An introduction to computational geometry

Cours électif - École des mines - Nancy - 2011

Xavier Goaoc

Question 1: What is it all about?

It's about designing algorithmic solutions to geometric problems.

Ex: *path planning, geometric model manipulation, visibility computation...*

It's about designing algorithmic solutions to geometric problems.

Ex: *path planning, geometric model manipulation, visibility computation...*

Ideally we want solutions that are:

- ⋆ Proven correct.

- ⋆ Proven efficient.

- ⋆ Work in practice.

It's about designing algorithmic solutions to geometric problems.

Ex: *path planning, geometric model manipulation, visibility computation...*

Ideally we want solutions that are:

- ⋆ Proven correct.

  In a mathematical sense.

  Often based on geometric arguments.

  Often requires new geometric insight.

- ⋆ Proven efficient.

  In the sense of complexity theory.

  Complexity of algorithms are analyzed in an adequate model of computation.

  Understanding the complexity of the problems themselves is important.

- ⋆ Work in practice.

  Algorithms that are simple enough to be implemented.

  Implementations that handle degeneracies and finite precision arithmetic.

It's about designing algorithmic solutions to geometric problems.

Ex: *path planning, geometric model manipulation, visibility computation...*

Ideally we want solutions that are:

- ⋆ Proven correct.

  In a mathematical sense.

  Often based on geometric arguments.

  Often requires new geometric insight.

- ⋆ Proven efficient.

  In the sense of complexity theory.

  Complexity of algorithms are analyzed in an adequate model of computation.

  Understanding the complexity of the problems themselves is important.

- ⋆ Work in practice.

  Algorithms that are simple enough to be implemented.

  Implementations that handle degeneracies and finite precision arithmetic.

Model of computation

Definition of the operations allowed in an algorithm and their cost.

Goal: estimate the ressources required by an algorithm

as a function of the input size.

Ex: execution time, memory space, number of I/O transfers, number of processors...

Principle: we do not want a precise cost estimation

We want to speak of algorithms independently of the technology.

We want to compare algorithms, not implementations.

Model of computation

Definition of the operations allowed in an algorithm and their cost.

Goal: estimate the ressources required by an algorithm

as a function of the input size.

Ex: execution time, memory space, number of I/O transfers, number of processors...

Principle: we do not want a precise cost estimation

We want to speak of algorithms independently of the technology.

We want to compare algorithms, not implementations.

Classical model in CG: Real RAM model

Allows manipulation of real (as in $\mathbb{R}$) numbers.

Input size $n \rightarrow$ complexity $f(n) = \max_{\text{input } |X|=n} f(X)$

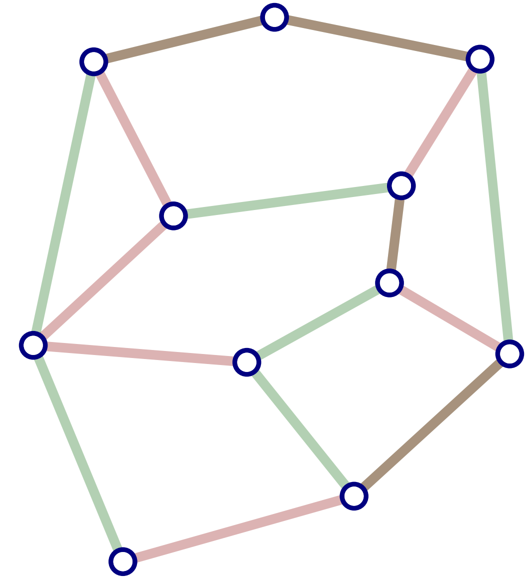
Care about asymptotic order of magnitude of f $(O(), \Omega(), \Theta())$.

Brute force does not scale well (or: why should we think?)

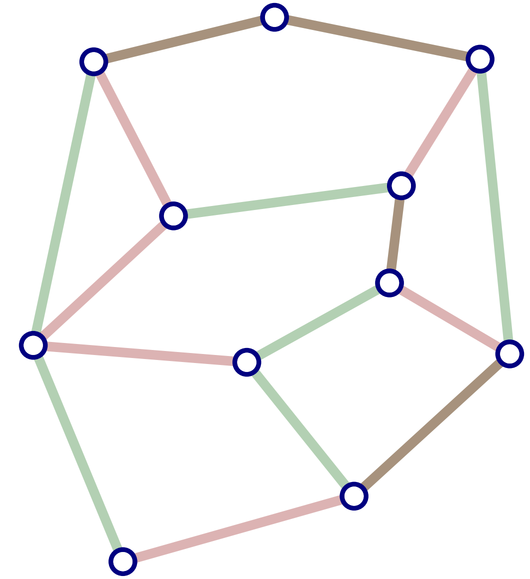
The "Travelling salesman problem".

