Algorithmical Geometry: Computation of Delaunay Triangulations Using a Divide-and-Conquer Algorithm

Markus Pawellek

January 20, 2022

Outline

Related Work

Mathematical Preliminaries

Geometric Primitives

Quad-Edge Data Structure

Algorithm

Implementation Notes

Conclusions

Related Work

Educational Problems:

Related Work

Educational Problems:

- Many Resources

Related Work

Educational Problems:

- Many Resources
- Duality to Voronoi Diagrams

Related Work

Educational Problems:

- Many Resources
- Duality to Voronoi Diagrams
- Multiple Algorithm Types:
  Incremental, Sweepline, Divide-and-Conquer

Related Work

Educational Problems:

- Many Resources
- Duality to Voronoi Diagrams
- Multiple Algorithm Types:
  Incremental, Sweepline, Divide-and-Conquer
- Varying Data Structures

Related Work: References

Related Work: References

1980 Lee and Schachter, “Two Algorithms for Constructing a Delaunay Triangulation”

Related Work: References

1980 Lee and Schachter, “Two Algorithms for Constructing a Delaunay Triangulation”

1985 Guibas and Stolfi, “Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams”

Related Work: References

1980 Lee and Schachter, “Two Algorithms for Constructing a Delaunay Triangulation”

1985 Guibas and Stolfi, “Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams”

1987 Dwyer, “A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations”

Related Work: References

1980 Lee and Schachter, "Two Algorithms for Constructing a Delaunay Triangulation"

1985 Guibas and Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams"

1987 Dwyer, "A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations"

1996 Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator"

Related Work: References

1980 Lee and Schachter, “Two Algorithms for Constructing a Delaunay Triangulation”

1985 Guibas and Stolfi, “Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams”

1987 Dwyer, “A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations”

1996 Shewchuk, “Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator”

2014 Fuetterling, Lojewski, and Pfreundt, “High-Performance Delaunay Triangulation for Many-Core Computers”

Related Work: References

1980 Lee and Schachter, "Two Algorithms for Constructing a Delaunay Triangulation"

1985 **Guibas and Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams"**

1987 Dwyer, "A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations"

1996 Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator"

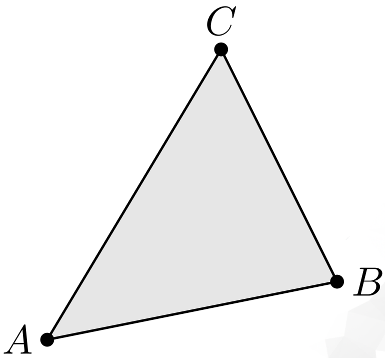
2014 Fuetterling, Lojewski, and Pfreundt, "High-Performance Delaunay Triangulation for Many-Core Computers"

Mathematical Preliminaries

Mathematical Preliminaries: Triangle and Circumcircle

Triangle

$A, B, C \in \mathbb{R}^2$ affinely independent define vertices of a triangle.



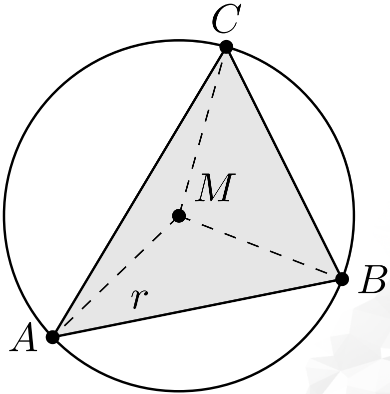
Mathematical Preliminaries: Triangle and Circumcircle

Triangle

$A, B, C \in \mathbb{R}^2$ affinely independent define vertices of a triangle.

Circumcircle

Circle that intersects with all vertices of the triangle.

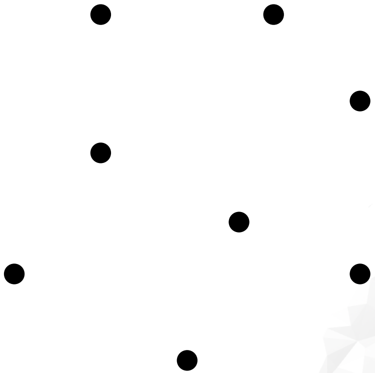


Mathematical Preliminaries: Point Set

Point Set

$\mathcal{V} \subset \mathbb{R}^2$ finite, $\#\mathcal{V} \geq 3$,

affinely span $\mathbb{R}^2$



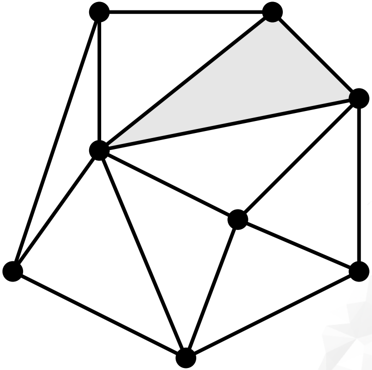
Mathematical Preliminaries: Triangulation

Point Set

$\mathcal{V} \subset \mathbb{R}^2$ finite, $\#\mathcal{V} \geq 3$,
affinely span $\mathbb{R}^2$

Triangulation

Planar straight-line graph over $\mathcal{V}$ such that its edges form a maximal subset of non-crossing edges.



Mathematical Preliminaries: Delaunay Triangulation

Point Set

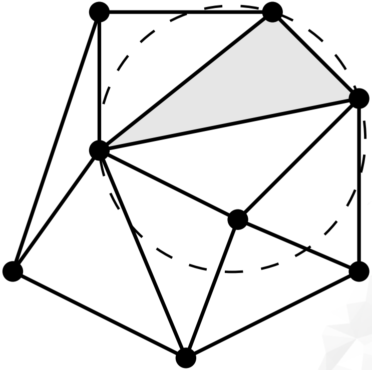
$\mathcal{V} \subset \mathbb{R}^2$ finite, $\#\mathcal{V} \geq 3$,
affinely span $\mathbb{R}^2$

Triangulation

Planar straight-line graph over $\mathcal{V}$ such that its edges form a maximal subset of non-crossing edges.

Delaunay Triangulation

Circumcircle of any triangle contains no other points of $\mathcal{V}$.



Mathematical Preliminaries: Delaunay Triangulation

Point Set

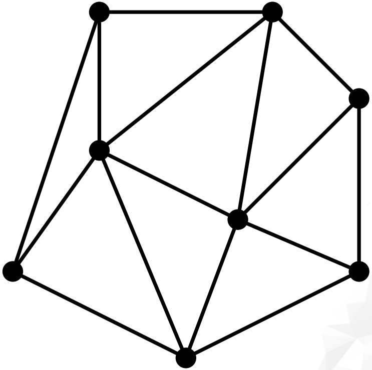
$\mathcal{V} \subset \mathbb{R}^2$ finite, $\#\mathcal{V} \geq 3$,
affinely span $\mathbb{R}^2$

Triangulation

Planar straight-line graph over $\mathcal{V}$ such that its edges form a maximal subset of non-crossing edges.

Delaunay Triangulation

Circumcircle of any triangle contains no other points of $\mathcal{V}$.



Point Set

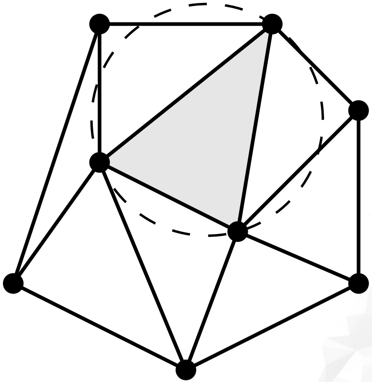
$\mathcal{V} \subset \mathbb{R}^2$ finite, $\#\mathcal{V} \geq 3$,
affinely span $\mathbb{R}^2$

Triangulation

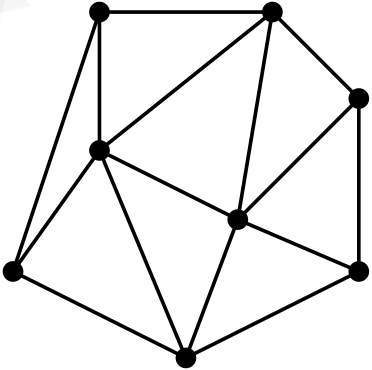
Planar straight-line graph over $\mathcal{V}$ such that its edges form a maximal subset of non-crossing edges.

