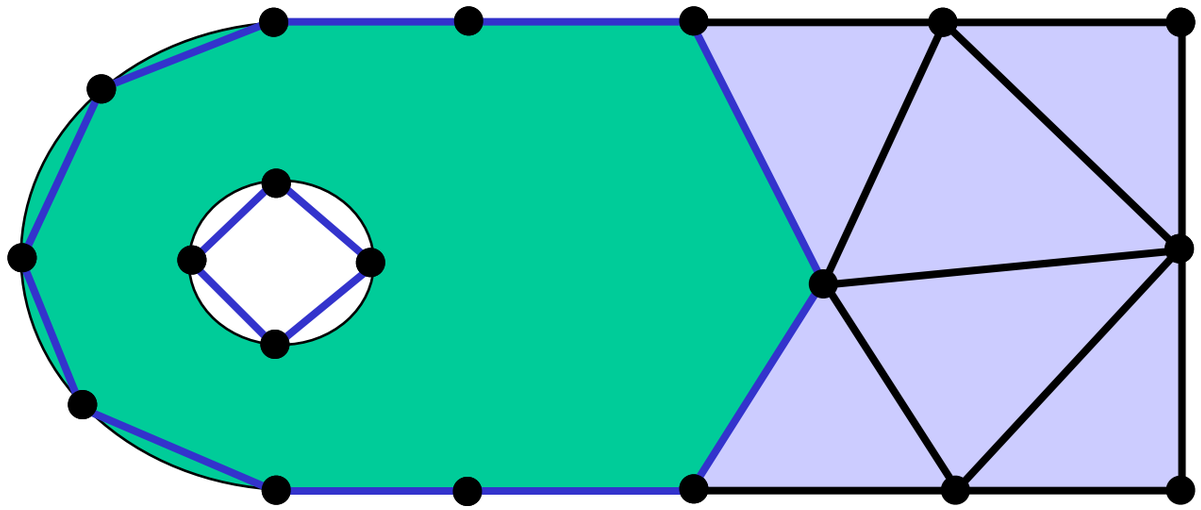
•Book-Keeping: New *front edges* added and deleted from *front* as triangles are formed
•Continue until no *front edges* remain on *front*

Advancing Front



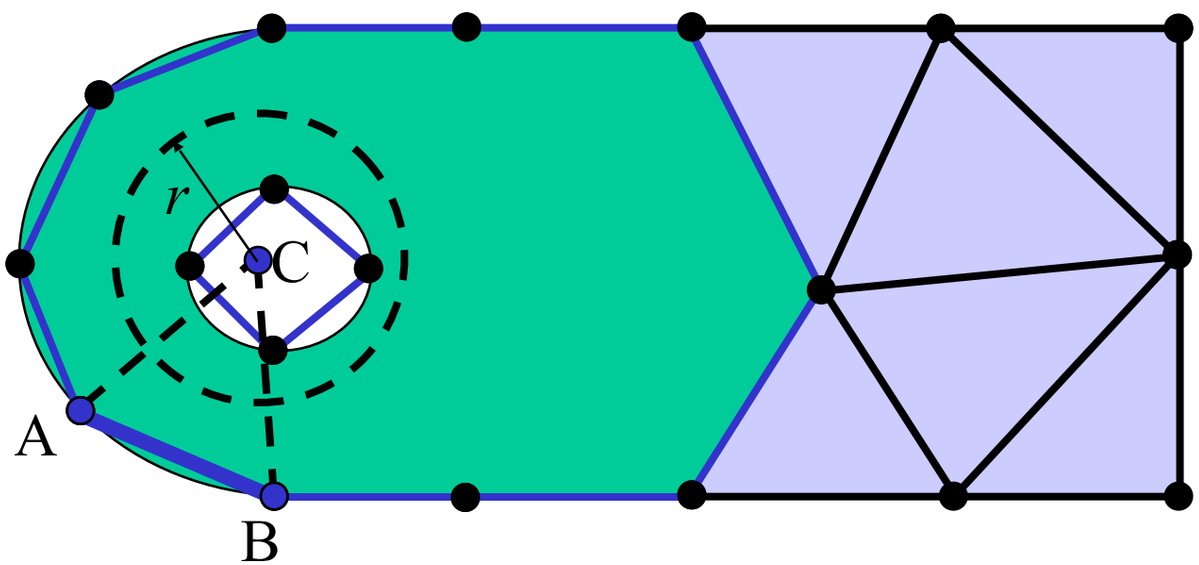
•Book-Keeping: New *front edges* added and deleted from *front* as triangles are formed
•Continue until no *front edges* remain on *front*

Advancing Front



•Book-Keeping: New *front edges* added and deleted from *front* as triangles are formed
•Continue until no *front edges* remain on *front*

Advancing Front



- Where multiple choices are available, use best quality (closest shape to equilateral)
- Reject any that would intersect existing front
- Reject any inverted triangles ($|AB \times AC| > 0$)
- (Lohner,88;96)(Lo,91)

Detail: Creation of New Points

1. Select a front entity f (based on a specific criterion)

2. Determine a ``best-point" position P_opt for this entity.

3. Identify if a point P exists in the current mesh that should be used in preference to P_opt. If such a point exists, consider using it as P_opt.

4. Form an element K with f and P_opt.

5. Check if the element K intersects any mesh entity. If this check fails, pick a new point P (if any) and return to 4.

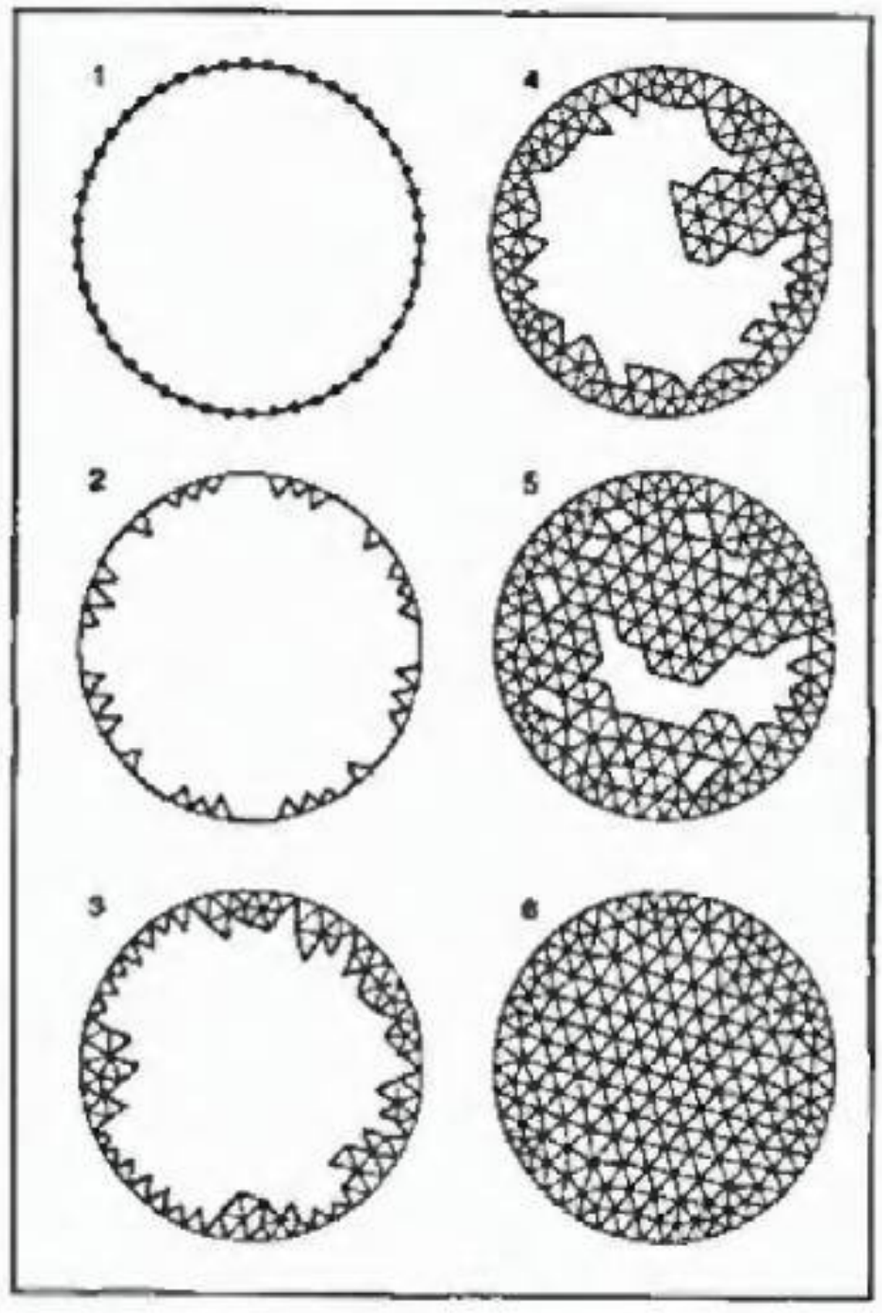


FIGURE 17.5 The advancing front technique showing different stages during the triangulation process.

Detail: Finding Optimal Point Location

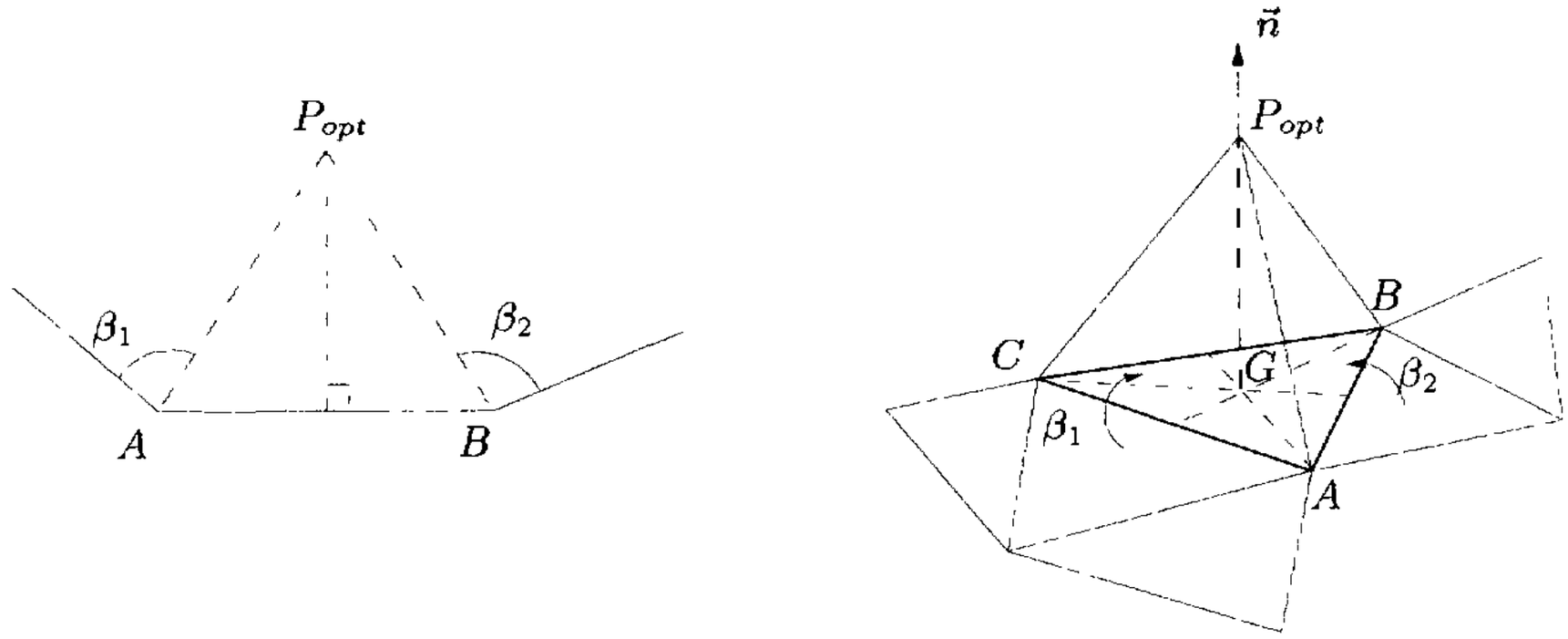


Figure 6.4: *Optimal point creation in two dimensions (left-hand side) and three dimensions (right-hand side).*

Detail: Finding Optimal Point Location

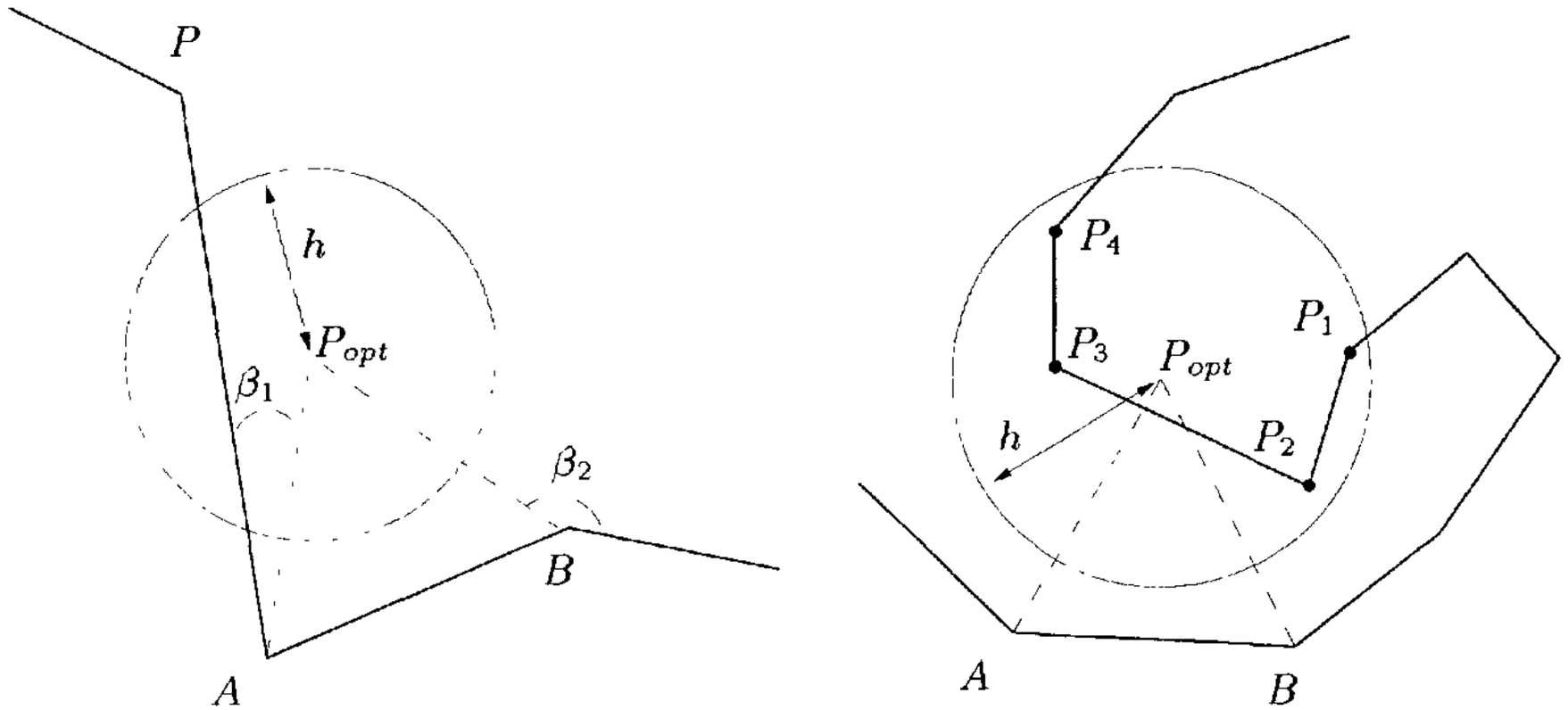
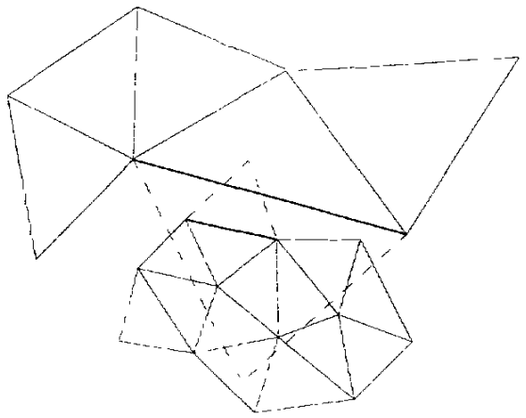