Input: n cities and all inter-city distances.

Output: order on the cities that minimizes the distance travelled.



Brute force does not scale well (or: why should we think?)



The "Travelling salesman problem".

Input: n cities and all inter-city distances.

Output: order on the cities that minimizes the distance travelled.

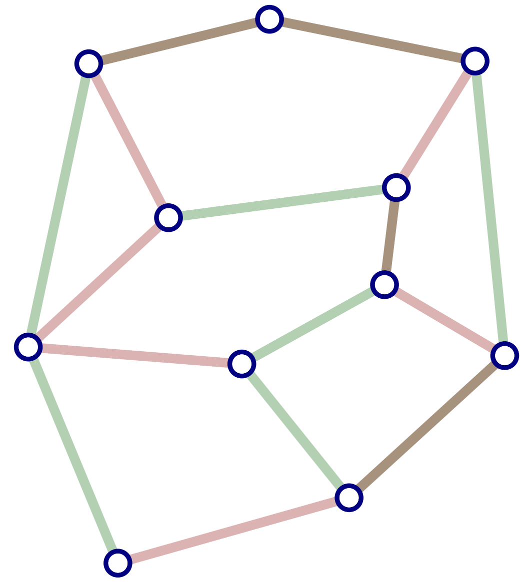
Brute-force approach: test all $n!$ orders and pick the best.

Brute force does not scale well (or: why should we think?)

The "Travelling salesman problem".

Input: n cities and all inter-city distances.

Output: order on the cities that minimizes the distance travelled.



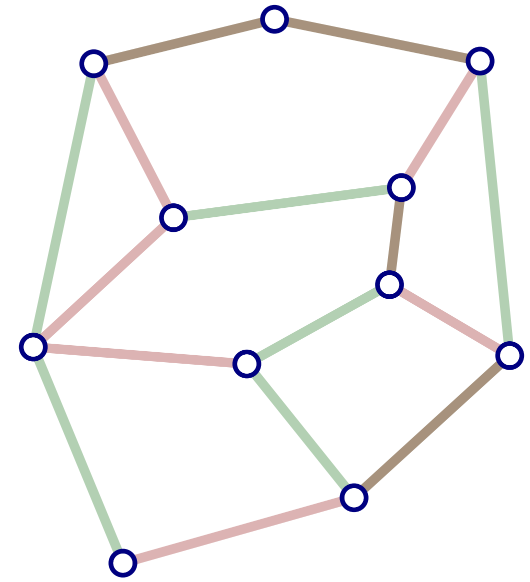
Brute-force approach: test all $n!$ orders and pick the best.

Assume your computer can process 10^{15} orders per second.

Generate the order, add-up the distances, compare to the current best...

A very generous over-estimation.

Brute force does not scale well (or: why should we think?)



The "Travelling salesman problem".

Input: n cities and all inter-city distances.

Output: order on the cities that minimizes the distance travelled.

Brute-force approach: test all $n!$ orders and pick the best.

Assume your computer can process 10^{15} orders per second.

Generate the order, add-up the distances, compare to the current best...

A very generous over-estimation.

Start the computation now. It will end...

in 30-60 min. for $n = 20$.

in two weeks for $n = 22$.

in twenty years for $n = 24$.

in four centuries for $n = 25$.

in the dark for $n = 30$.

Orders of magnitude

Sort by increasing asymptotic orders of magnitude:

$$n,\ 2^n,\ n^2,\ n!,\ \sqrt{n},\ \log n,\ \log^* n,\ 2^{n^2}$$

Orders of magnitude

Sort by increasing asymptotic orders of magnitude:

$$n,\ 2^n,\ n^2,\ n!,\ \sqrt{n},\ \log n,\ \log^* n,\ 2^{n^2}$$

$$\log^* n \ll \log n \ll \sqrt{n} \ll n \ll n^2 \ll 2^n \ll n! \ll 2^{n^2}$$

Orders of magnitude

Sort by increasing asymptotic orders of magnitude:

$$n,\ 2^n,\ n^2,\ n!,\ \sqrt{n},\ \log n,\ \log^* n,\ 2^{n^2}$$

$$\log^* n \ll \log n \ll \sqrt{n} \ll n \ll n^2 \ll 2^n \ll n! \ll 2^{n^2}$$

$$\log^* n \ll \log^a n \ll n^b \ll 2^{cn} \ll (n!)^d \ll 2^{en^2}$$

$$\forall a, b, c, d, e \in \mathbb{R}_+$$

Orders of magnitude

Sort by increasing asymptotic orders of magnitude:

$$n,\ 2^n,\ n^2,\ n!,\ \sqrt{n},\ \log n,\ \log^* n,\ 2^{n^2}$$

$$\log^* n \ll \log n \ll \sqrt{n} \ll n \ll n^2 \ll 2^n \ll n! \ll 2^{n^2}$$

$$\log^* n \ll \log^a n \ll n^b \ll 2^{cn} \ll (n!)^d \ll 2^{en^2} \qquad \forall a,b,c,d,e \in \mathbb{R}_+$$

Three classes of problems

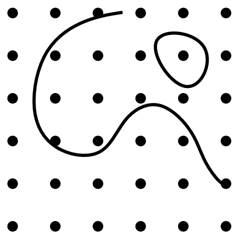
Undecidable: no algorithm will solve the problem. Ever.

NP-hard: conjectured unlikely that a polynomial-time algorithm exists.

Polynomial-time: solvable by an algorithm with complexity $O(n^c)$

for some constant c.

Hilbert's tenth problem



Input: a polynomial P in n variables with integer coefficients.

Output: yes if P has a integer solution, no otherwise.

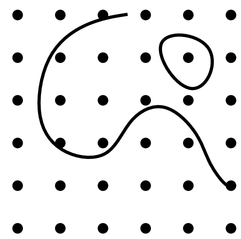
Ex: $P(x_1, x_2, x_3) = x_1^2 + 3x_1x_2 - 2x_2^2 + 4x_3 + 3$

Tenth question in Hilbert's list of *Problèmes futurs des mathématiques*.

Raised in 1900. Algorithmic question before the age of computers.

"Solved" in 1970 by Y. Matiyasevitch.

Hilbert's tenth problem



Input: a polynomial P in n variables with integer coefficients.

Output: yes if P has a integer solution, no otherwise.

Ex: $P(x_1, x_2, x_3) = x_1^2 + 3x_1x_2 - 2x_2^2 + 4x_3 + 3$

Tenth question in Hilbert's list of *Problèmes futurs des mathématiques*.

Raised in 1900. Algorithmic question before the age of computers.

"Solved" in 1970 by Y. Matiyasevitch.

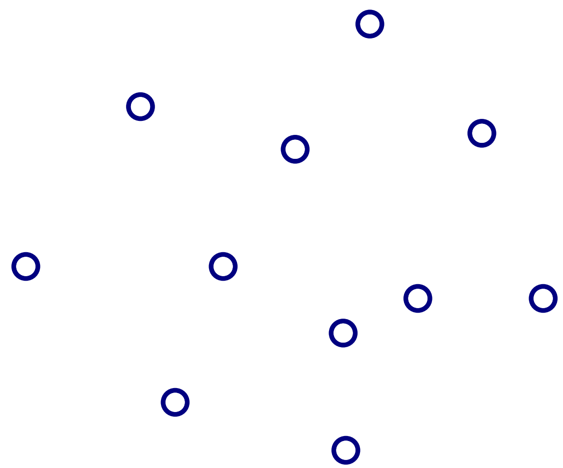
UNDECIDABLE

Minimum-weigth triangulation

Given a set of points in the plane

find a triangulation of the convex hull

that minimizes the sum of edge lengths.

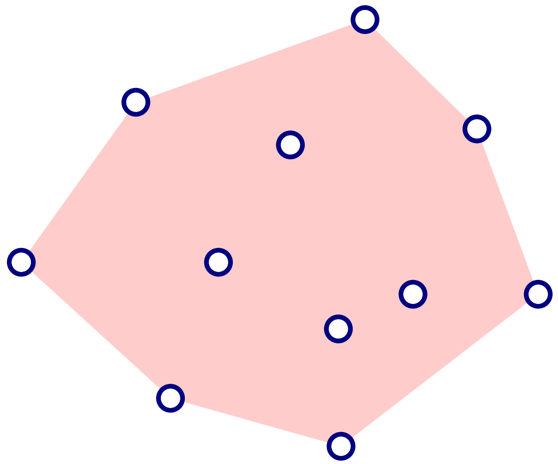


Minimum-weigth triangulation

Given a set of points in the plane

find a triangulation of the convex hull

that minimizes the sum of edge lengths.