Delaunay Triangulation

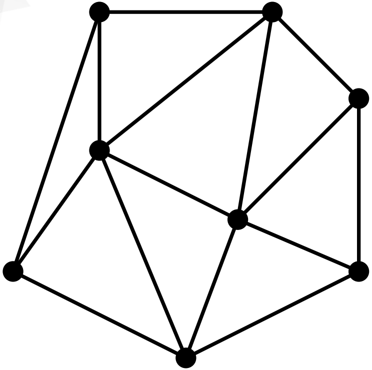
Circumcircle of any triangle contains no other points of $\mathcal{V}$.



Mathematical Preliminaries: Properties

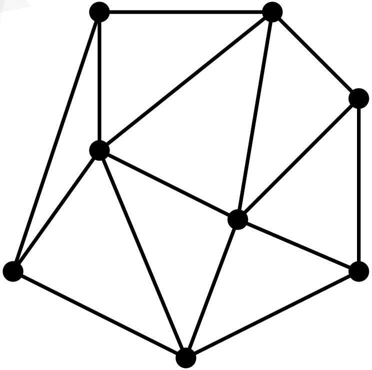


Mathematical Preliminaries: Properties



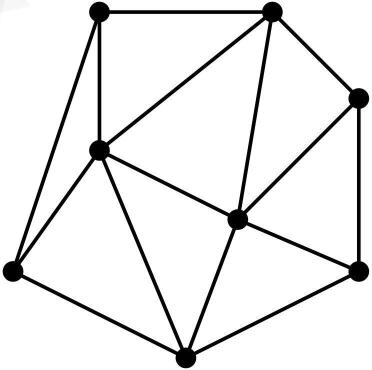
- Existence is guaranteed

Mathematical Preliminaries: Properties



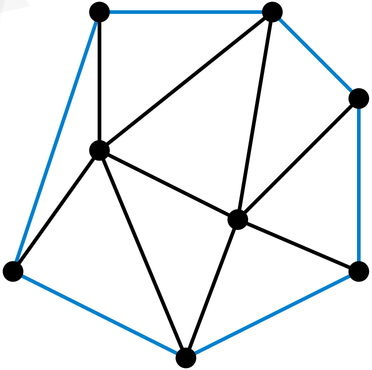
- Existence is guaranteed
- Unique if there are no four points that are cocircular

Mathematical Preliminaries: Properties



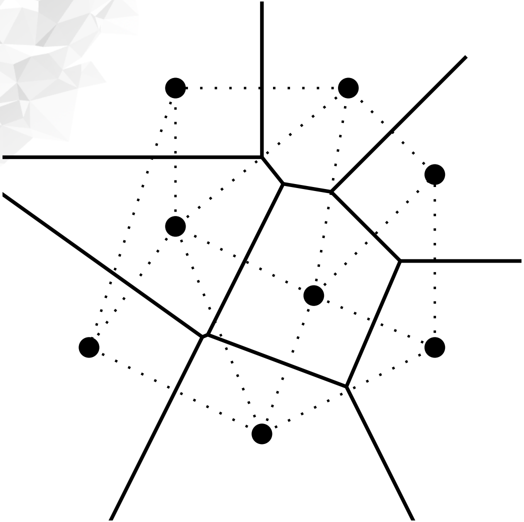
- Existence is guaranteed
- Unique if there are no four points that are cocircular
- Optimality: maximization of the minimum angle of all angles

Mathematical Preliminaries: Properties



- Existence is guaranteed
- Unique if there are no four points that are cocircular
- Optimality: maximization of the minimum angle of all angles
- Convex hull is contained

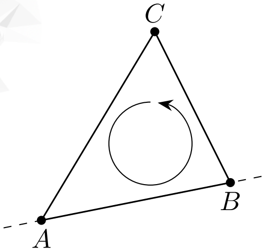
Mathematical Preliminaries: Properties



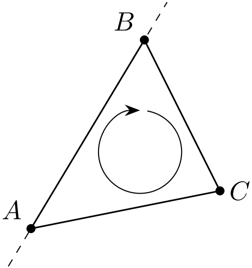
- Existence is guaranteed
- Unique if there are no four points that are cocircular
- Optimality: maximization of the minimum angle of all angles
- Convex hull is contained
- Dual of Voronoi diagram

Geometric Primitives

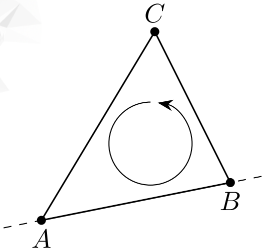
Geometric Primitives: Triangle Orientation



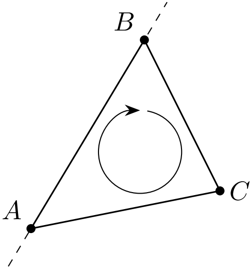
$\longleftrightarrow$



Geometric Primitives: Triangle Orientation

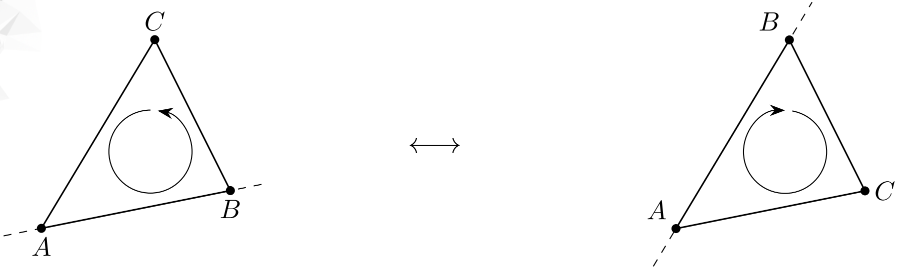


$\longleftrightarrow$



Counterclockwise Orientation $\iff$ C is left of $\overline{AB}$

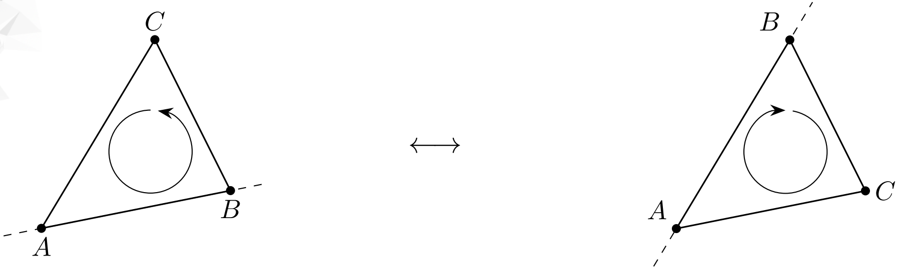
Geometric Primitives: Triangle Orientation



Counterclockwise Orientation $\iff$ C is left of $\overline{AB}$

$$0 < \begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix}$$

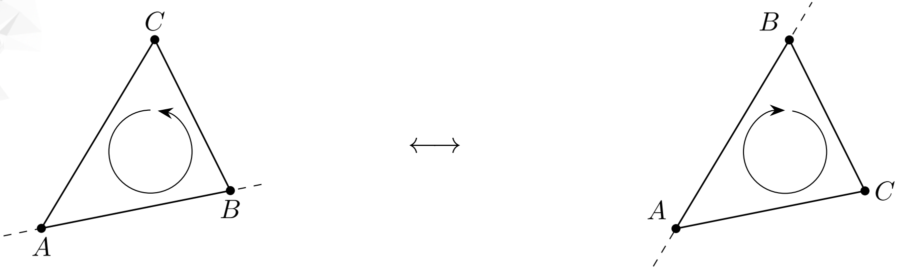
Geometric Primitives: Triangle Orientation