Figure 6.5: *Identification of all potential candidates for optimal point creation in two dimensions. Left : no candidate other than P_{opt} exists. Right : the P_i's represent all the possible candidates.*

Detail: Avoid Front Collision



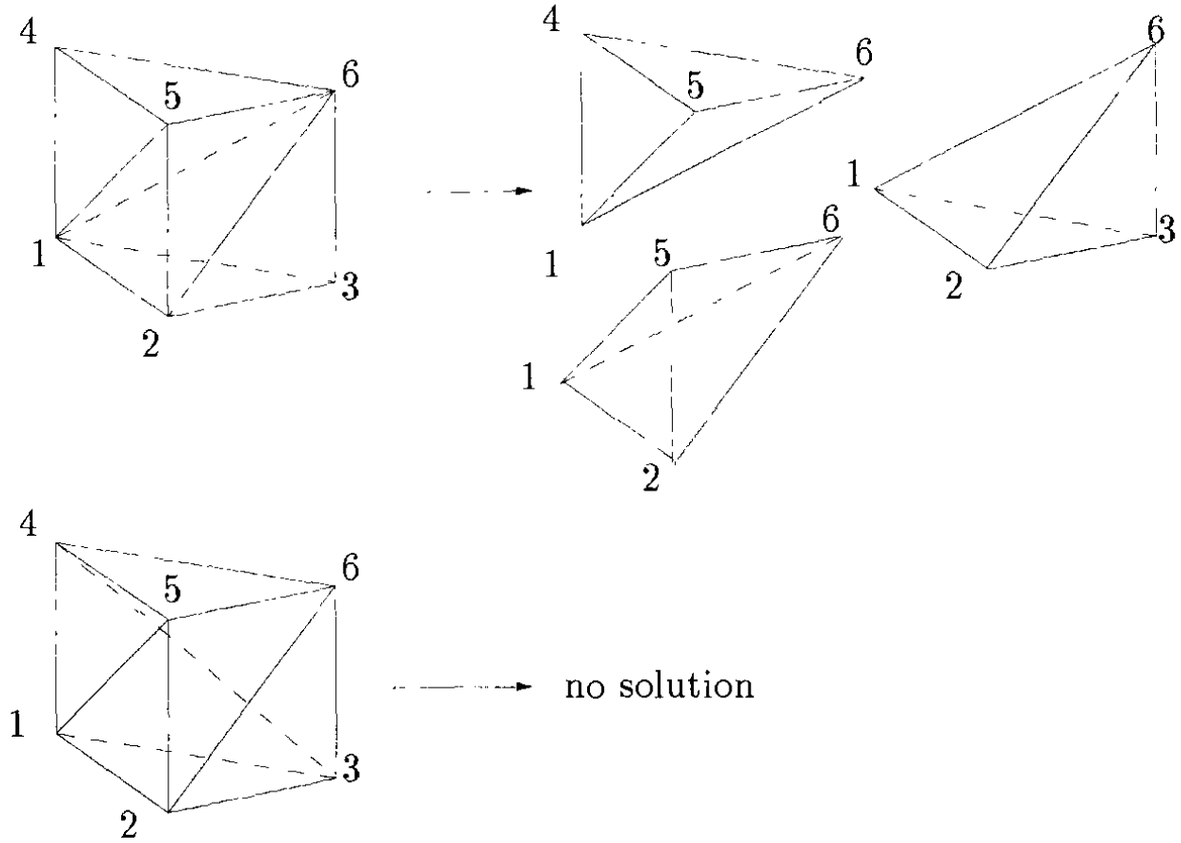


Figure 6.7: *Schönhardt polyhedron : valid and non-decomposable (without adding an internal point) constrained triangulation of a regular prism.*

Remarks

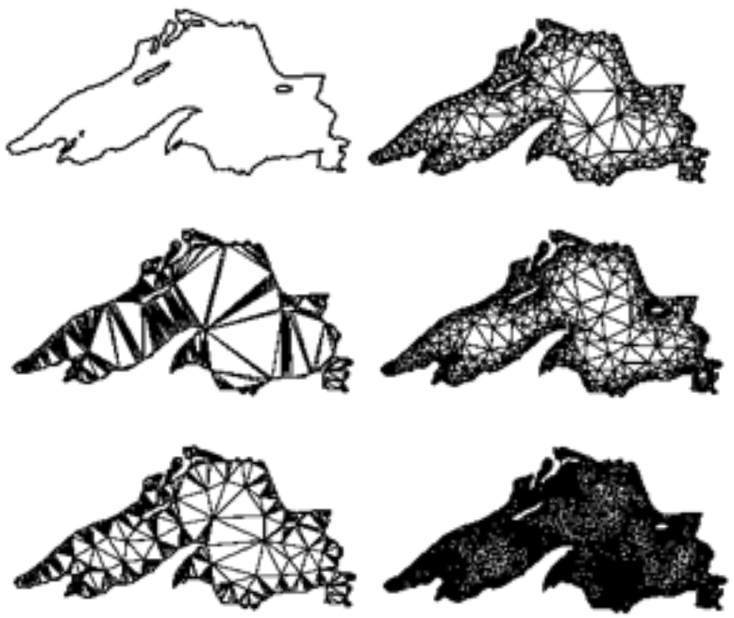
- The principle of any AFT method is relatively simple and practical. It generates high quality meshes.

- However, several details to be implemented are all based on heuristics.

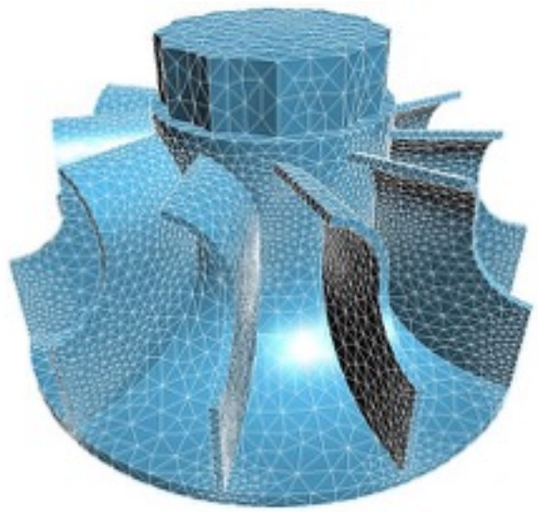
- In 3d, none of the AFT methods has guarantee that it will complete.

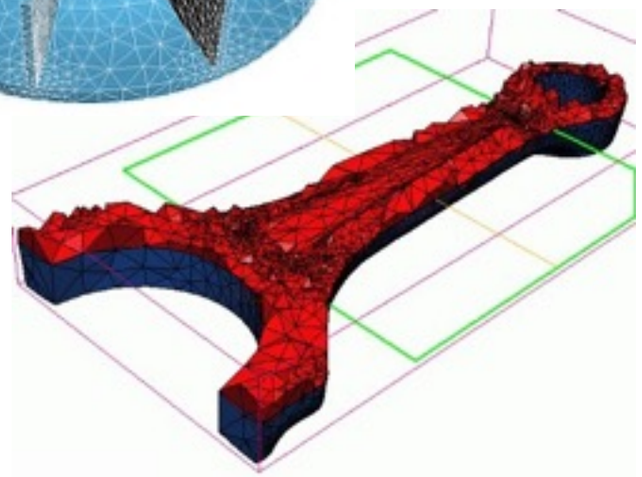
Delaunay-based Methods

Delaunay



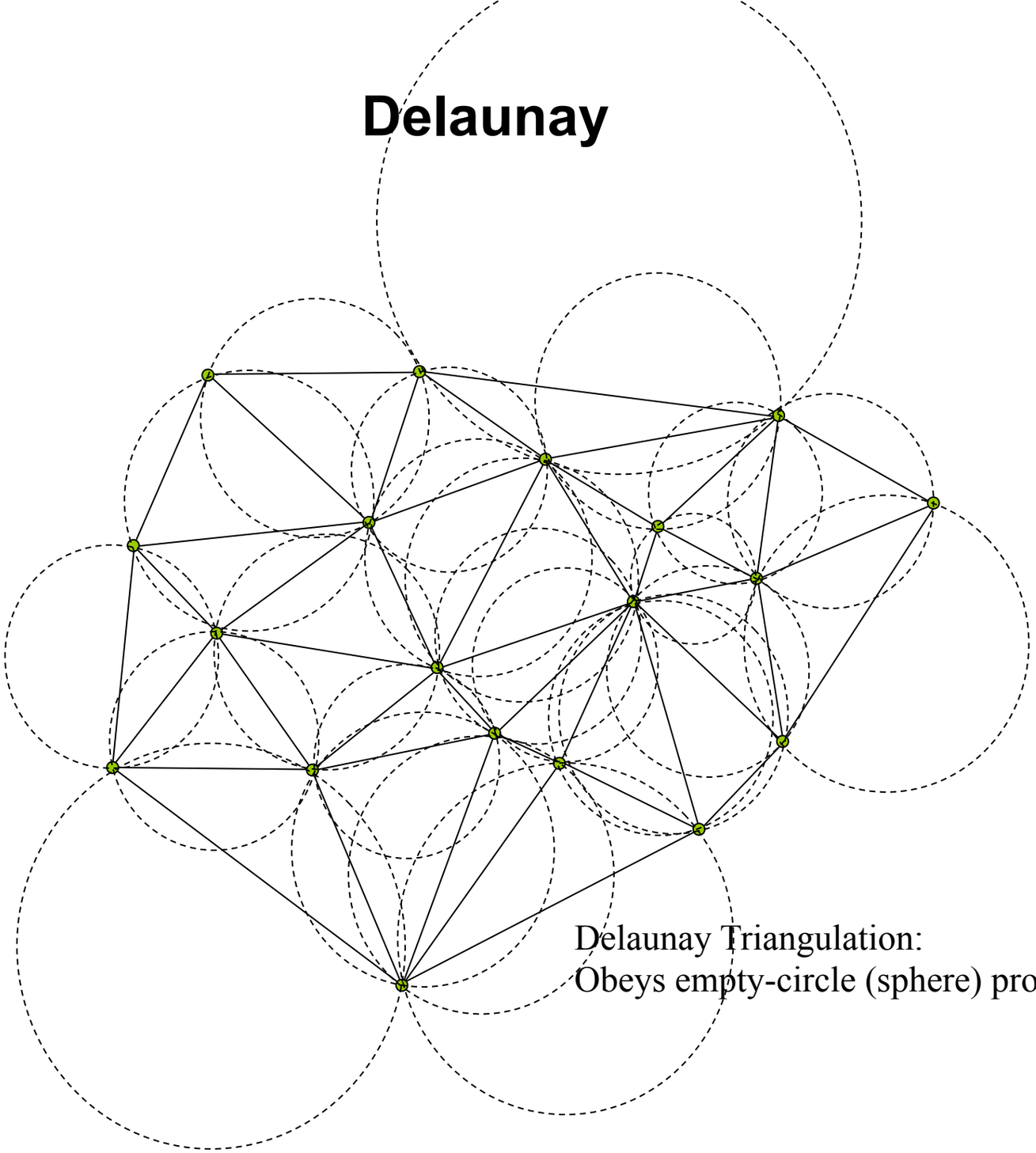
Triangle
Jonathon Shewchuk
http://www-2.cs.cmu.edu/~quake/triangle.html





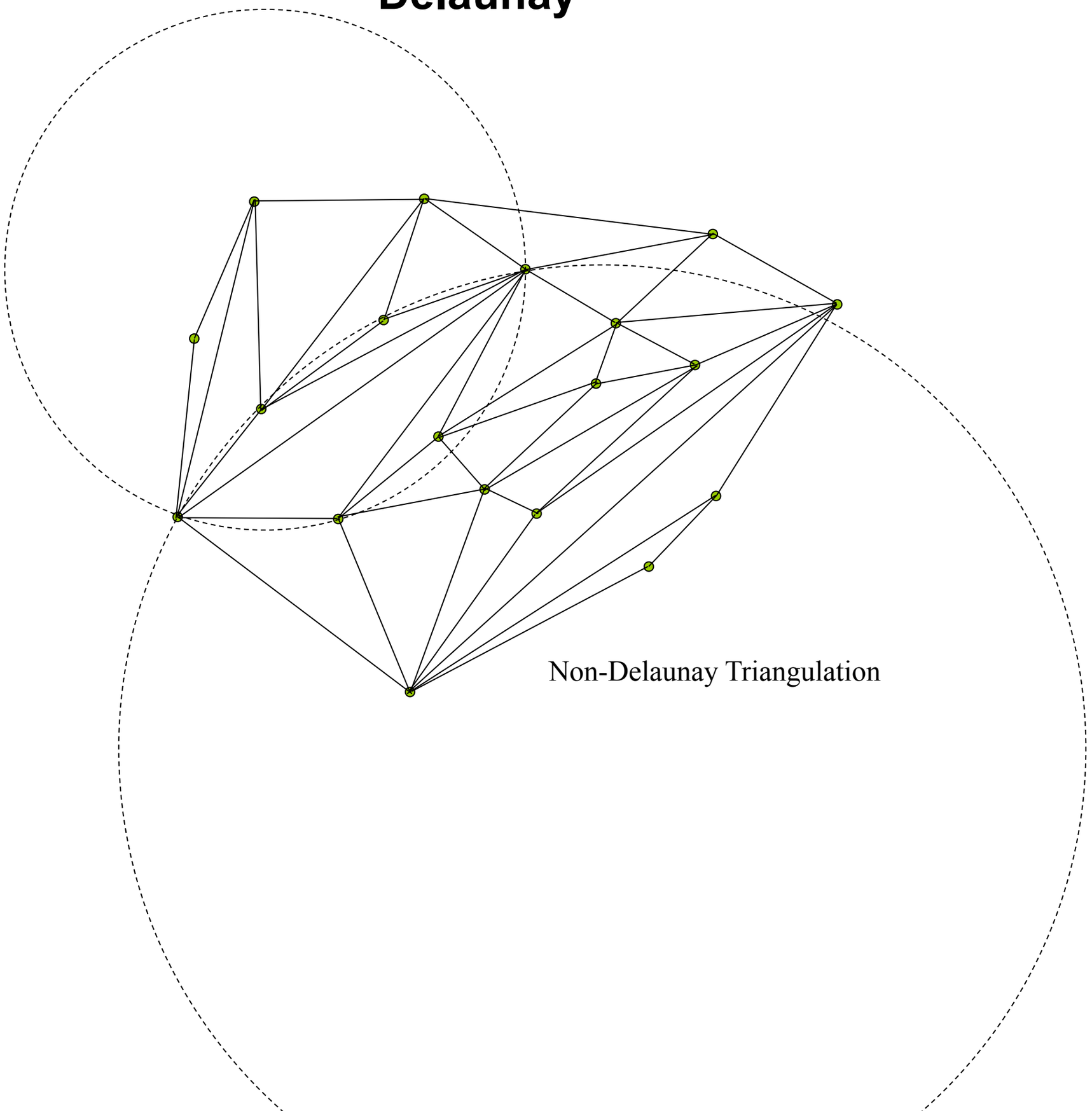
Tetmesh-GHS3D
INRIA, France
http://www.simulog.fr/tetmesh/