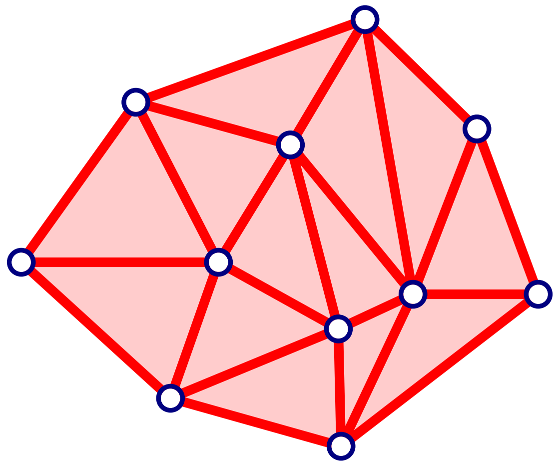


Minimum-weigth triangulation

Given a set of points in the plane

find a triangulation of the convex hull

that minimizes the sum of edge lengths.

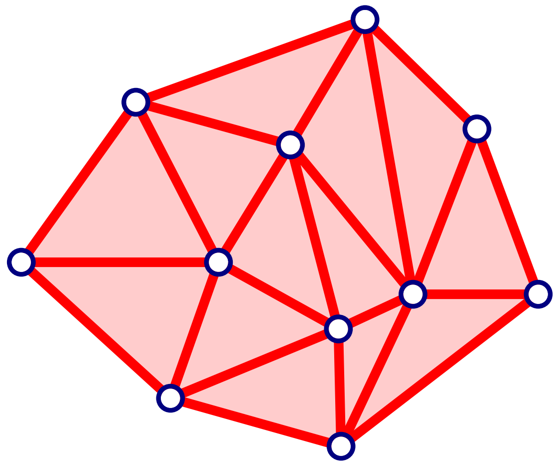


Minimum-weigth triangulation

Given a set of points in the plane

find a triangulation of the convex hull

that minimizes the sum of edge lengths.



Raised in the early 1970's.

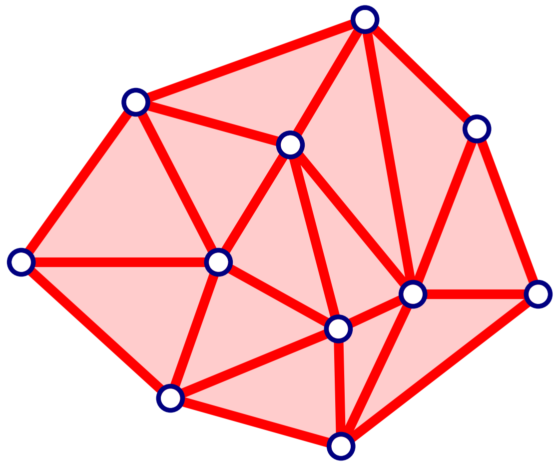
"Solved" in 2008 by Mulzer and Rote.

Minimum-weigth triangulation

Given a set of points in the plane

find a triangulation of the convex hull

that minimizes the sum of edge lengths.



Raised in the early 1970's.

"Solved" in 2008 by Mulzer and Rote.

NP-hard

Problems solvable in polynomial time

Algorithms for the same problem may have different complexities.

Ex: Merge sort has $\Theta(n \log n)$ complexity.

Bubble sort has $\Theta(n^2)$ complexity.

Quick sort has $\Theta(n^2)$ complexity but $O(n \log n)$ average-case complexity.

Problems solvable in polynomial time

Algorithms for the same problem may have different complexities.

Ex: Merge sort has $\Theta(n \log n)$ complexity.

Bubble sort has $\Theta(n^2)$ complexity.

Quick sort has $\Theta(n^2)$ complexity but $O(n \log n)$ average-case complexity.

This can have a drastic impact.

http://cg.scs.carleton.ca/∼morin/misc/sortalg/

Wrap-up: what is it about?

Algorithmic solutions to geometric problems.

Proofs of correctness and complexity bounds.

Beware of undecidable or NP-hard problems.

Asymptotic complexity matters in practice.

(Attention to degeneracy and numerical issues.)