Counterclockwise Orientation $\iff$ C is left of $\overline{AB}$

$$0 < \begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix} = \begin{vmatrix} B_x - A_x & B_y - A_y \\ C_x - A_x & C_y - A_y \end{vmatrix}$$

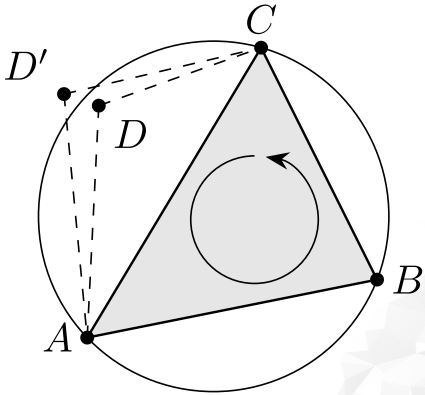
Geometric Primitives: Triangle Orientation



Counterclockwise Orientation $\iff$ C is left of $\overline{AB}$

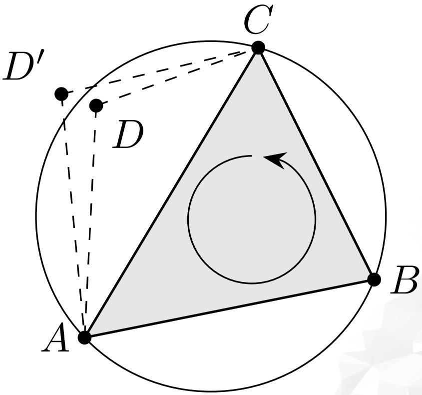
$$0 < \begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix} = \begin{vmatrix} B_x - A_x & B_y - A_y \\ C_x - A_x & C_y - A_y \end{vmatrix} = \det \begin{pmatrix} B - A & C - A \end{pmatrix}$$

Geometric Primitives: Inside Circumcircle



Geometric Primitives: Inside Circumcircle

$$0 < \begin{vmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_y^2 & 1 \\ C_x & C_y & C_x^2 + C_y^2 & 1 \\ D_x & D_y & D_x^2 + D_y^2 & 1 \end{vmatrix}$$

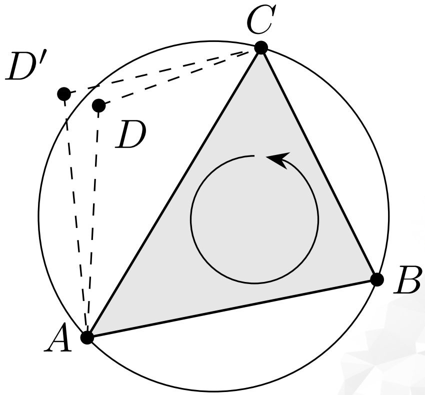


Geometric Primitives: Inside Circumcircle

$$0 < \begin{vmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_y^2 & 1 \\ C_x & C_y & C_x^2 + C_y^2 & 1 \\ D_x & D_y & D_x^2 + D_y^2 & 1 \end{vmatrix}$$

$$= \left\langle x, \operatorname{adj} \begin{pmatrix} u & v \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \|u\|^2 \\ \|v\|^2 \end{pmatrix} \right\rangle - \det \begin{pmatrix} u & v \end{pmatrix} \|x\|^2$$

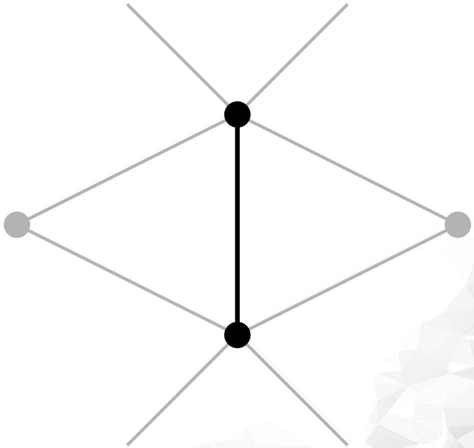
$u := B - A, \quad v := C - A, \quad x := D - A$



Quad-Edge Data Structure

Quad-Edge Data Structure: Scheme

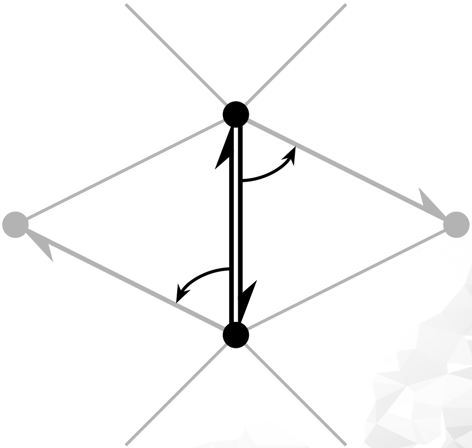
Edge-Based List-Like Data Structure
for Storing Neighbor Information:



Quad-Edge Data Structure: Scheme

Edge-Based List-Like Data Structure
for Storing Neighbor Information:

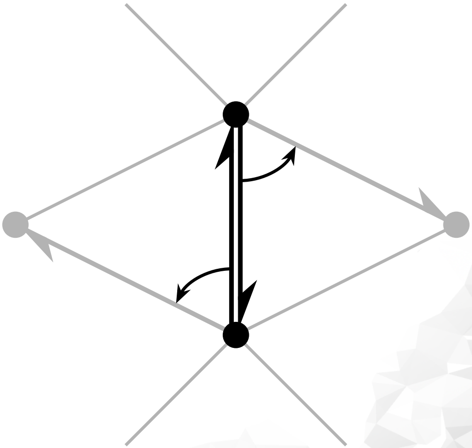
- Directed edges for vertices



Quad-Edge Data Structure: Scheme

Edge-Based List-Like Data Structure
for Storing Neighbor Information:

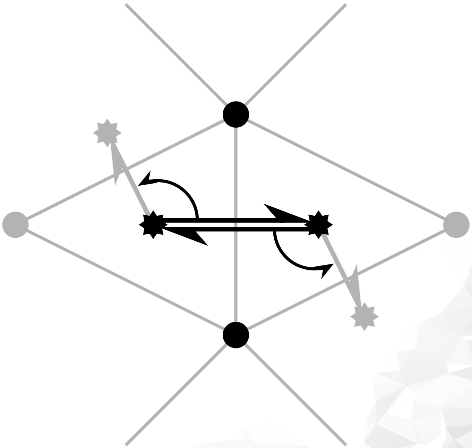
- Directed edges for vertices
- Pointer to ccw. next directed edge with same origin vertex



Quad-Edge Data Structure: Scheme

Edge-Based List-Like Data Structure
for Storing Neighbor Information:

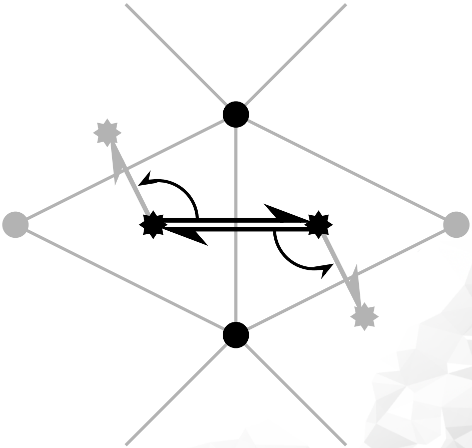
- Directed edges for vertices
- Pointer to ccw. next directed edge with same origin vertex
- Directed dual edges for adjacent faces



Quad-Edge Data Structure: Scheme

Edge-Based List-Like Data Structure
for Storing Neighbor Information:

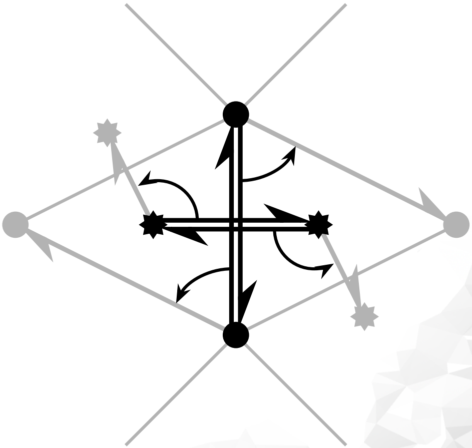
- Directed edges for vertices
- Pointer to ccw. next directed edge with same origin vertex
- Directed dual edges for adjacent faces
- Pointer to ccw. next directed dual edge with same origin face