Delaunay



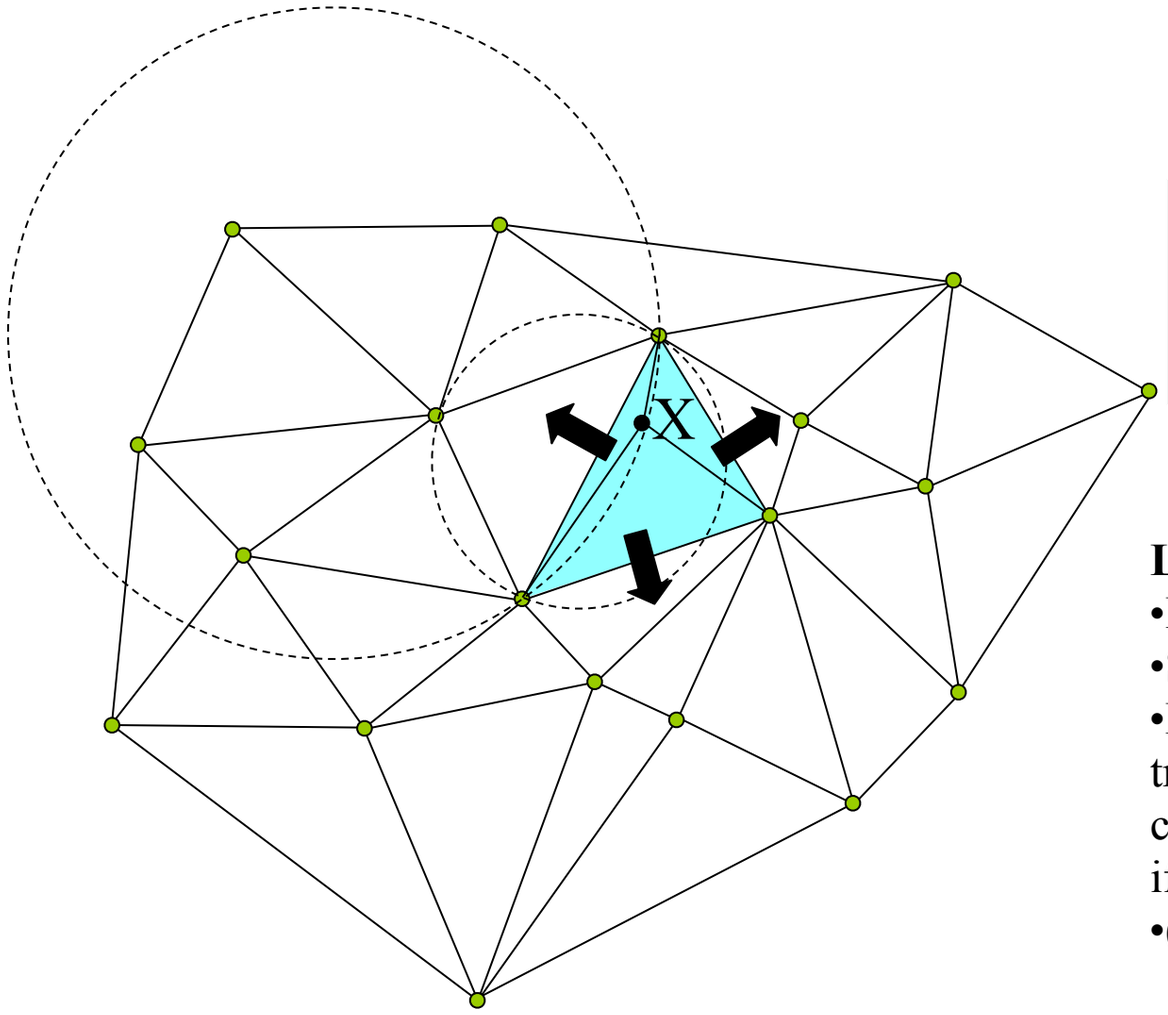
Delaunay Triangulation:
Obeys empty-circle (sphere) property

Delaunay



Non-Delaunay Triangulation

Delaunay

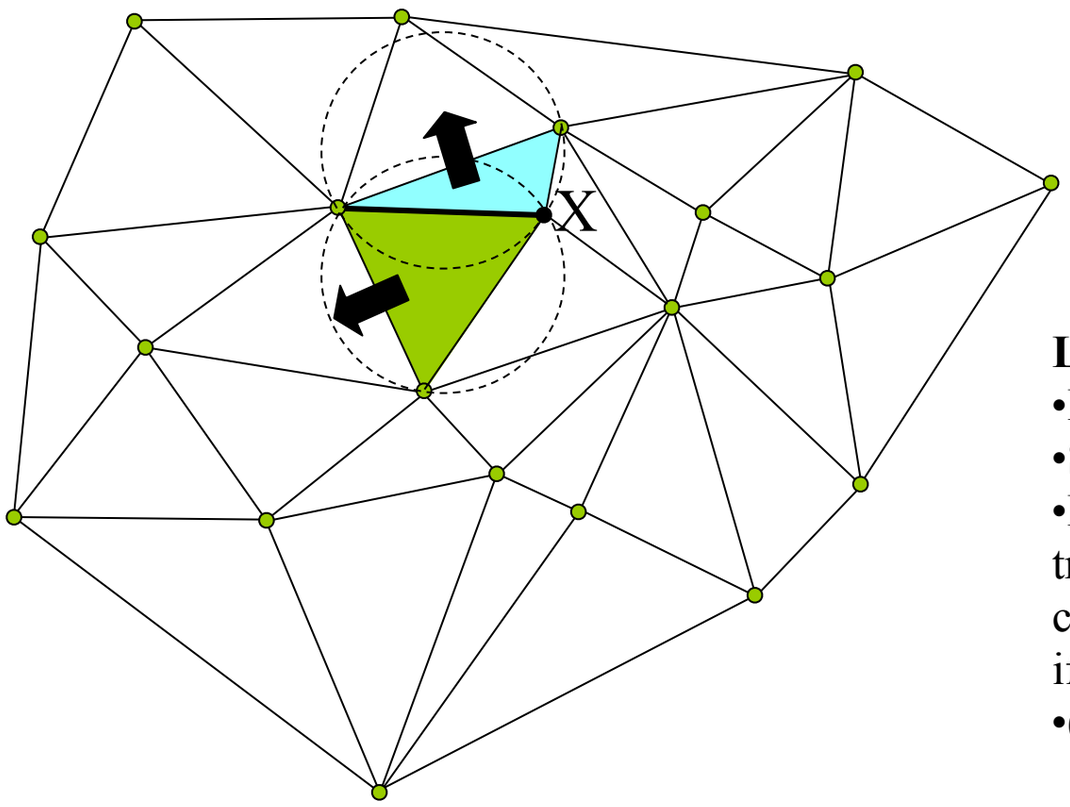


Given a Delaunay Triangulation of n nodes, How do I insert node n+1 ?

Lawson Algorithm

- Locate triangle containing X
- Subdivide triangle
- Recursively check adjoining triangles to ensure empty-circle property. Swap diagonal if needed
- (Lawson,77)

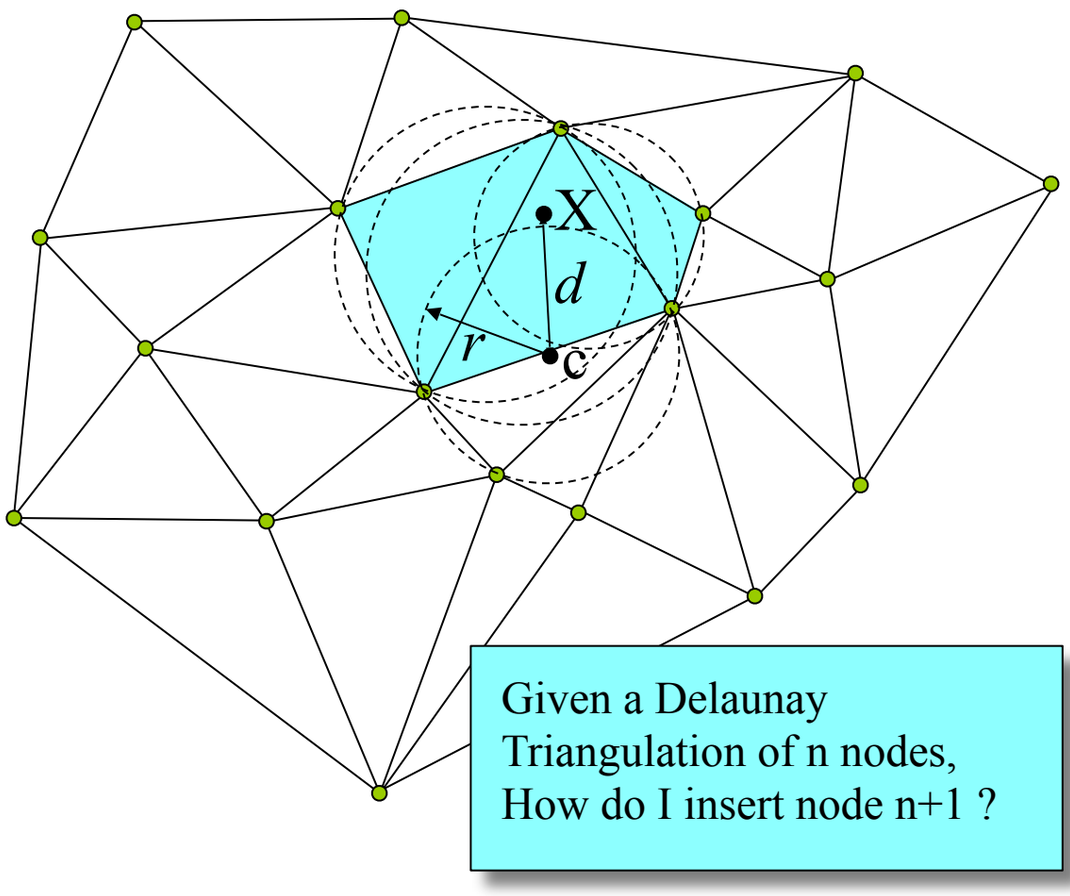
Delaunay



Lawson Algorithm

- Locate triangle containing X
- Subdivide triangle
- Recursively check adjoining triangles to ensure empty-circle property. Swap diagonal if needed
- (Lawson,77)

Delaunay

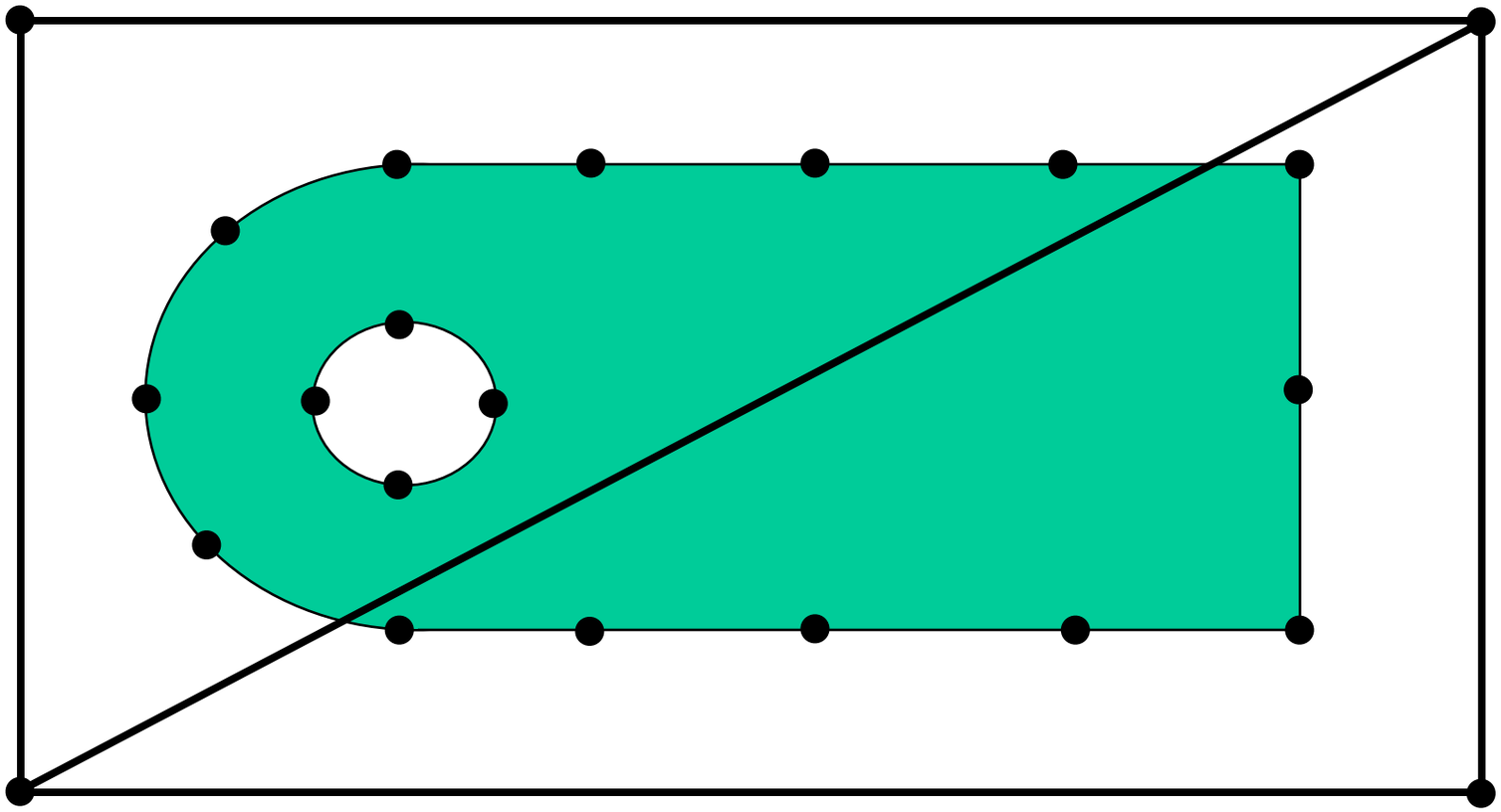


Given a Delaunay Triangulation of n nodes, How do I insert node n+1 ?

Bowyer-Watson Algorithm

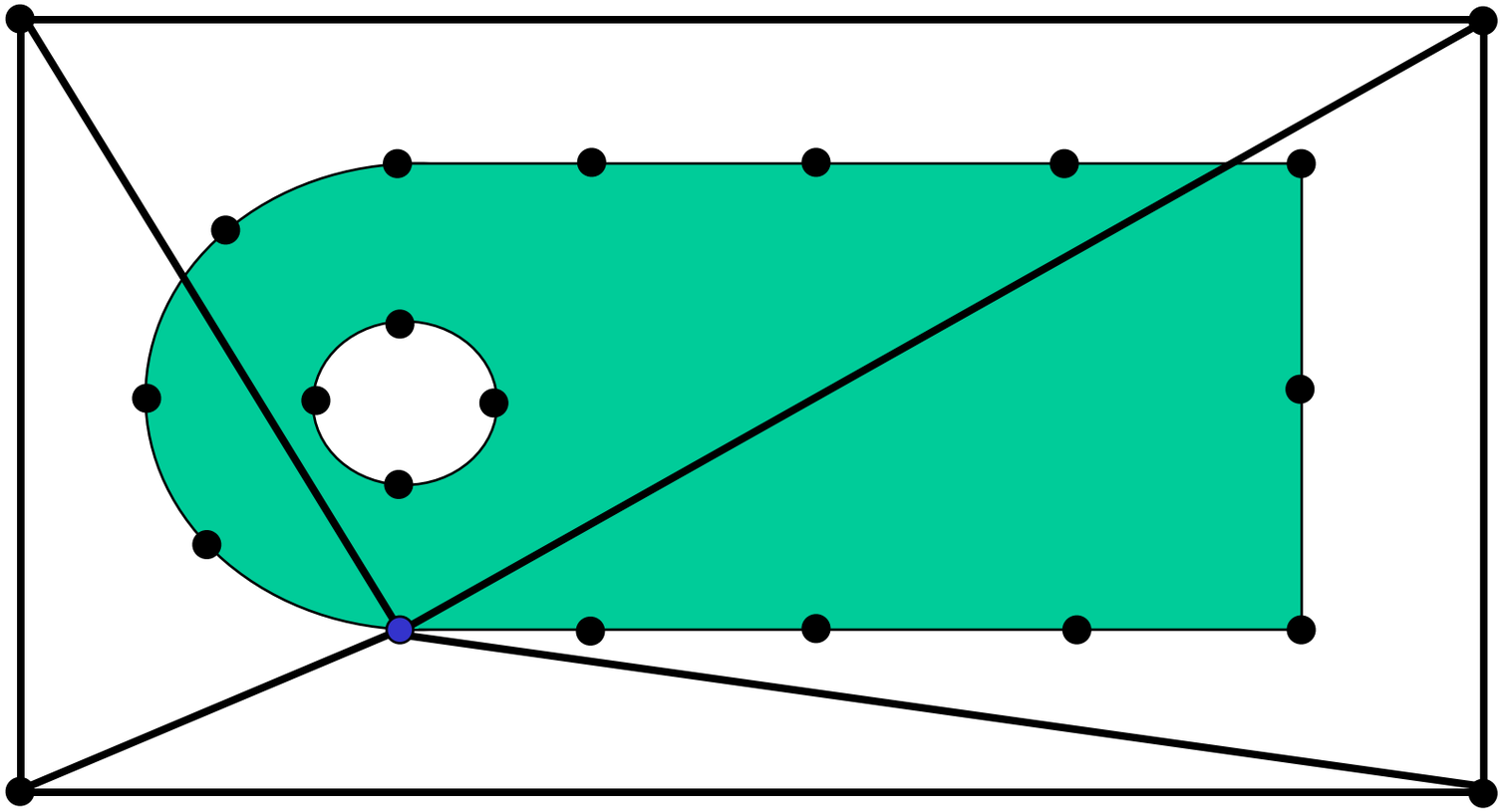
- Locate triangle that contains the point
- Search for all triangles whose circumcircle contain the point ($d<r$)
- Delete the triangles (creating a void in the mesh)
- Form new triangles from the new point and the void boundary
- (Watson,81)