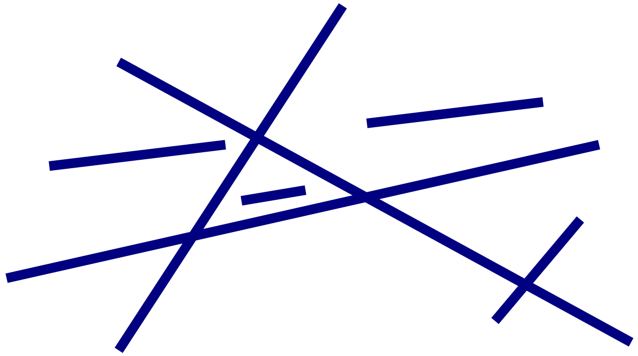
Question 2 (getting started)

How to compute the intersections among n segments in 2D?

Question 2 (getting started)

How to compute the intersections among n segments in 2D?

Input:

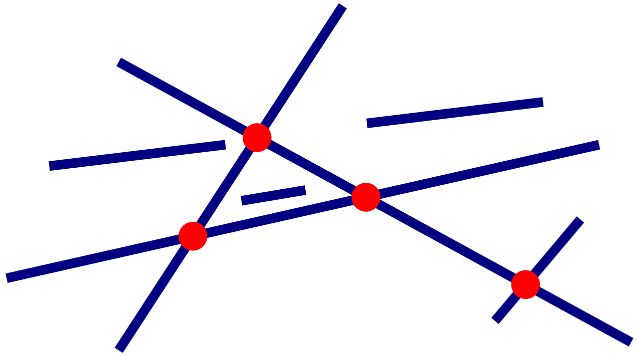


Question 2 (getting started)

How to compute the intersections among n segments in 2D?

Input:

Output:

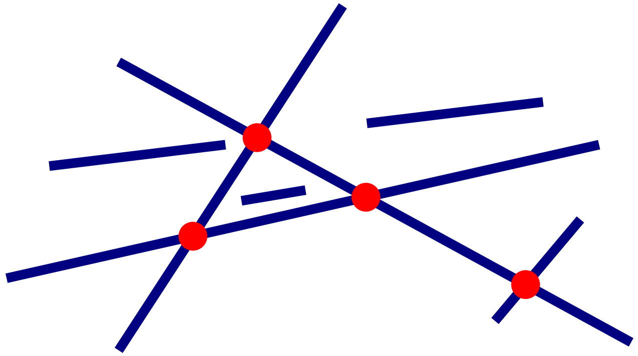


Question 2 (getting started)

How to compute the intersections among n segments in 2D?

Input:

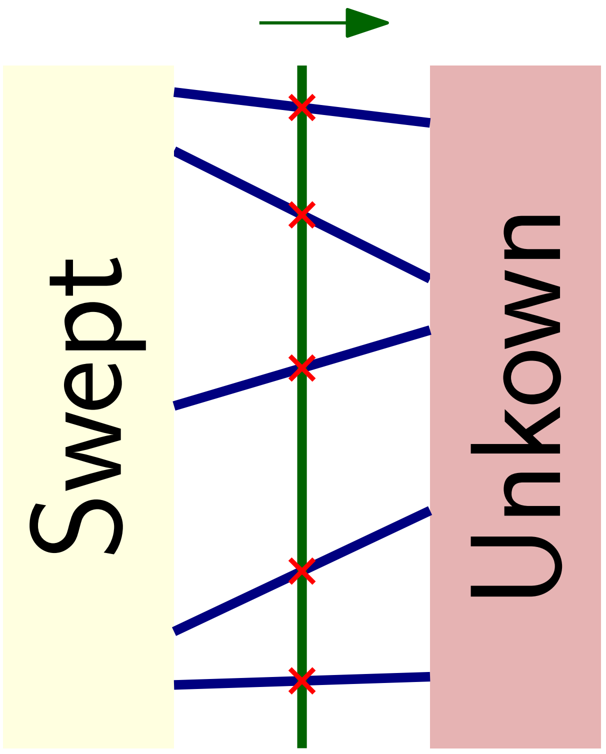
Output:



Any idea?