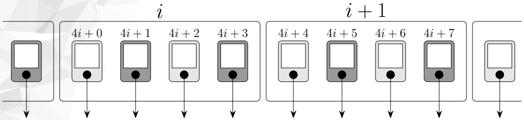
Quad-Edge Data Structure: Scheme

Edge-Based List-Like Data Structure for Storing Neighbor Information:

- Directed edges for vertices
- Pointer to ccw. next directed edge with same origin vertex
- Directed dual edges for adjacent faces
- Pointer to ccw. next directed dual edge with same origin face



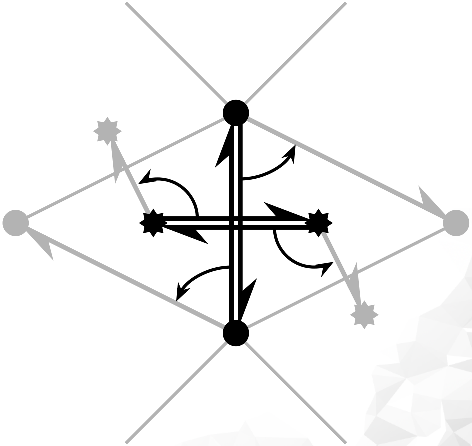
Quad-Edge Data Structure: Implementation



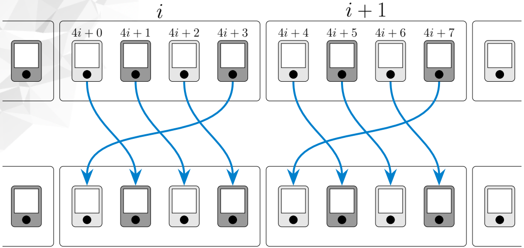

```
struct edge {
  size_t next;
  size_t data;
};

struct quad_edge {
  edge data[4];
};

vector<vertex>    vertices{};
vector<quad_edge> quad_edges{};
vector<size_t>    free_edges{};
```

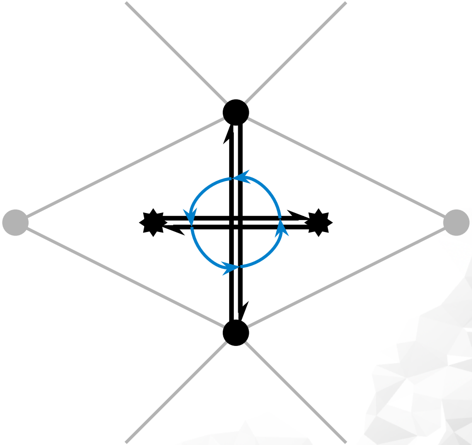


Quad-Edge Data Structure: Implementation

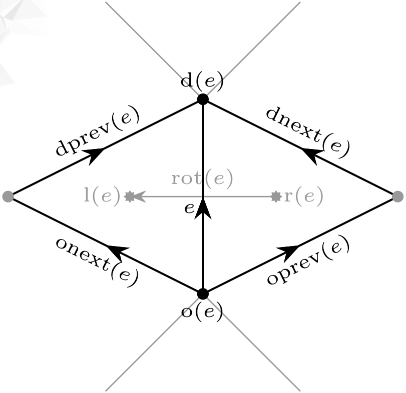


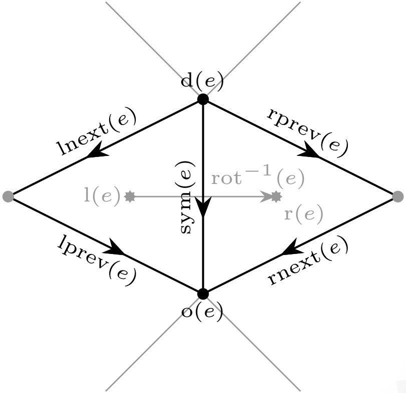
$$\mathrm{rot}\colon \mathbb{N}_0 \to \mathbb{N}_0$$

$$\mathrm{rot}(x) = 4 \cdot \left\lfloor \frac{x}{4} \right\rfloor + (x + 1 \mod 4)$$

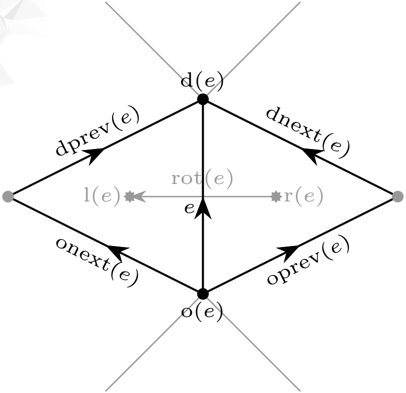


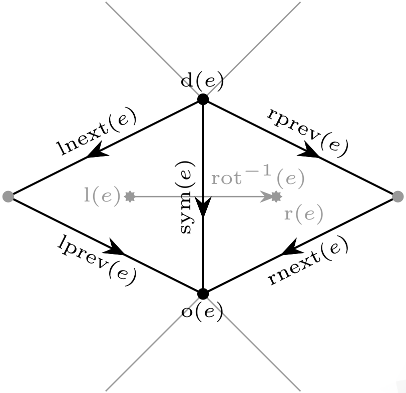
Quad-Edge Data Structure: Edge Functions and Operators





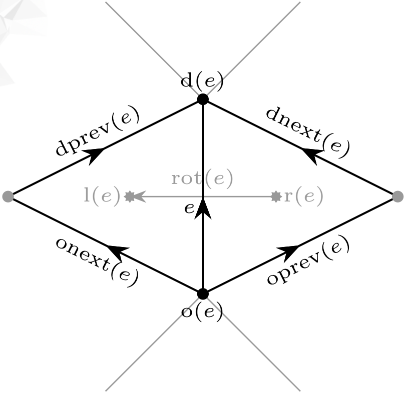
Quad-Edge Data Structure: Edge Functions and Operators

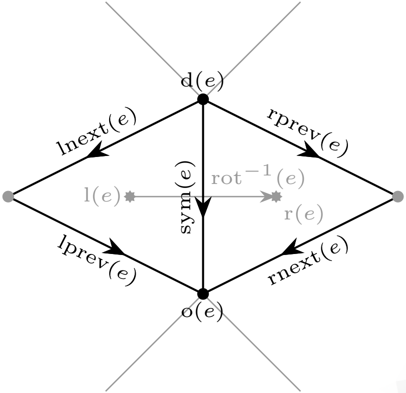




- Create a new edge

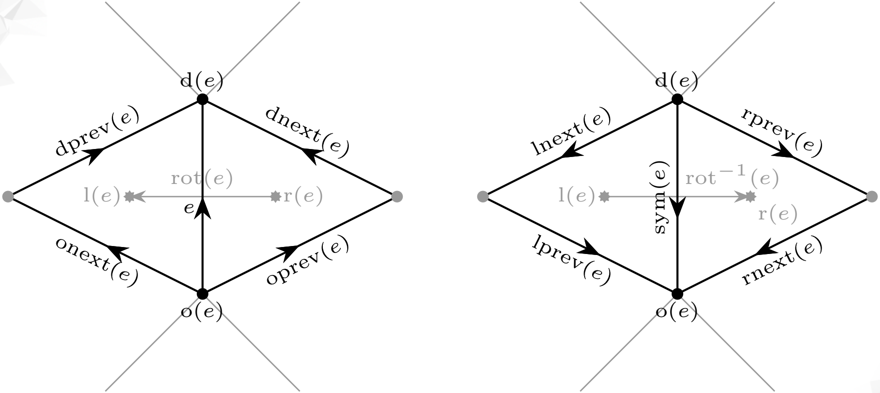
Quad-Edge Data Structure: Edge Functions and Operators