Delaunay



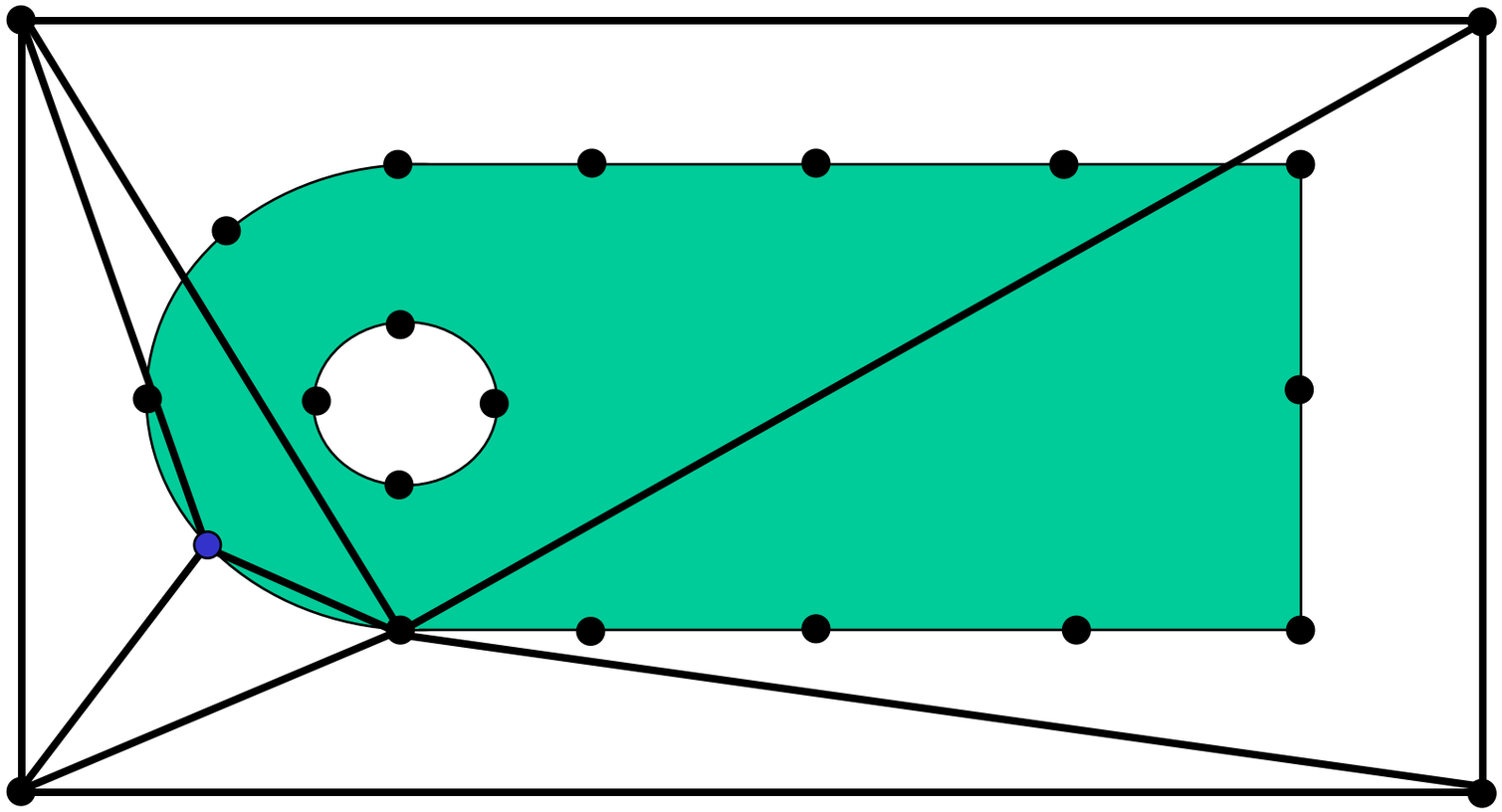
•Begin with Bounding Triangles (or Tetrahedra)

Delaunay



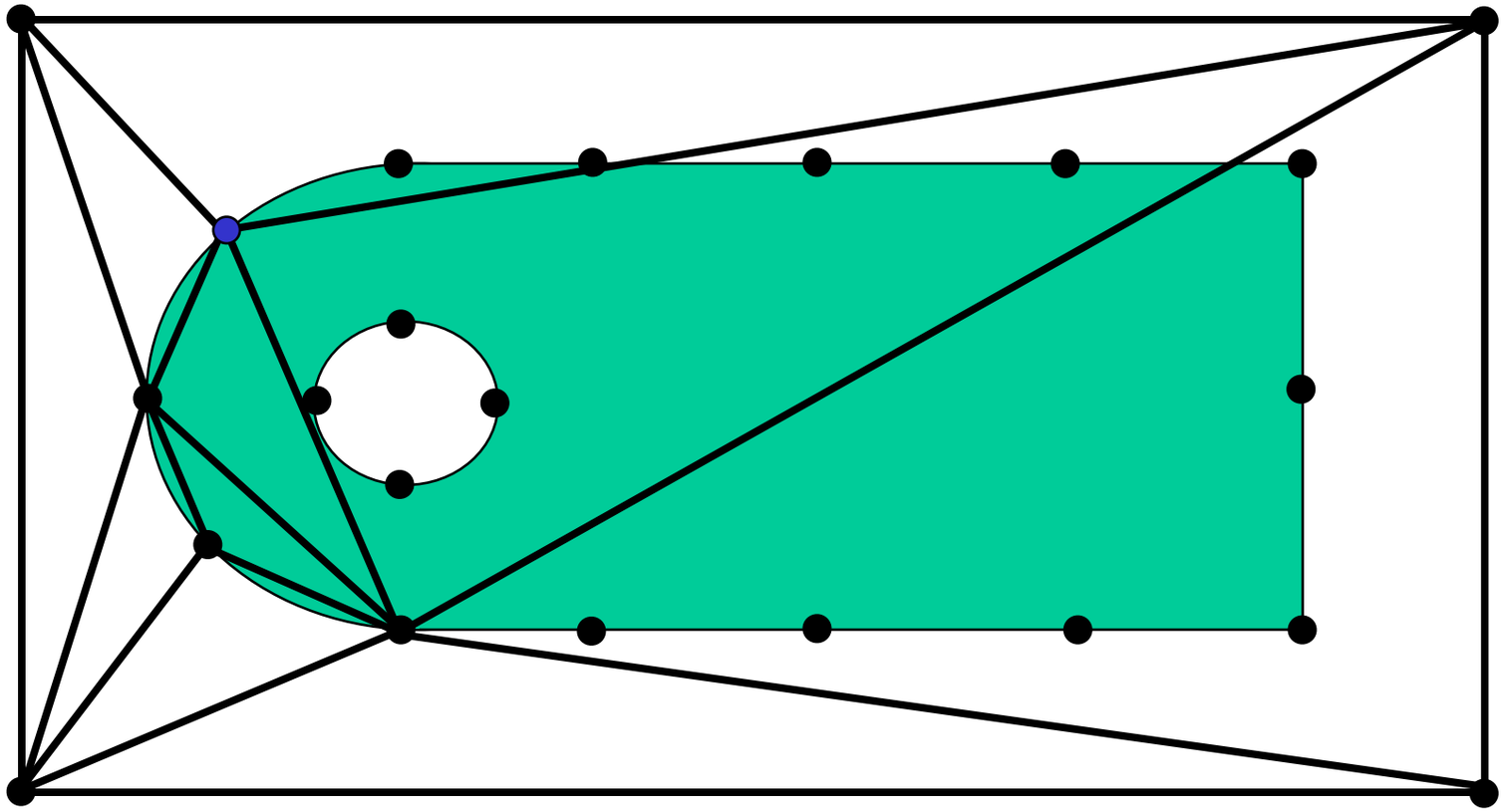
•Insert boundary nodes using Delaunay method
(Lawson or Bowyer-Watson)

Delaunay



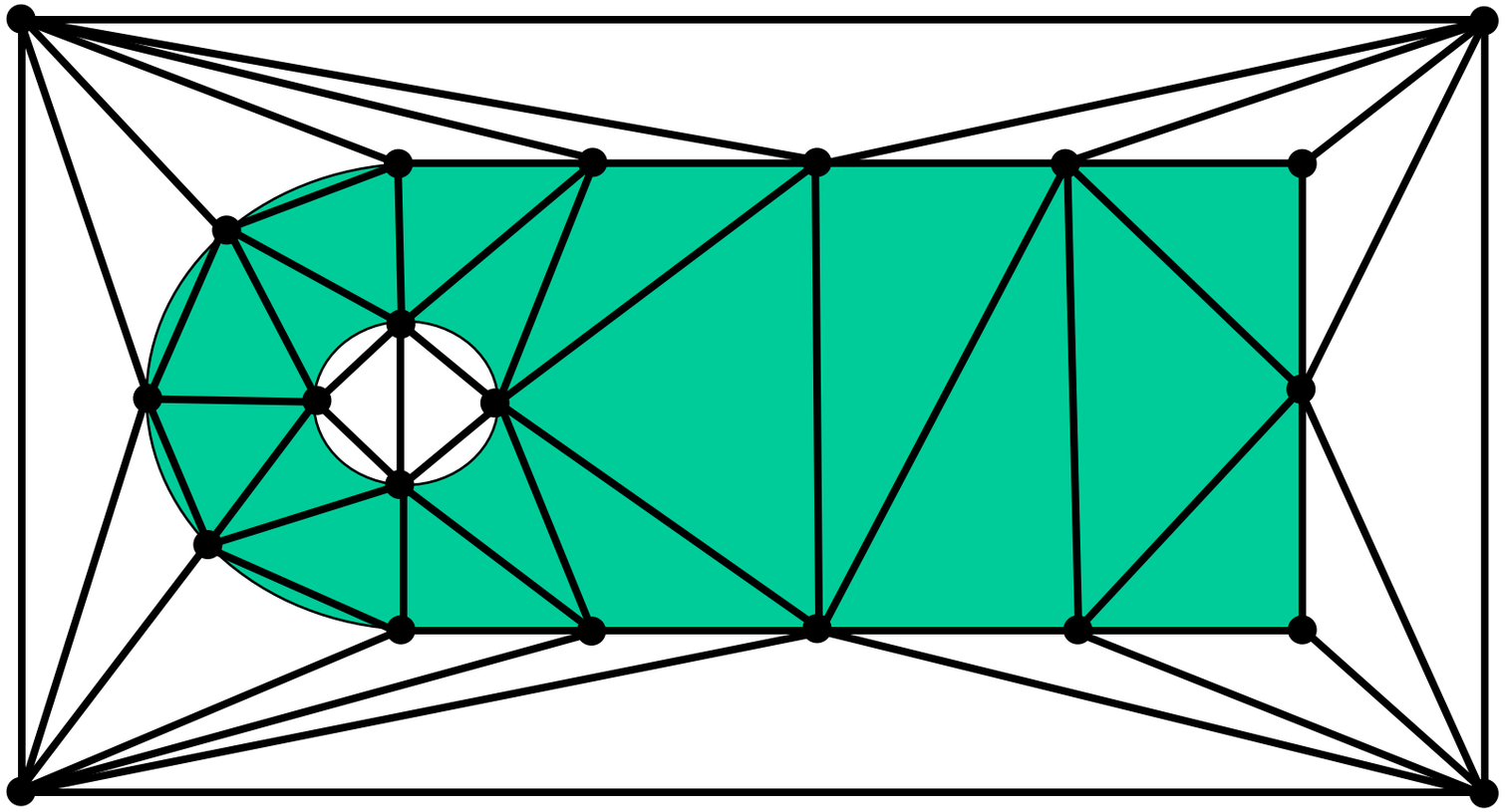
•Insert boundary nodes using Delaunay method
(Lawson or Bowyer-Watson)

Delaunay



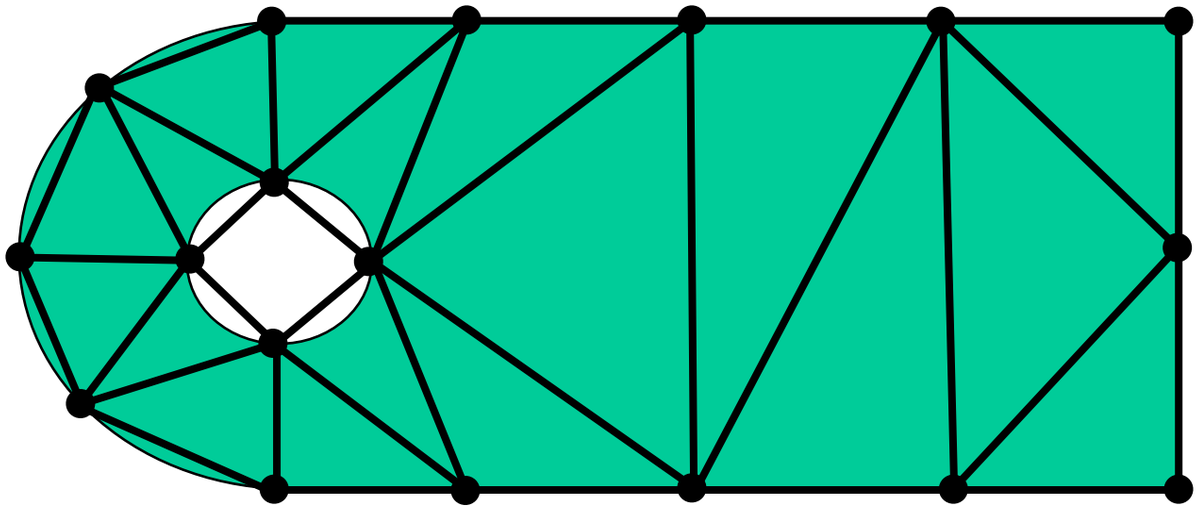
•Insert boundary nodes using Delaunay method
(Lawson or Bowyer-Watson)