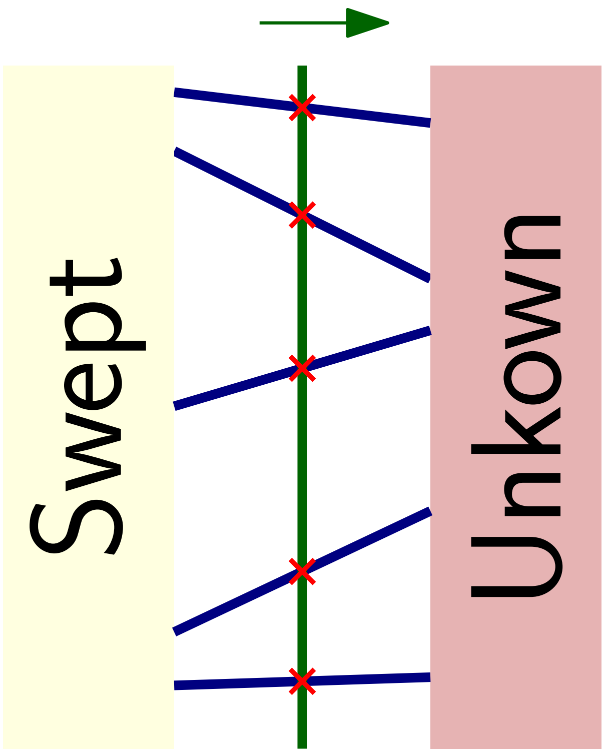
Use geometry to avoid testing unnecessary pairs.

Idea: sweep the plane using a line.



Use geometry to avoid testing unnecessary pairs.

Idea: sweep the plane using a line.

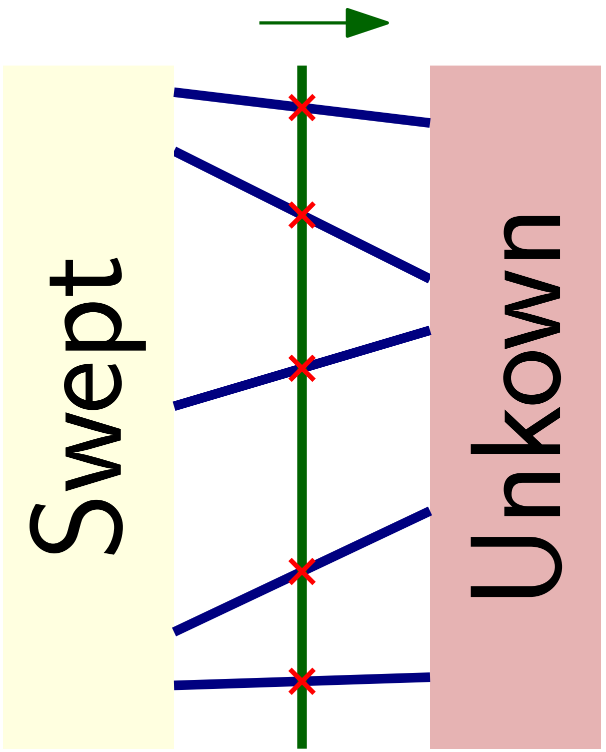


Principle:

Two segments that intersect must meet the sweep line consecutively before it reaches the intersection point.

Use geometry to avoid testing unnecessary pairs.

Idea: sweep the plane using a line.



Principle:

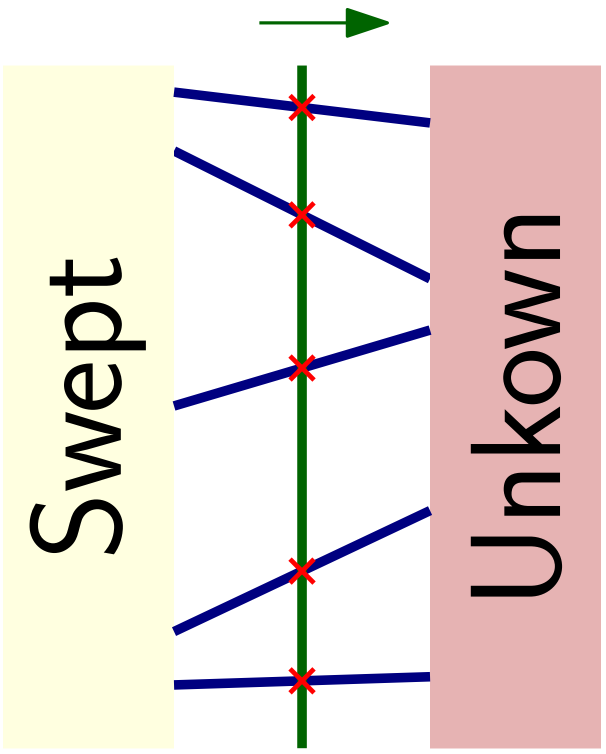
Two segments that intersect
must meet the sweep line consecutively
before it reaches the intersection point.