- Create a new edge
- Delete existing edge

Quad-Edge Data Structure: Edge Functions and Operators



- Create a new edge
- Delete existing edge
- Connect points by a new edge

Algorithm

Algorithm: Overview

Algorithm: Overview

Triangulation Algorithm

Algorithm: Overview

Triangulation Algorithm

1. Sort the given point set by increasing x coordinate.

Algorithm: Overview

Triangulation Algorithm

1. Sort the given point set by increasing x coordinate.
2. Triangulate sorted point set.

Algorithm: Overview

Triangulation Algorithm

1. Sort the given point set by increasing x coordinate.
2. Triangulate sorted point set.

Subroutine: Triangulate

Algorithm: Overview

Triangulation Algorithm

1. Sort the given point set by increasing x coordinate.
2. Triangulate sorted point set.

Subroutine: Triangulate

1. If point count is smaller than four, make edge or triangle and return.

Algorithm: Overview

Triangulation Algorithm

1. Sort the given point set by increasing x coordinate.
2. Triangulate sorted point set.

Subroutine: Triangulate

1. If point count is smaller than four, make edge or triangle and return.
2. Split point set into left and right half.

Algorithm: Overview

Triangulation Algorithm

1. Sort the given point set by increasing x coordinate.
2. Triangulate sorted point set.

Subroutine: Triangulate

1. If point count is smaller than four, make edge or triangle and return.
2. Split point set into left and right half.
3. Triangulate left and right half.

Algorithm: Overview

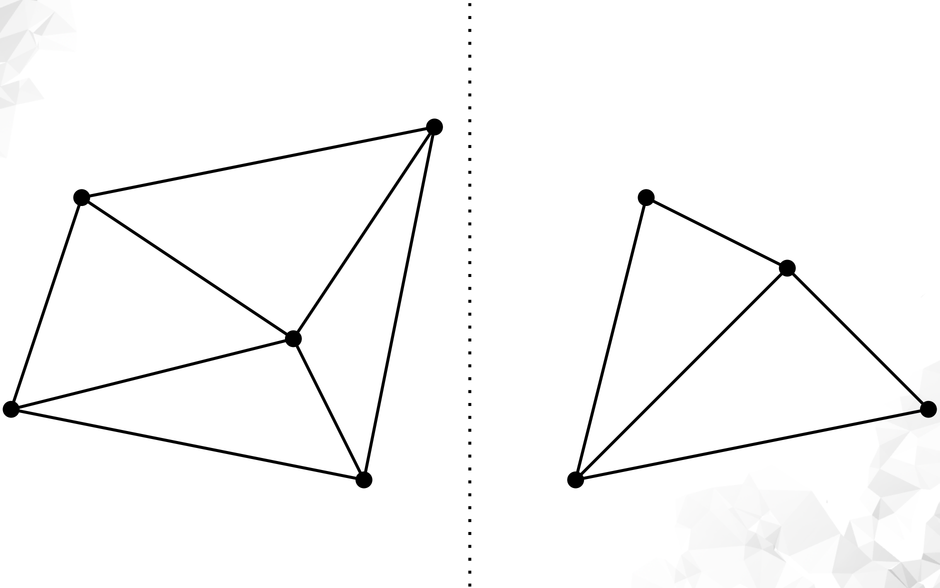
Triangulation Algorithm

1. Sort the given point set by increasing x coordinate.
2. Triangulate sorted point set.

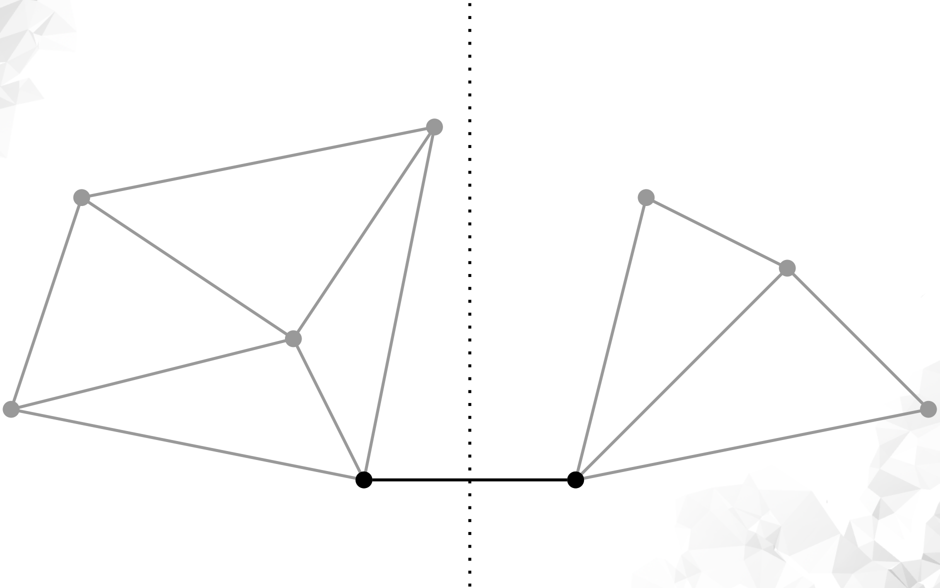
Subroutine: Triangulate

1. If point count is smaller than four, make edge or triangle and return.
2. Split point set into left and right half.
3. Triangulate left and right half.
4. Merge left and right triangulations.