Delaunay



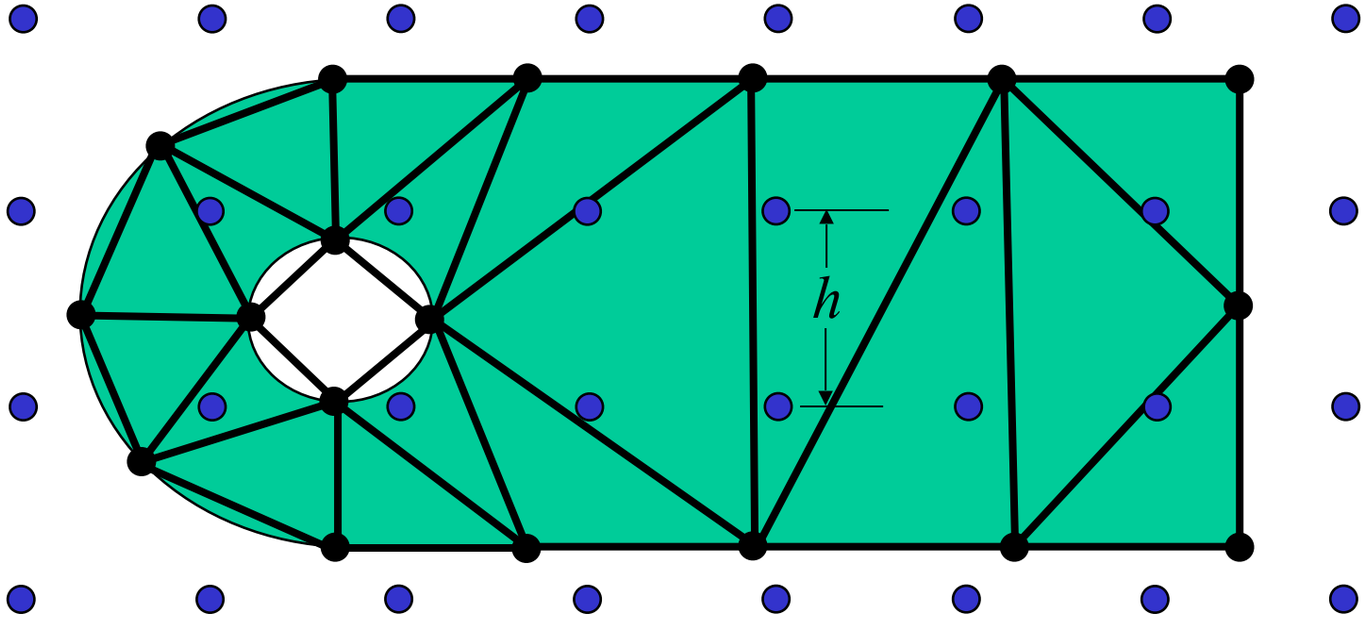
•Insert boundary nodes using Delaunay method
(Lawson or Bowyer-Watson)

Delaunay



•Recover boundary
•Delete outside triangles
•Insert internal nodes

Delaunay

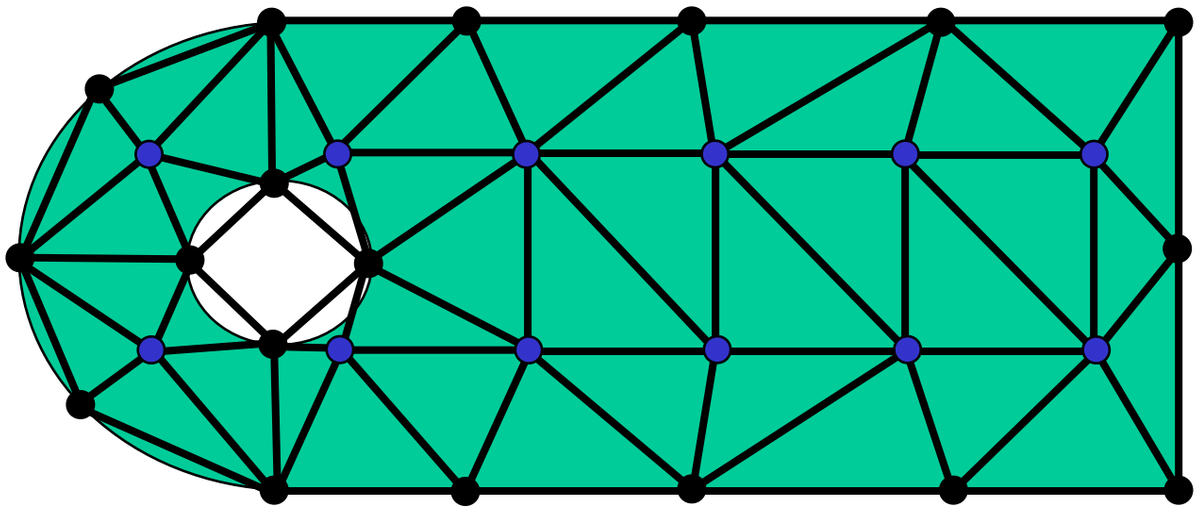


Grid Based

- Nodes introduced based on a regular lattice
- Lattice could be rectangular, triangular, quadtree, etc…
- Outside nodes ignored

Node Insertion

Delaunay



Grid Based

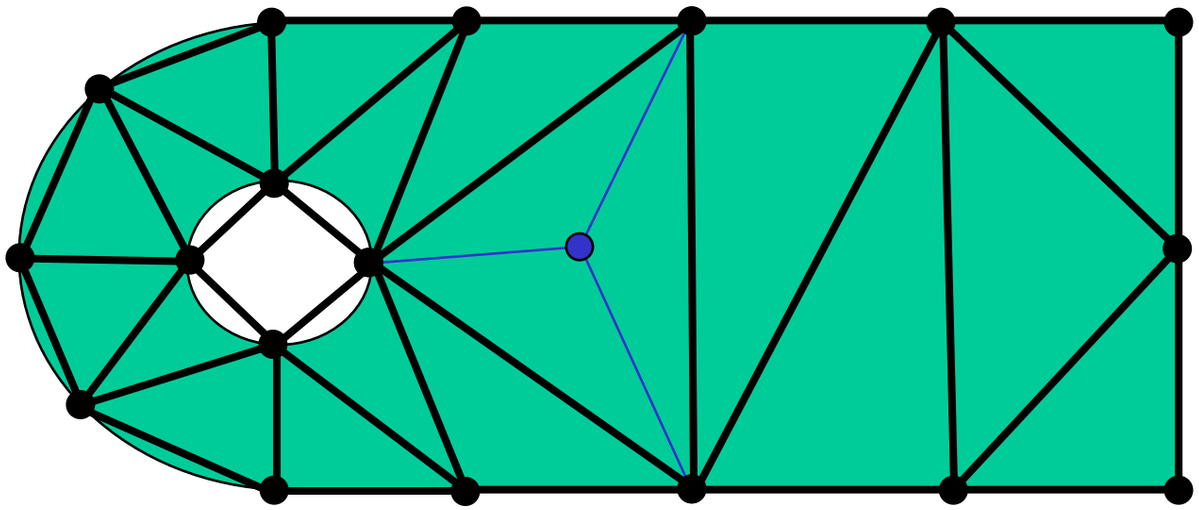
•Nodes introduced based on a regular lattice

•Lattice could be rectangular, triangular, quadtree, etc…

•Outside nodes ignored

Node Insertion

Delaunay



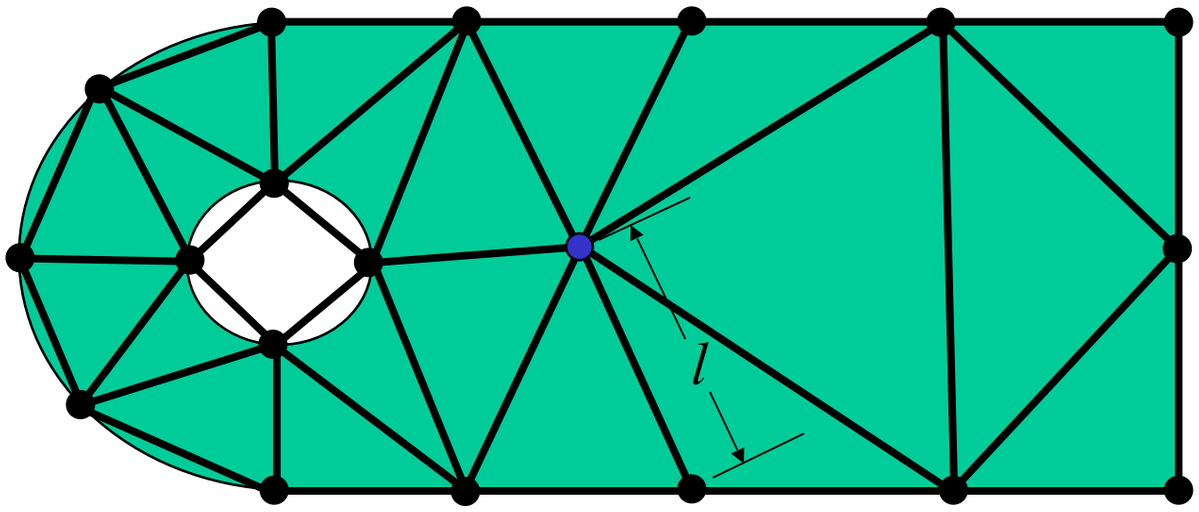
Centroid

•Nodes introduced at triangle centroids

•Continues until edge length, $l \approx h$

Node Insertion

Delaunay

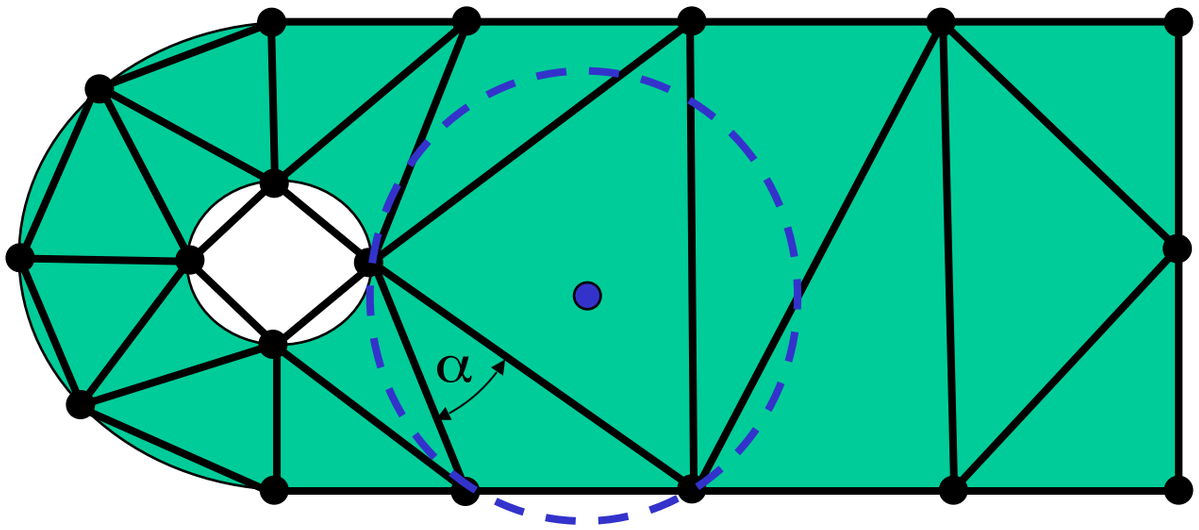


Centroid

- Nodes introduced at triangle centroids
- Continues until edge length, $l \approx h$

Node Insertion

Delaunay



Circumcenter (“Guaranteed Quality”)

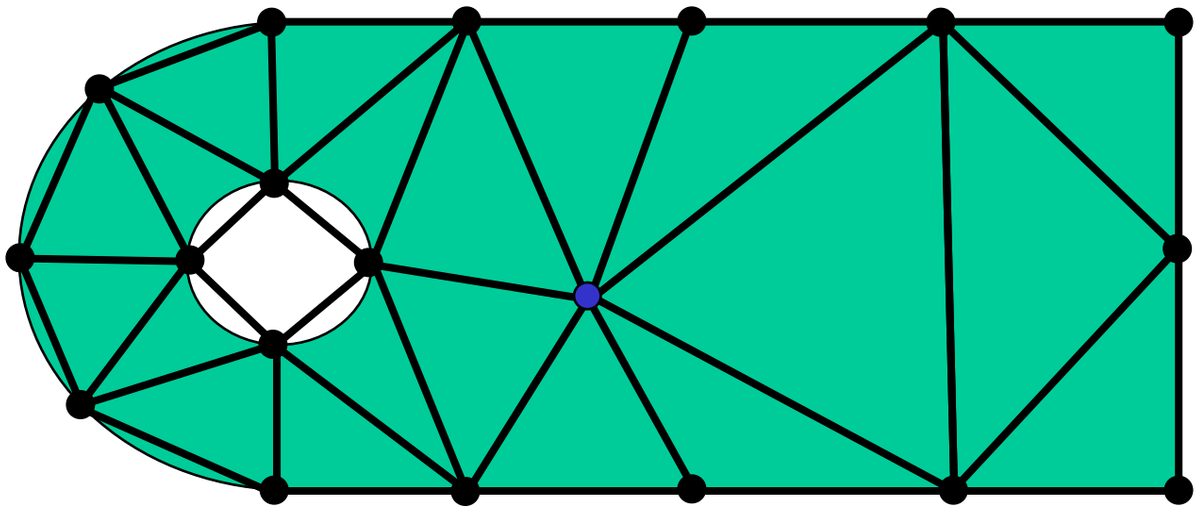
•Nodes introduced at triangle circumcenters

•Order of insertion based on minimum angle of any triangle

•Continues until minimum angle > predefined minimum ($\alpha \approx 30^\circ$)

Node Insertion (Chew,Ruppert,Shewchuk)

Delaunay



Circumcenter (“Guaranteed Quality”)

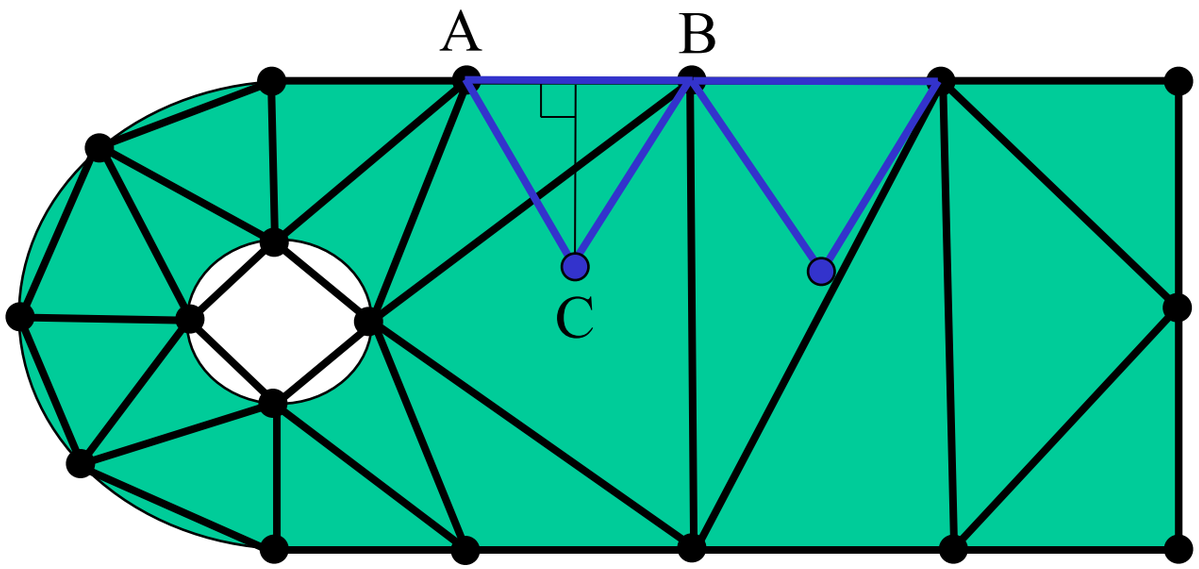
•Nodes introduced at triangle circumcenters

•Order of insertion based on minimum angle of any triangle

•Continues until minimum angle > predefined minimum ($\alpha \approx 30^\circ$)

Node Insertion (Chew,Ruppert,Shewchuk)