Idea of algorithm:

maintain the ordered list of
segments intersecting the sweep line.

Use geometry to avoid testing unnecessary pairs.

Idea: sweep the plane using a line.



Principle:

Two segments that intersect
must meet the sweep line consecutively
before it reaches the intersection point.

Idea of algorithm:

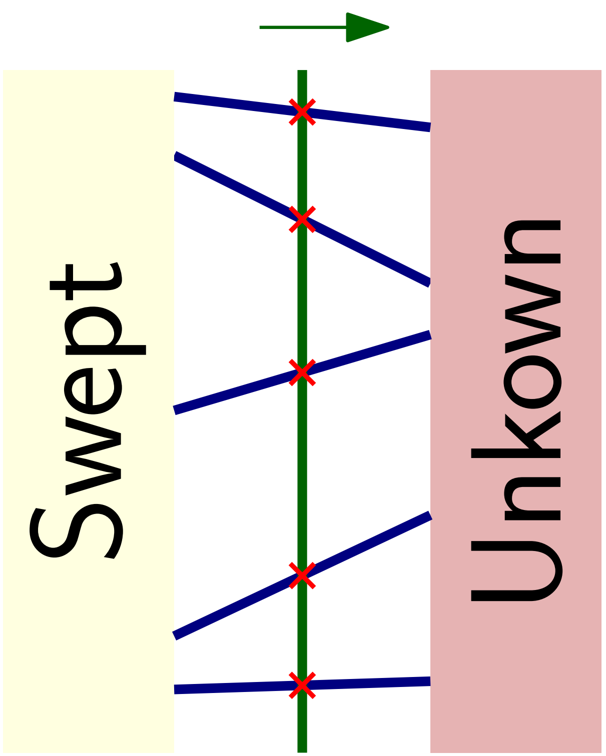
maintain the ordered list of
segments intersecting the sweep line.

Details:

How to detect the changes in the ordered list?
Data structure? Predicates?

Use geometry to avoid testing unnecessary pairs.

Idea: sweep the plane using a line.



Principle:

Two segments that intersect
must meet the sweep line consecutively
before it reaches the intersection point.

Idea of algorithm:

maintain the ordered list of
segments intersecting the sweep line.

Details:

How to detect the changes in the ordered list?
Data structure? Predicates?

Details:

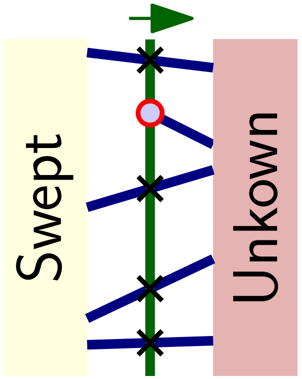
How to detect the changes in the ordered list?

Data structure? Predicates?

Three types of events

each event happens

at a particular x-coordinate



Details:

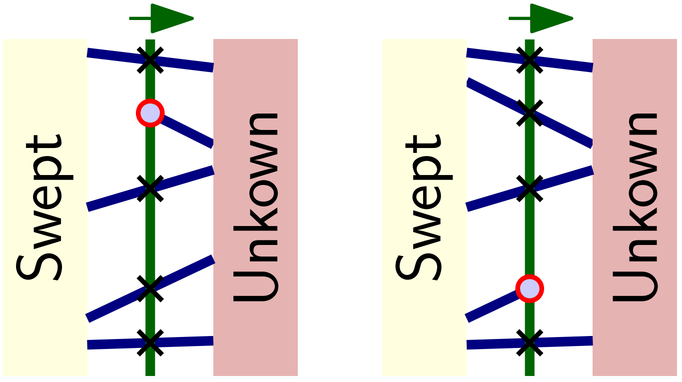
How to detect the changes in the ordered list?

Data structure? Predicates?