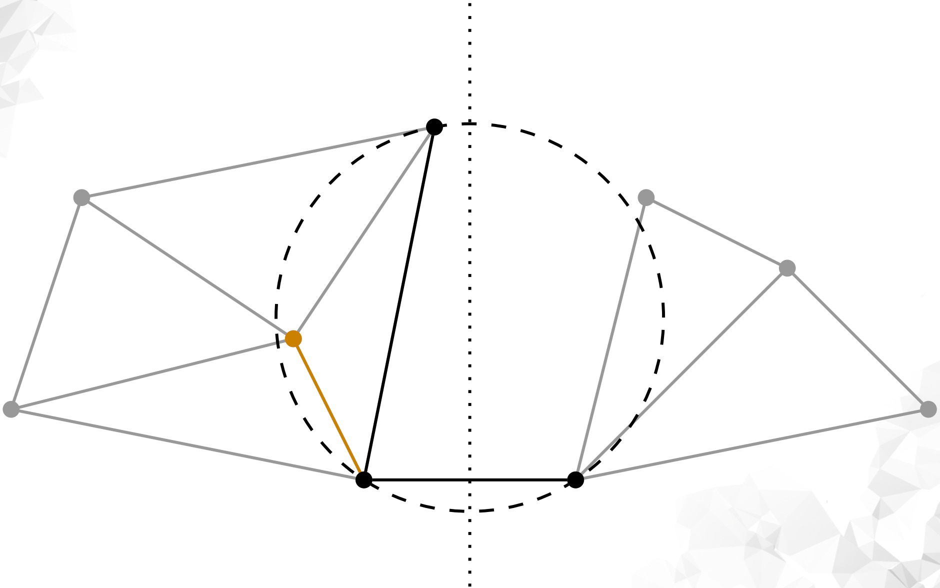
Algorithm: Merge Triangulations Example



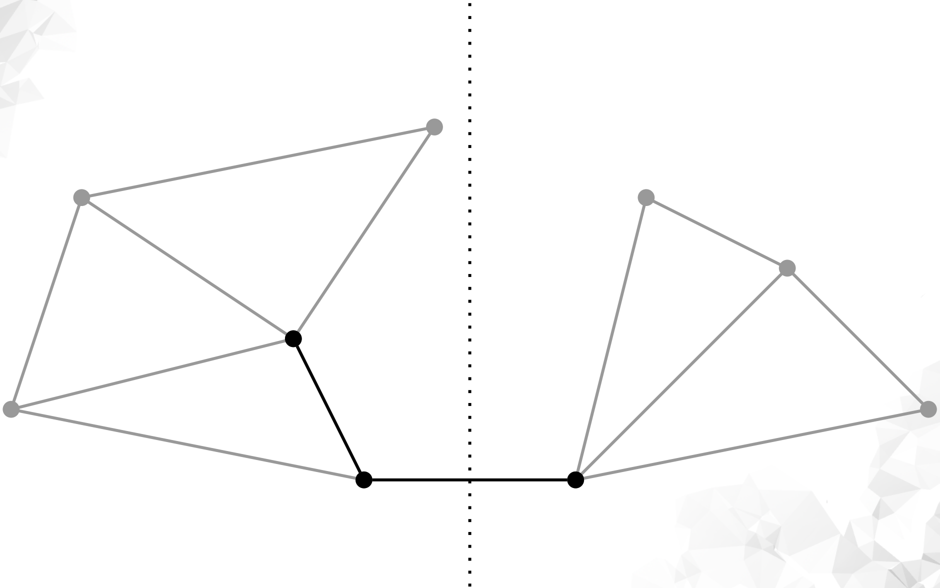
Algorithm: Merge Triangulations Example



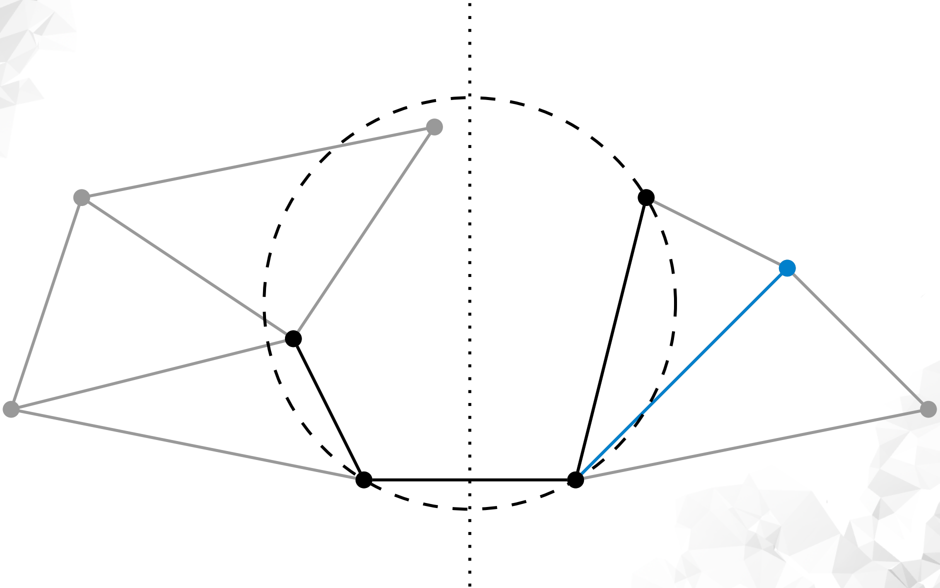
Algorithm: Merge Triangulations Example



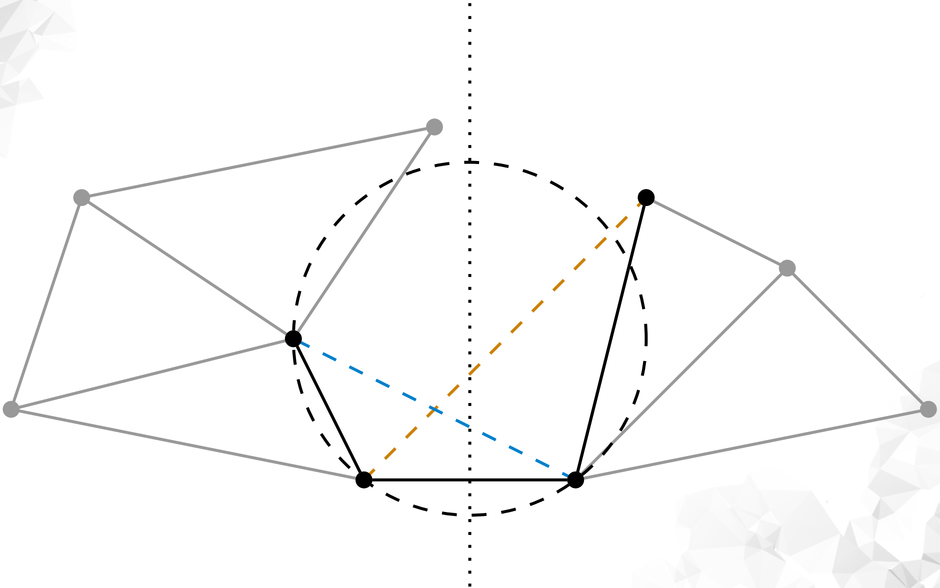
Algorithm: Merge Triangulations Example



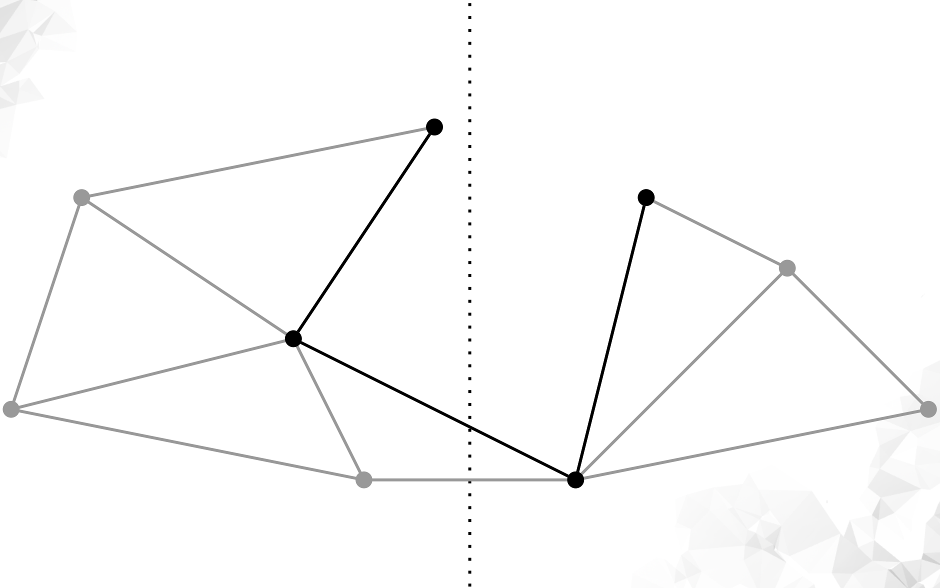
Algorithm: Merge Triangulations Example



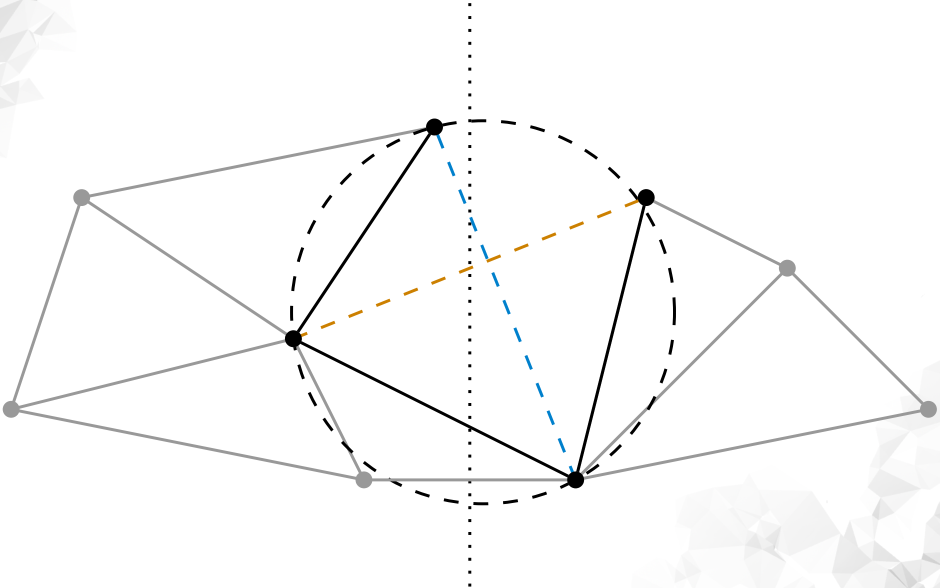
Algorithm: Merge Triangulations Example



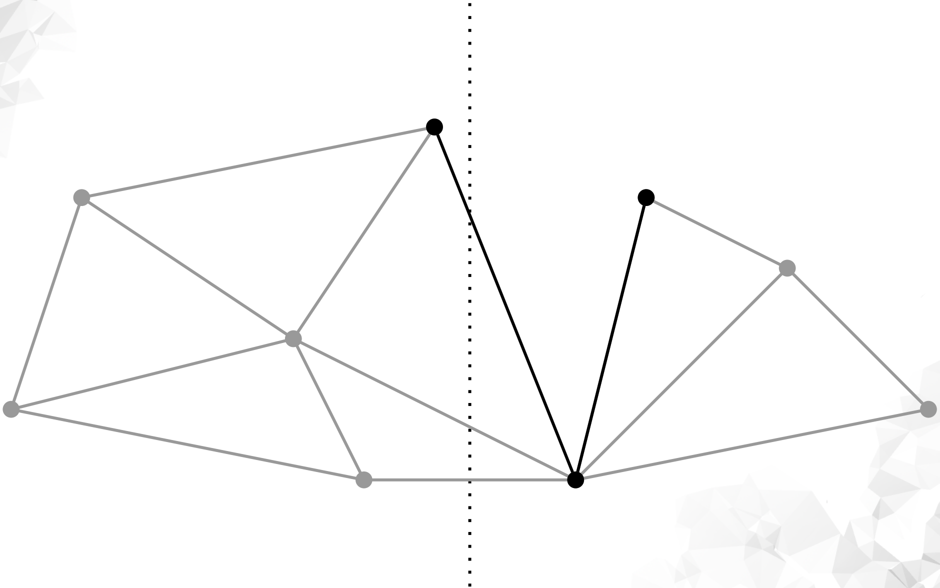
Algorithm: Merge Triangulations Example



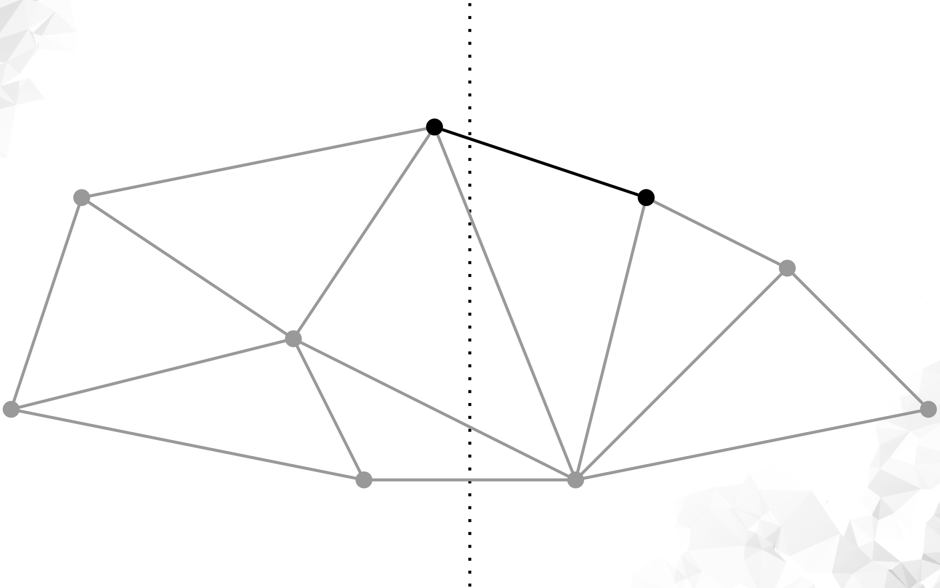
Algorithm: Merge Triangulations Example



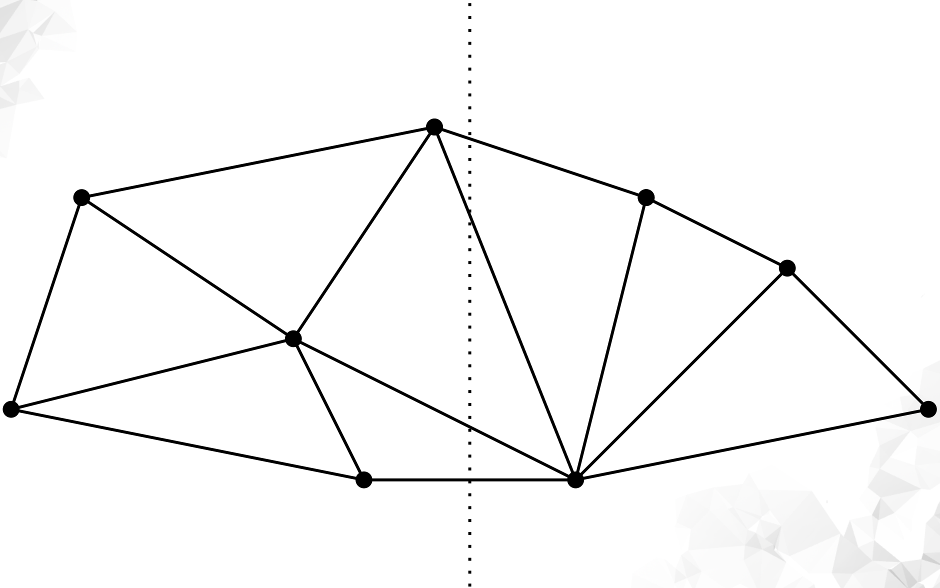
Algorithm: Merge Triangulations Example



Algorithm: Merge Triangulations Example



Algorithm: Merge Triangulations Example



Algorithm: Merge Triangulations

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.
3. Loop until baseline becomes upper common tangent:

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.
3. Loop until baseline becomes upper common tangent:
   3.1 Remove invalid edges adjacent to and above baseline.

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.
3. Loop until baseline becomes upper common tangent:
   3.1 Remove invalid edges adjacent to and above baseline.
   3.2 Insert cross edge above baseline.

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.
3. Loop until baseline becomes upper common tangent:
   3.1 Remove invalid edges adjacent to and above baseline.
   3.2 Insert cross edge above baseline.
   3.3 Make this cross edge the new baseline.

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.
3. Loop until baseline becomes upper common tangent:
   3.1 Remove invalid edges adjacent to and above baseline.
   3.2 Insert cross edge above baseline.
   3.3 Make this cross edge the new baseline.

Details:

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.
3. Loop until baseline becomes upper common tangent:
   3.1 Remove invalid edges adjacent to and above baseline.
   3.2 Insert cross edge above baseline.
   3.3 Make this cross edge the new baseline.

Details:

- Computation of lower common tangent

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.
3. Loop until baseline becomes upper common tangent:
   3.1 Remove invalid edges adjacent to and above baseline.
   3.2 Insert cross edge above baseline.
   3.3 Make this cross edge the new baseline.

Details:

- Computation of lower common tangent
- Circle test for adjacent edges

Algorithm: Merge Triangulations

Subroutine: Merge Triangulations

1. Compute and add lower common tangent.
2. Use lower common tangent as baseline.
3. Loop until baseline becomes upper common tangent:
   3.1 Remove invalid edges adjacent to and above baseline.
   3.2 Insert cross edge above baseline.
   3.3 Make this cross edge the new baseline.

Details:

- Computation of lower common tangent
- Circle test for adjacent edges
- Circle test for cross edge

Algorithm: Complexity

Subroutine: Triangulate

1. If $n \leq 3$, make edge or triangle and return.
2. Split point set into left and right.
3. Triangulate left and right half.
4. Merge left and right triangulations.

Algorithm: Complexity

- $n \in \mathbb{N}$ points

Subroutine: Triangulate

1. If $n \leq 3$, make edge or triangle and return.
2. Split point set into left and right.
3. Triangulate left and right half.
4. Merge left and right triangulations.