Delaunay



Advancing Front

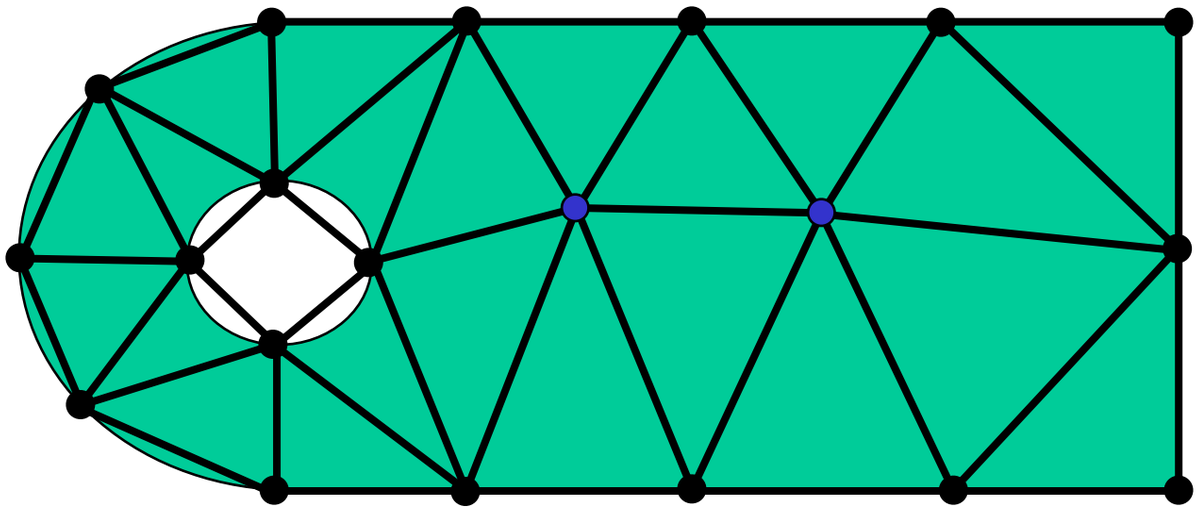
•“Front” structure maintained throughout

•Nodes introduced at ideal location from current front edge

(Marcum,95)

Node Insertion

Delaunay



Advancing Front

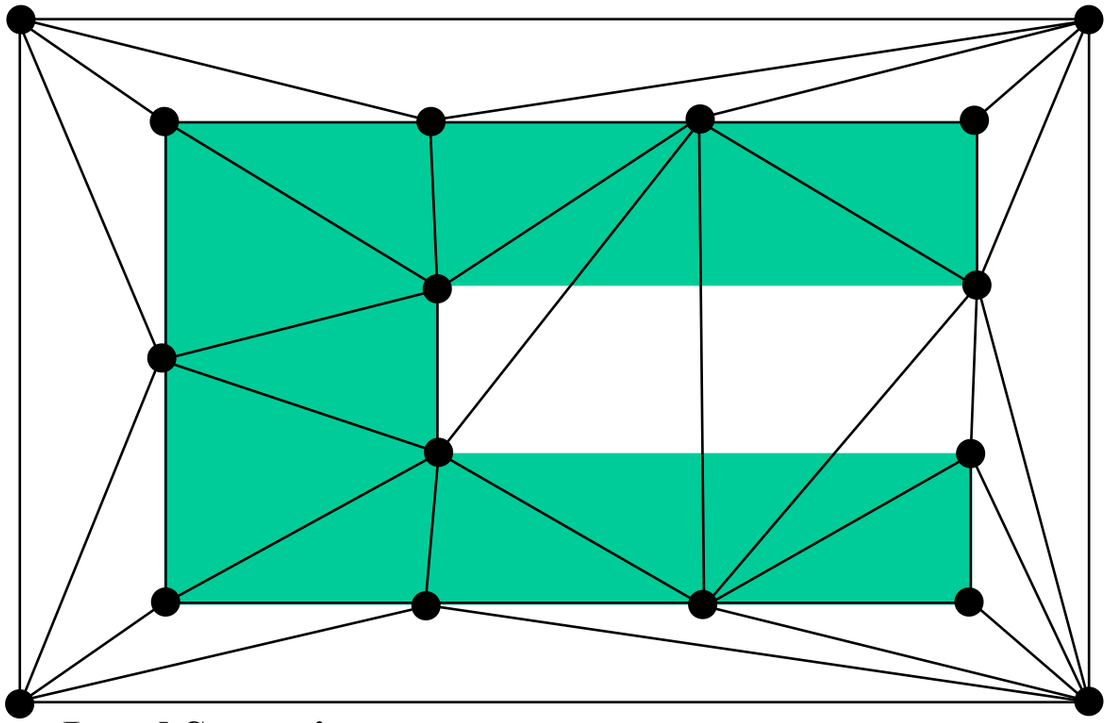
•“Front” structure maintained throughout

•Nodes introduced at ideal location from current front edge

(Marcum,95)

Node Insertion

Delaunay

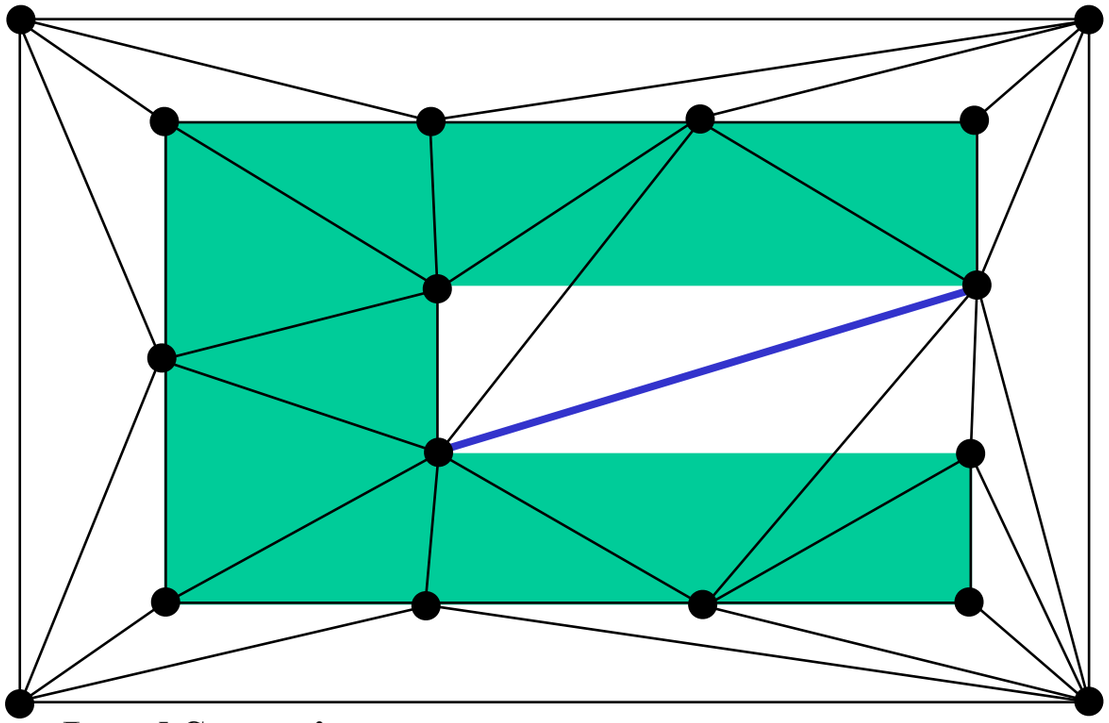


Local Swapping

•Edges swapped between adjacent pairs of triangles until boundary is maintained

Boundary Constrained

Delaunay

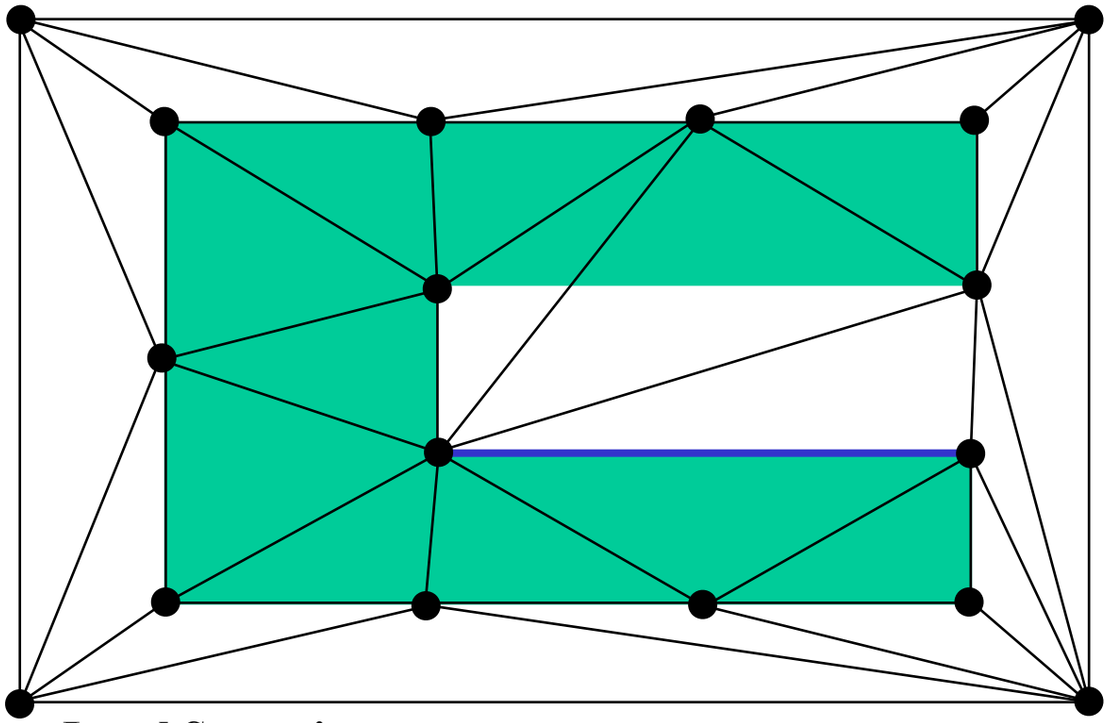


Local Swapping

•Edges swapped between adjacent pairs of triangles until boundary is maintained

Boundary Constrained

Delaunay

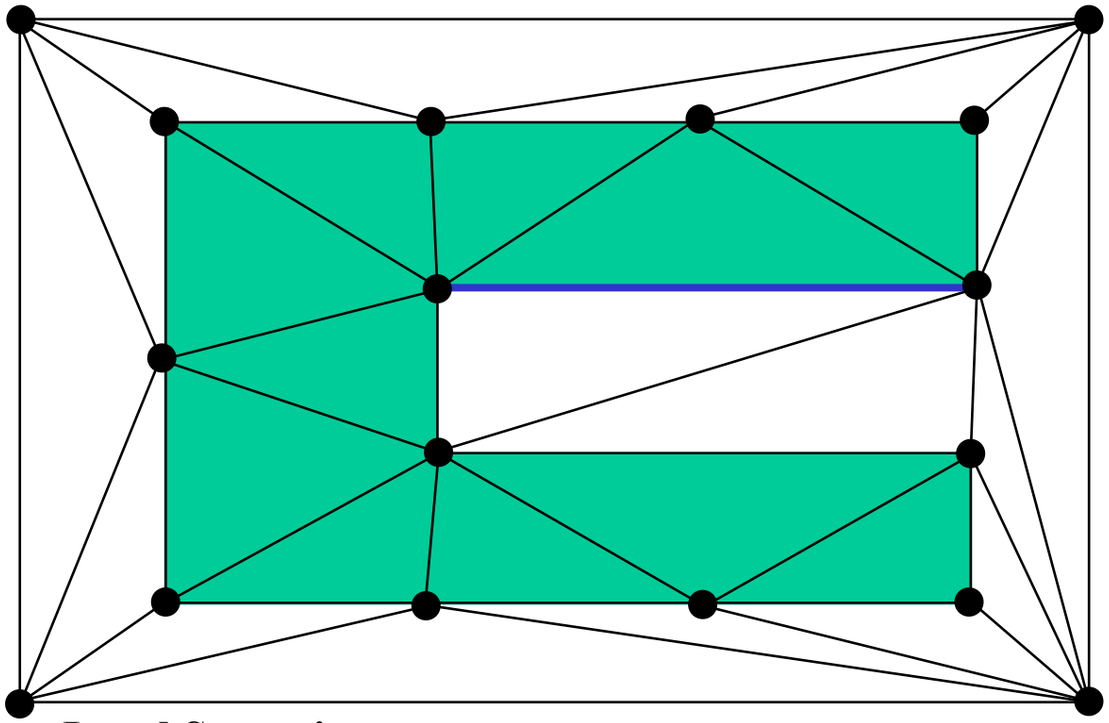


Local Swapping

•Edges swapped between adjacent pairs of triangles until boundary is maintained

Boundary Constrained

Delaunay

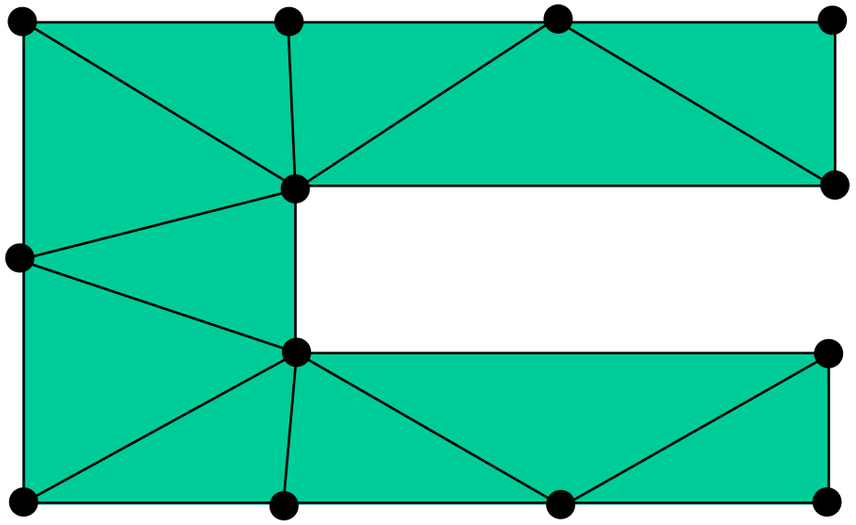


Local Swapping

•Edges swapped between adjacent pairs of triangles until boundary is maintained

Boundary Constrained

Delaunay



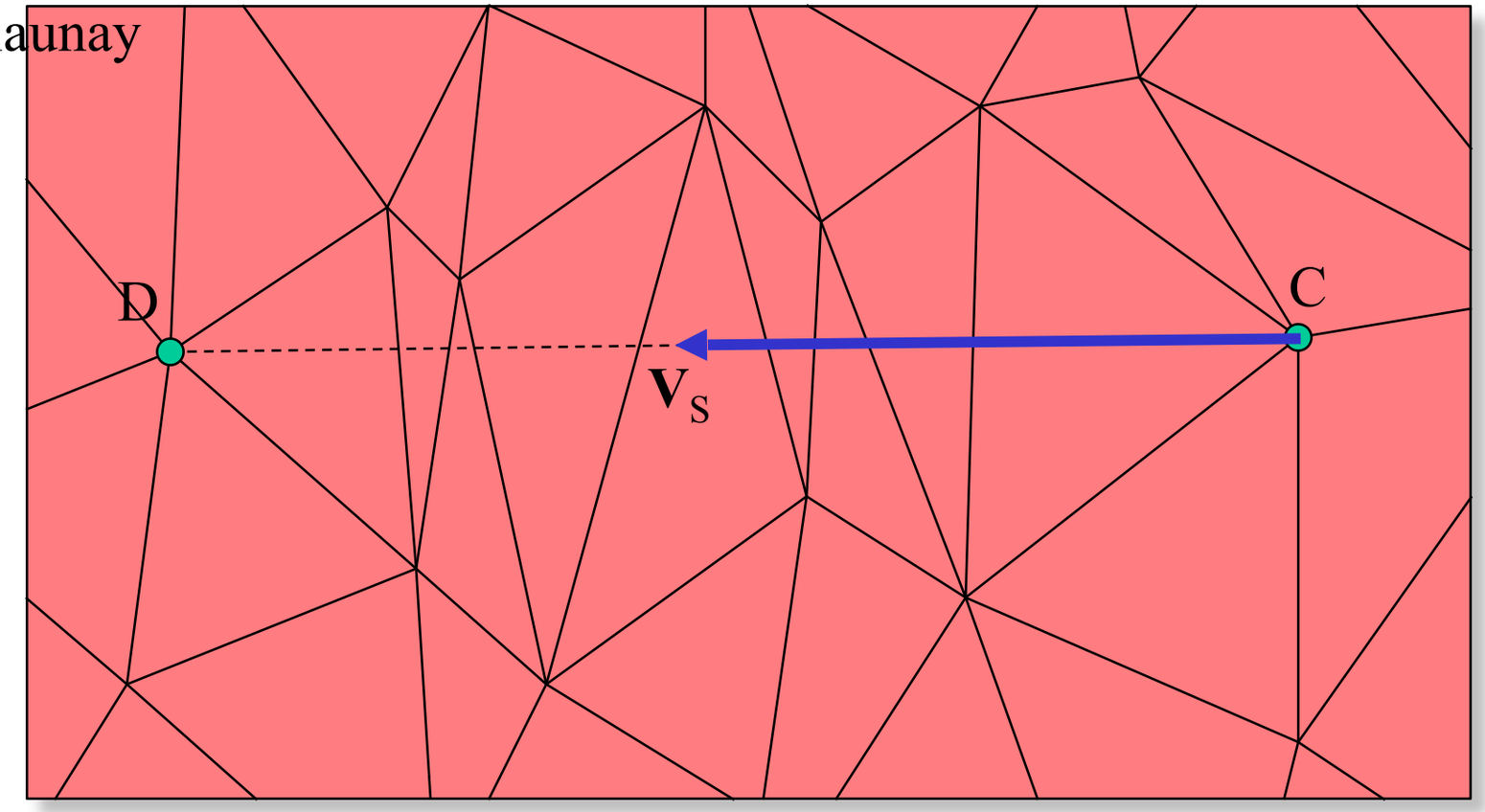
Local Swapping

•Edges swapped between adjacent pairs of triangles until boundary is maintained

(George,91)(Owen,99)

Boundary Constrained

Delaunay

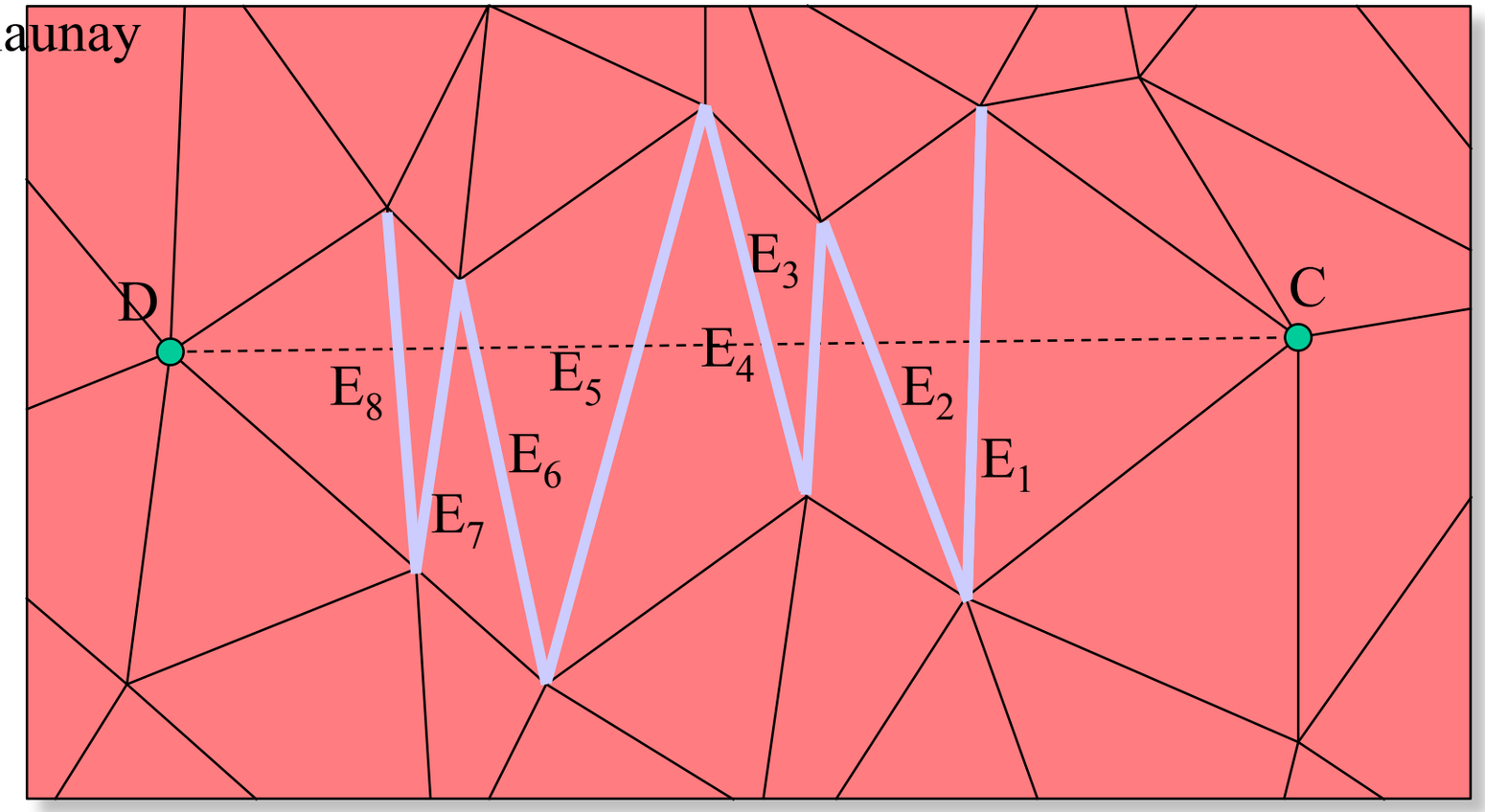


Local Swapping Example

- Recover edge CD at vector $\mathbf{V}_s$

Boundary Constrained

Delaunay

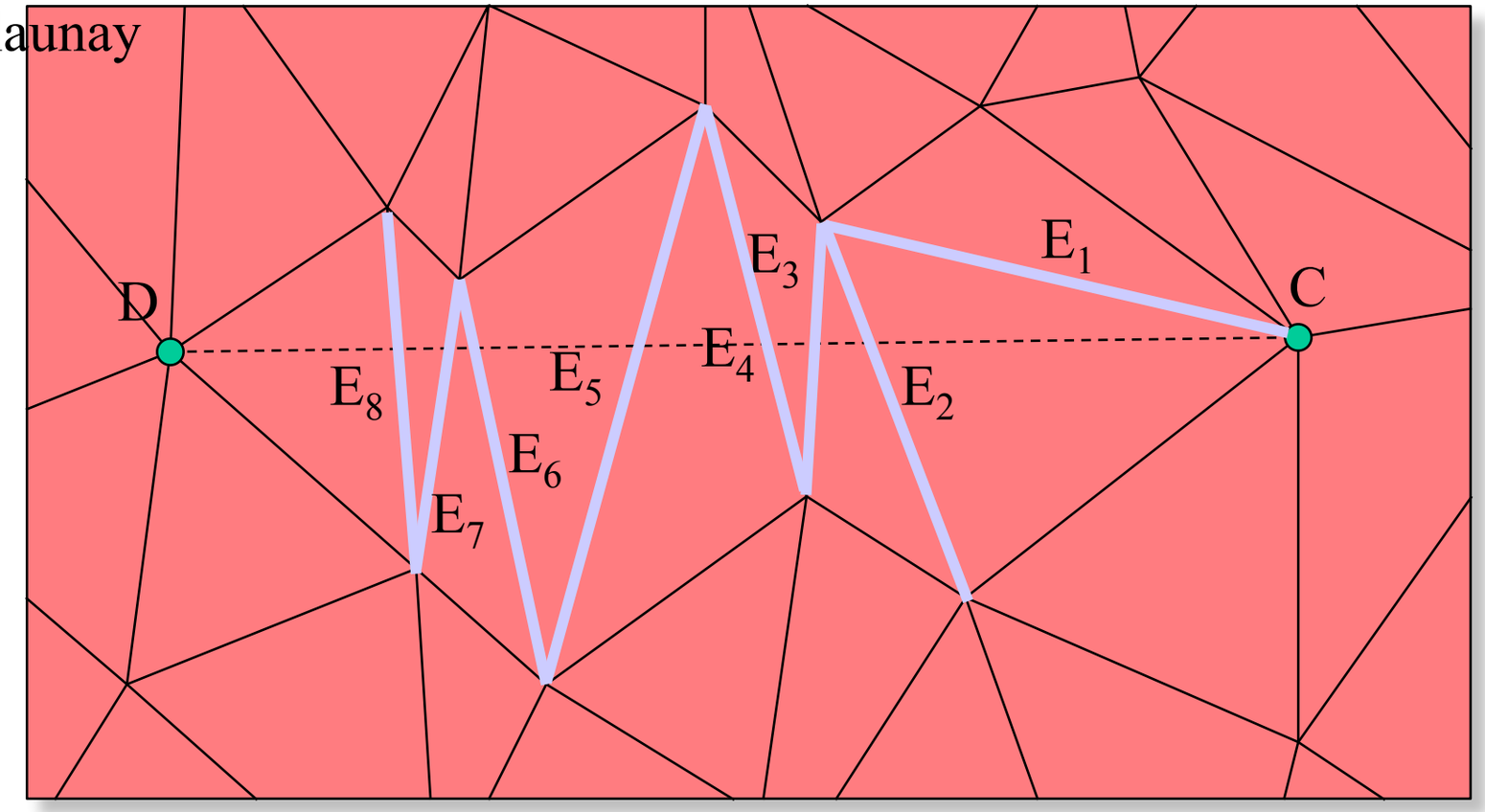


Local Swapping Example

•Make a list (queue) of all edges E_i, that intersect $\mathbf{V}_s$

Boundary Constrained

Delaunay

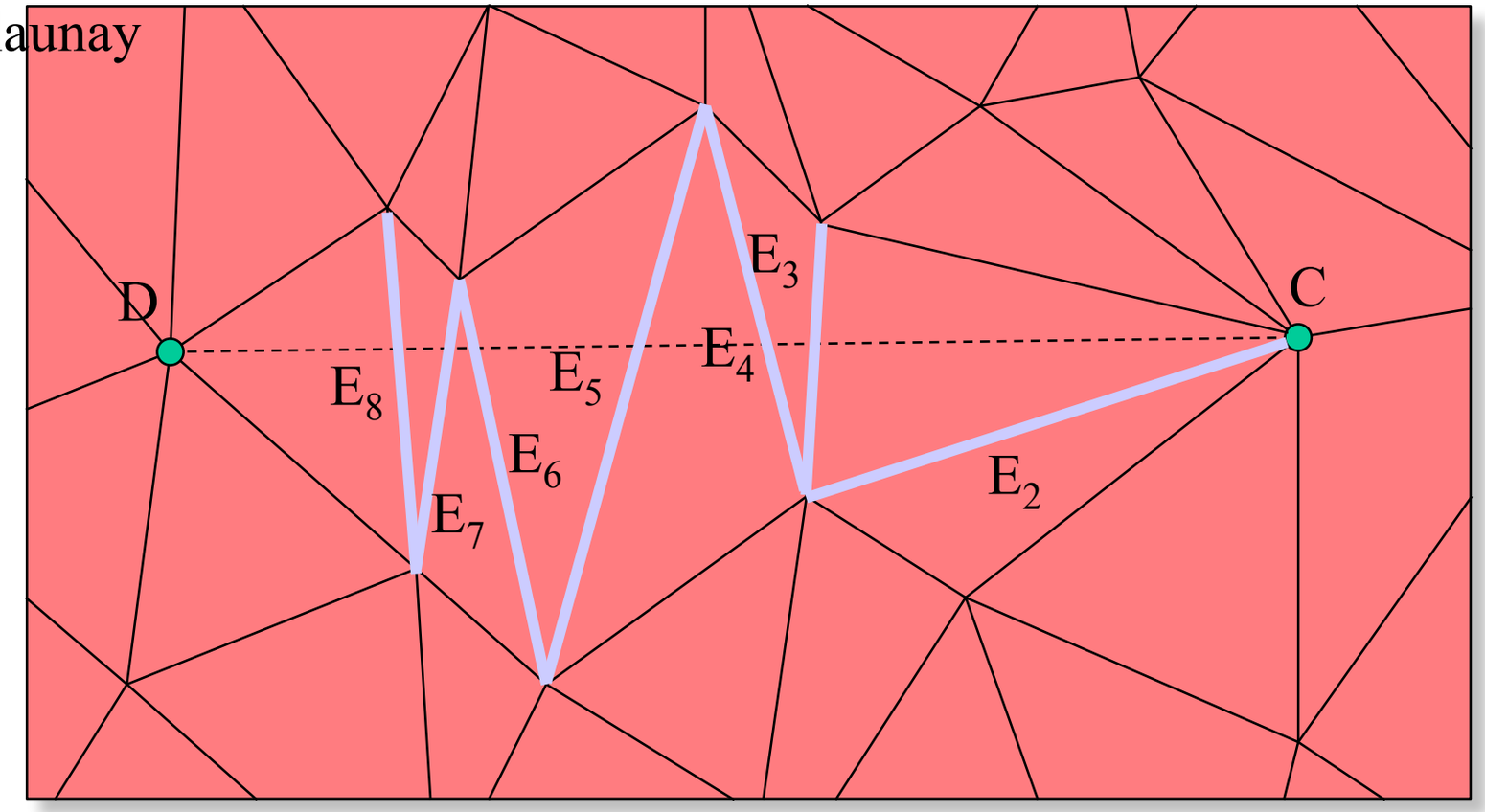


Local Swapping Example

•Swap the diagonal of adjacent triangle pairs for each edge in the list

Boundary Constrained

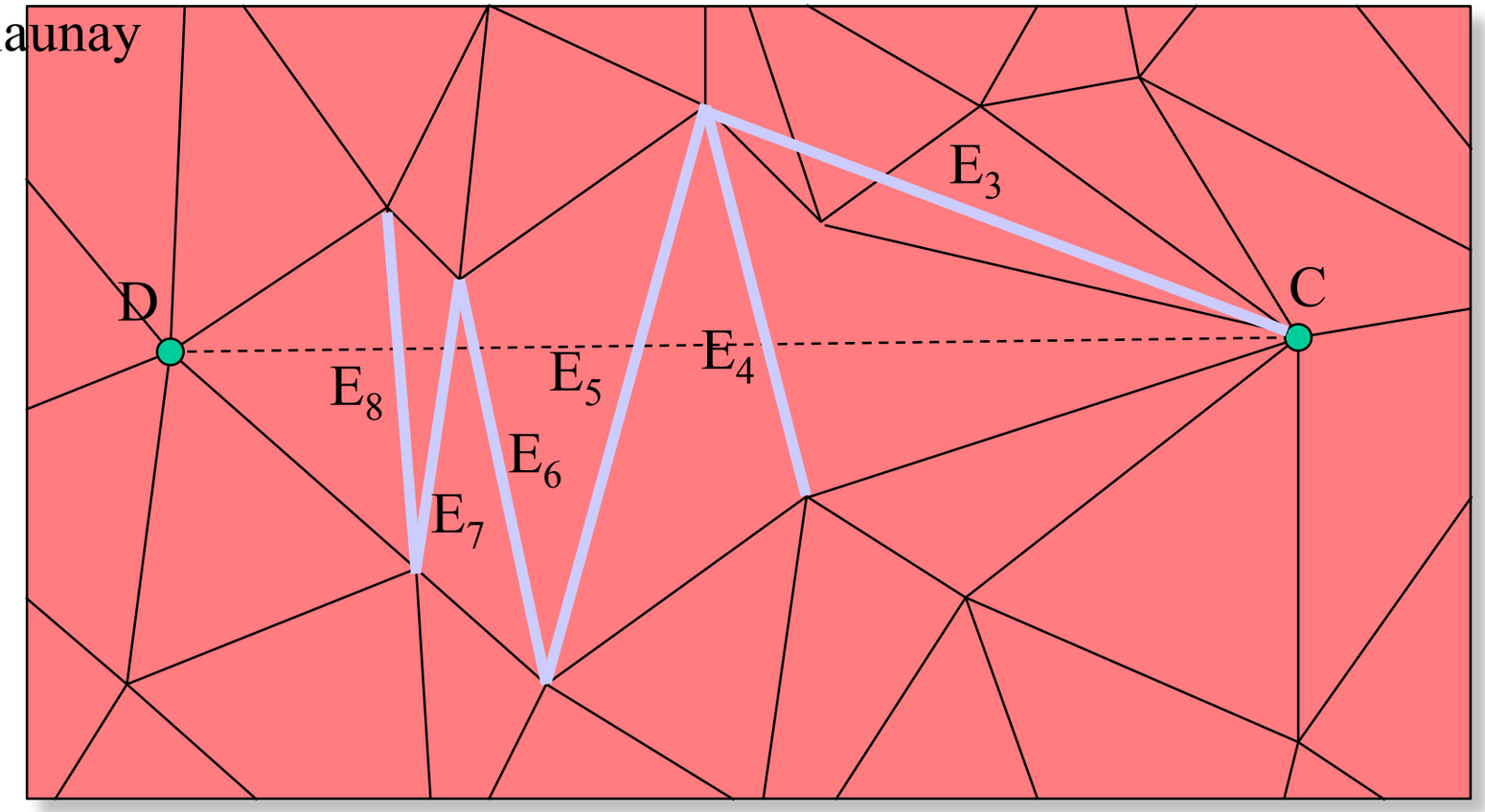
Delaunay



Local Swapping Example

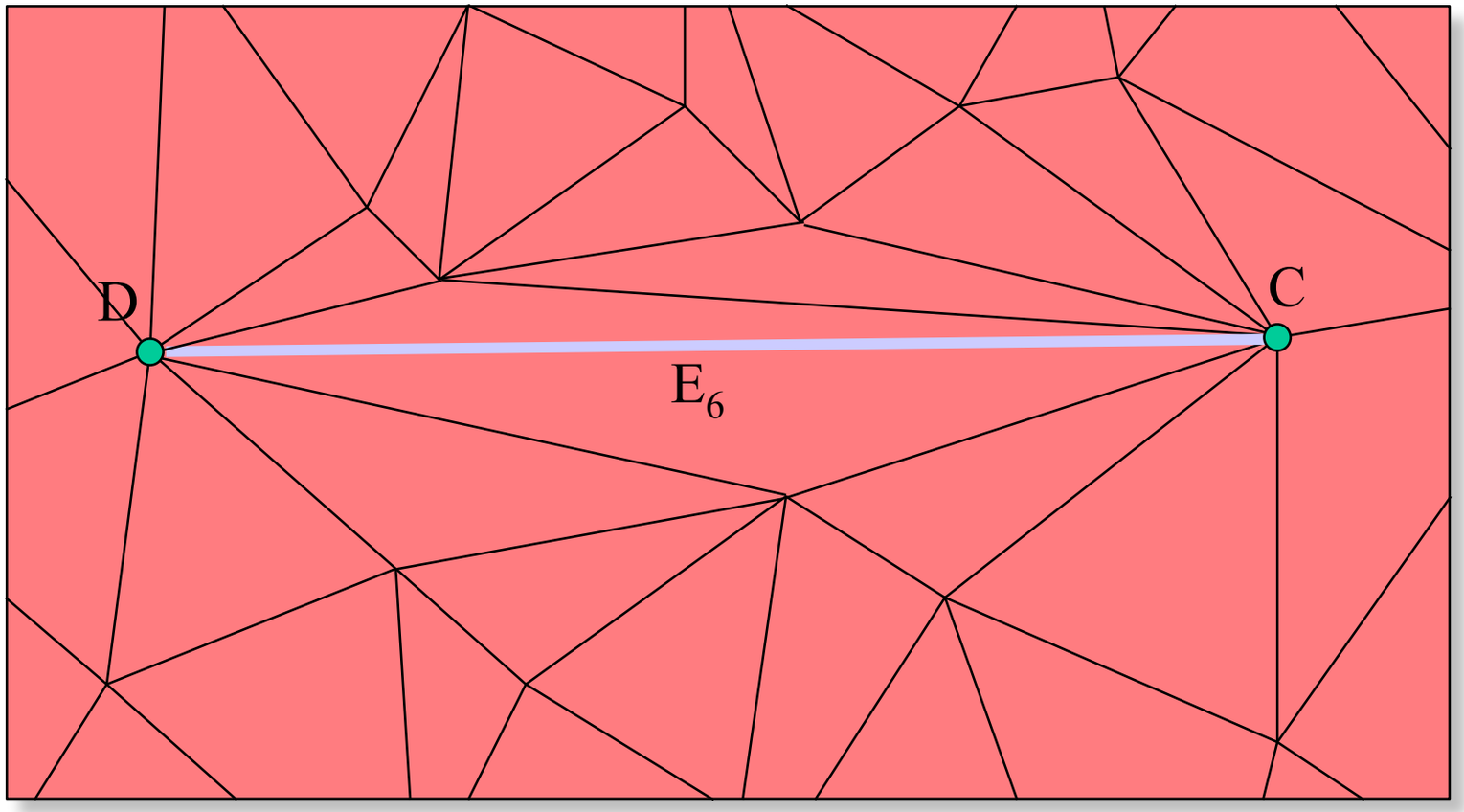
•Check that resulting swaps do not cause overlapping triangles. I they do, then place edge at the back of the queue and try again later

Delaunay



Local Swapping Example

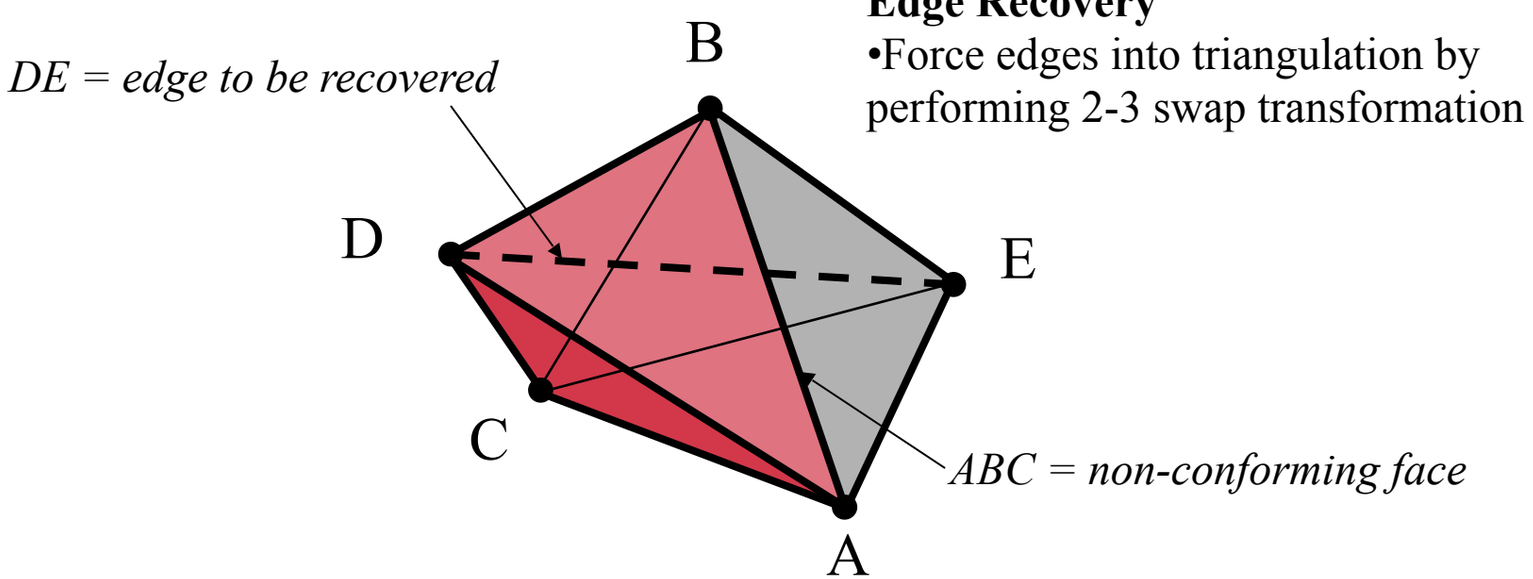
•Check that resulting swaps do not cause overlapping triangles. If they do, then place edge at the back of the queue and try again later



Local Swapping Example

- Final swap will recover the desired edge.
- Resulting triangle quality may be poor if multiple swaps were necessary
- Does not maintain Delaunay criterion!

Delaunay



Edge Recovery
•Force edges into triangulation by performing 2-3 swap transformation

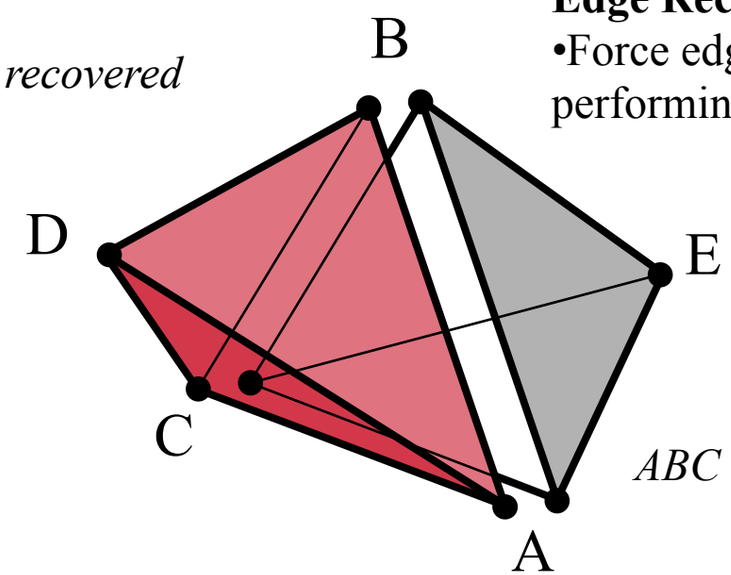
3D Local Swapping
•Requires both boundary *edge* recovery and boundary *face* recovery