Algorithm: Complexity

- $n \in \mathbb{N}$ points
- $T(n)$ runtime

Subroutine: Triangulate

1. If $n \leq 3$, make edge or triangle and return.
2. Split point set into left and right.
3. Triangulate left and right half.
4. Merge left and right triangulations.

Algorithm: Complexity

- $n \in \mathbb{N}$ points
- $T(n)$ runtime
- Use master theorem

Subroutine: Triangulate

1. If $n \leq 3$, make edge or triangle and return.
2. Split point set into left and right.
3. Triangulate left and right half.
4. Merge left and right triangulations.

Algorithm: Complexity

- $n \in \mathbb{N}$ points
- $T(n)$ runtime
- Use master theorem
- Complexity of merge is linear in worst case

Subroutine: Triangulate

1. If $n \leq 3$, make edge or triangle and return.
2. Split point set into left and right.
3. Triangulate left and right half.
4. Merge left and right triangulations.

Algorithm: Complexity

- $n \in \mathbb{N}$ points
- $T(n)$ runtime
- Use master theorem
- Complexity of merge is linear in worst case

$$T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n)$$

Subroutine: Triangulate

1. If $n \leq 3$, make edge or triangle and return.
2. Split point set into left and right.
3. Triangulate left and right half.
4. Merge left and right triangulations.

Algorithm: Complexity

- $n \in \mathbb{N}$ points
- $T(n)$ runtime
- Use master theorem
- Complexity of merge is linear in worst case

$$T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n)$$

$$T(n) = \mathcal{O}(n \log n)$$

Subroutine: Triangulate

1. If $n \leq 3$, make edge or triangle and return.
2. Split point set into left and right.
3. Triangulate left and right half.
4. Merge left and right triangulations.

Algorithm: Correctness

Algorithm: Correctness

- Proof by induction for merging left and right half

Algorithm: Correctness

- Proof by induction for merging left and right half
- For two Delaunay triangulations separated by a vertical line, it is enough to remove inner edges and insert cross edges.

Algorithm: Correctness

- Proof by induction for merging left and right half
- For two Delaunay triangulations separated by a vertical line, it is enough to remove inner edges and insert cross edges.
- Common tangents are elements of the Delaunay triangulation.

Algorithm: Correctness

- Proof by induction for merging left and right half
- For two Delaunay triangulations separated by a vertical line, it is enough to remove inner edges and insert cross edges.
- Common tangents are elements of the Delaunay triangulation.
- Removed edges are indeed not Delaunay.

Algorithm: Correctness

- Proof by induction for merging left and right half
- For two Delaunay triangulations separated by a vertical line, it is enough to remove inner edges and insert cross edges.
- Common tangents are elements of the Delaunay triangulation.
- Removed edges are indeed not Delaunay.
- Insertion of cross edges generates new Delaunay triangle.

Algorithm: Correctness

- Proof by induction for merging left and right half
- For two Delaunay triangulations separated by a vertical line, it is enough to remove inner edges and insert cross edges.
- Common tangents are elements of the Delaunay triangulation.
- Removed edges are indeed not Delaunay.
- Insertion of cross edges generates new Delaunay triangle.
- There are no other edges that have to be removed.

Algorithm: Correctness

- Proof by induction for merging left and right half
- For two Delaunay triangulations separated by a vertical line, it is enough to remove inner edges and insert cross edges.
- Common tangents are elements of the Delaunay triangulation.
- Removed edges are indeed not Delaunay.
- Insertion of cross edges generates new Delaunay triangle.
- There are no other edges that have to be removed.
- Algorithm is correct

Implementation Notes

Implementation Notes

- Geometric primitives need to be robustly computed

Implementation Notes

- Geometric primitives need to be robustly computed
- Still no robust split, use Dwyer algorithm instead

Implementation Notes

- Geometric primitives need to be robustly computed
- Still no robust split, use Dwyer algorithm instead
- Triangular data structure increases speed but algorithm is more complicated

Implementation Notes

- Geometric primitives need to be robustly computed
- Still no robust split, use Dwyer algorithm instead
- Triangular data structure increases speed but algorithm is more complicated
- Divide-and-conquer variant seems to be most powerful and robust

Conclusions

Conclusions

Summary:

Conclusions

Summary:

- Delaunay triangulation can be generated by given divide-and-conquer algorithm in $\mathcal{O}(n \log n)$

Conclusions

Summary:

- Delaunay triangulation can be generated by given divide-and-conquer algorithm in $\mathcal{O}(n \log n)$
- Data structure needs to store neighbor information

Conclusions

Summary:

- Delaunay triangulation can be generated by given divide-and-conquer algorithm in $\mathcal{O}(n \log n)$
- Data structure needs to store neighbor information

Future Work:

Conclusions

Summary:

- Delaunay triangulation can be generated by given divide-and-conquer algorithm in $\mathcal{O}(n \log n)$
- Data structure needs to store neighbor information

Future Work:

- Use triangular data structure instead of quad-edge data structure

Conclusions

Summary:

- Delaunay triangulation can be generated by given divide-and-conquer algorithm in $\mathcal{O}(n \log n)$
- Data structure needs to store neighbor information

Future Work:

- Use triangular data structure instead of quad-edge data structure
- Use Dwyer's approach to make algorithm more robust

Conclusions

Summary:

- Delaunay triangulation can be generated by given divide-and-conquer algorithm in $\mathcal{O}(n \log n)$
- Data structure needs to store neighbor information

Future Work:

- Use triangular data structure instead of quad-edge data structure
- Use Dwyer's approach to make algorithm more robust
- Parallelization

Conclusions

Summary:

- Delaunay triangulation can be generated by given divide-and-conquer algorithm in $\mathcal{O}(n \log n)$
- Data structure needs to store neighbor information

Future Work:

- Use triangular data structure instead of quad-edge data structure
- Use Dwyer's approach to make algorithm more robust
- Parallelization
- Multi-dimensional Implementation

Thank you for Your Attention!

References

(1) D. T. Lee and B. J. Schachter. "Two Algorithms for Constructing a Delaunay Triangulation". In: *International Journal of Computer and Information Sciences* 9 (1980), pp. 219–242. DOI: 10.1007/BF00977785.

(2) Leonidas Guibas and Jorge Stolfi. "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams". In: *ACM Transactions on Graphics* 4 (April 1985), pp. 74–123. DOI: 10.1145/282918.282923. URL: http://sccg.sk/~samuelcik/dgs/quad_edge.pdf (visited on 11/07/2020).

(3) Rex A. Dwyer. "A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations". In: *Algorithmica* 2 (November 1987), pp. 137–151. DOI: 10.1007/BF01840356.

(4) Jonathan Richard Shewchuk. "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator". In: *Applied Computational Geometry: Towards Geometric Engineering*. Ed. by Ming C. Lin and Dinesh Manocha. Vol. 1148. Lecture Notes in Computer Science. From the First ACM Workshop on Applied Computational Geometry. Springer-Verlag, May 1996, pp. 203–222. URL: https:

(7) D. F. Watson. "Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes". In: *The Computer Journal* 24 (1981), pp. 167–172. DOI: 10.1093/comjnl/24.2.167.

(8) A. Bowyer. "Computing Dirichlet Tessellations". In: *The Computer Journal* 24 (1981), pp. 162–166. DOI: 10.1093/comjnl/24.2.162.

(9) Christoph Burnikel. *Delaunay Graphs by Divide and Conquer*. 1998. URL: https://pure.mpg.de/rest/items/item_1819432_4/component/file_2599484/content (visited on 11/07/2020).

(10) P. Cignoni, C. Montani, and R. Scopigno. "DeWall: A Fast Divide-and-Conquer Delaunay Triangulation Algorithm in E^d". In: *Computer-Aided Design* 30 (1998), pp. 333–341. DOI: 10.1016/S0010-4485(97)00082-1.

(11) Jyrki Katajainen and Markku Koppinen. "Constructing Delaunay Triangulations by Merging Buckets in Quad-Tree Order". In: *Fundamenta Informaticae* 11 (April 1988), pp. 275–288.