(George,91;99)(Owen,00)

Boundary Constrained

Delaunay

DE = edge to be recovered



Edge Recovery
•Force edges into triangulation by performing 2-3 swap transformation

2-3 Swap

ABCE
ACBD

ABC = non-conforming face

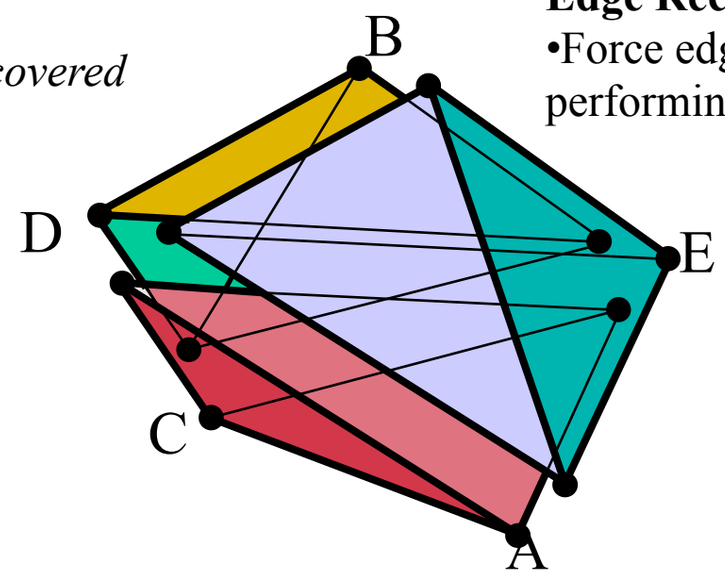
3D Local Swapping
•Requires both boundary *edge* recovery and boundary *face* recovery

(George,91;99)(Owen,00)

Boundary Constrained

Delaunay

DE = edge recovered



Edge Recovery
•Force edges into triangulation by performing 2-3 swap transformation

2-3 Swap

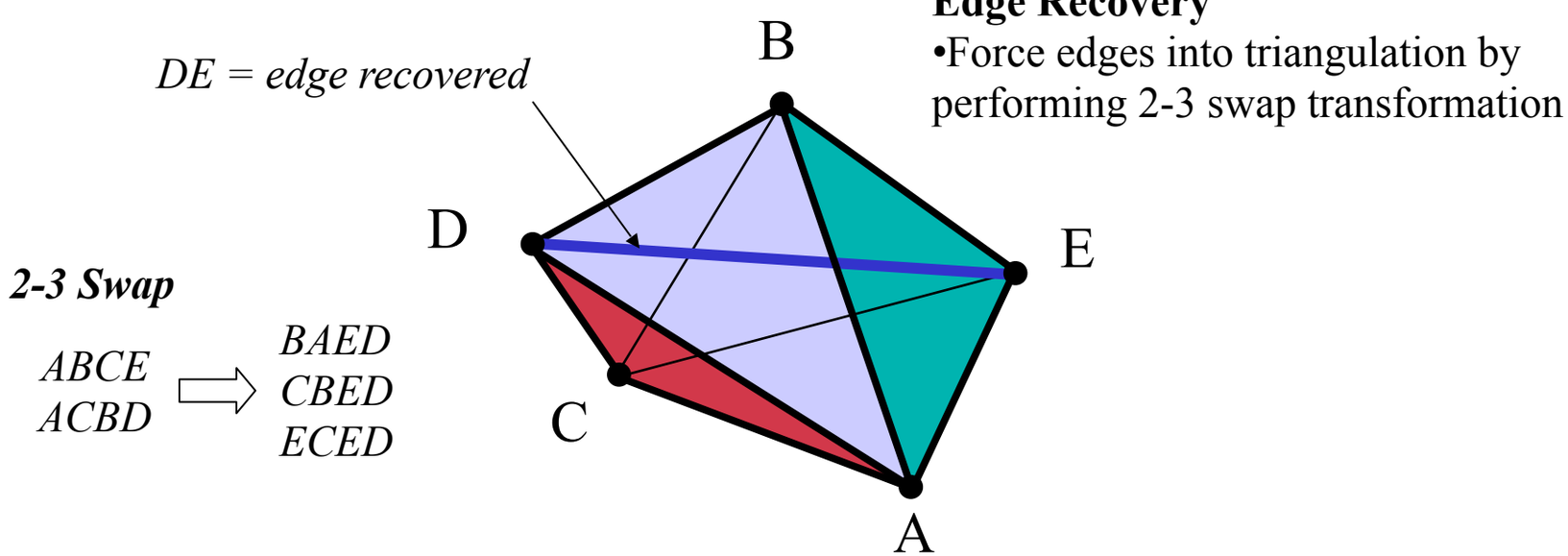
ABCE
ACBD
⇨
BAED
CBED
ACED

3D Local Swapping
•Requires both boundary *edge* recovery and boundary *face* recovery

(George,91;99)(Owen,00)

Boundary Constrained

Delaunay



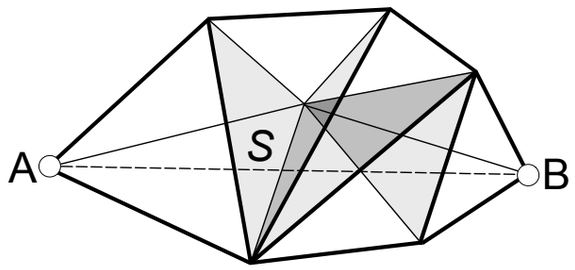
Edge Recovery
•Force edges into triangulation by performing 2-3 swap transformation

3D Local Swapping
•Requires both boundary *edge* recovery and boundary *face* recovery

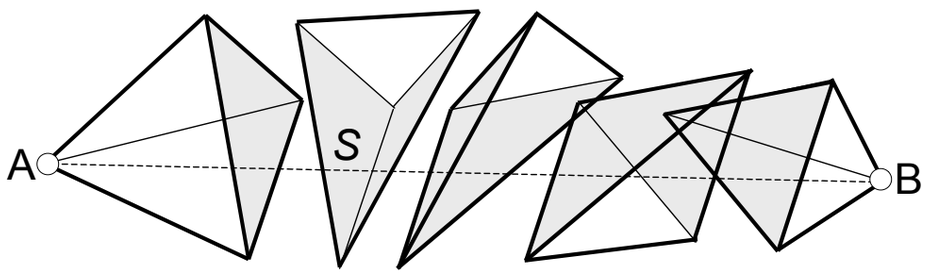
(George,91;99)(Owen,00)

Boundary Constrained

Delaunay



Edge AB to be recovered



Exploded view of tets intersected by AB

3D Edge Recovery

- Form queue of faces through which edge AB will pass
- Perform 2-3 swap transformations on all faces in the list
- If overlapping tets result, place back on queue and try again later
- If still cannot recover edge, then insert “steiner” point