Three types of events

each event happens

at a particular x-coordinate



Details:

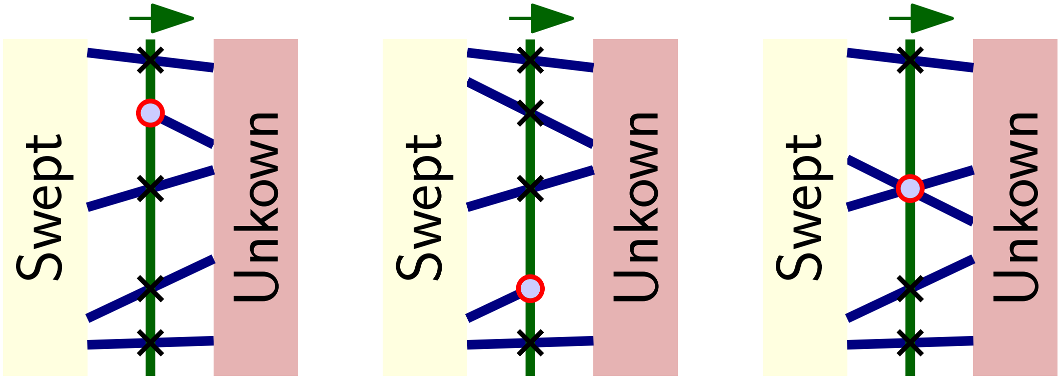
How to detect the changes in the ordered list?

Data structure? Predicates?

Three types of events

each event happens

at a particular x-coordinate



Details:

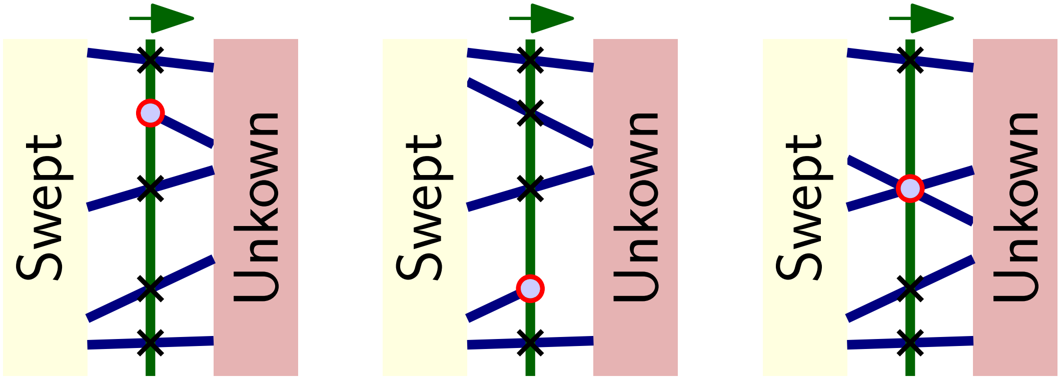
How to detect the changes in the ordered list?

Data structure? Predicates?

Three types of events

each event happens

at a particular x-coordinate

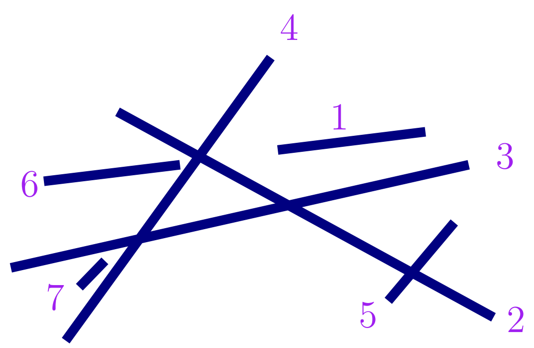


Data structures

Ordered list of segments intersected by the line.
Supports efficient insertion, deletion & exchange.

List of events sorted by x-coordinates.
Supports efficient insertion & deletion.

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

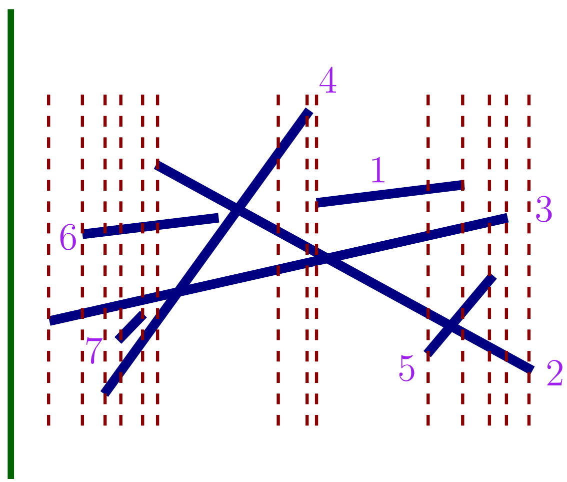
While Events $\neq \emptyset$

> Read the next event and remove it from the list.
>
> Insert, delete or swap segments in Sweep.
>
> Check intersections between new neighbors in Sweep.
>
> Add those intersections to the output and to Events.

Events: sorted list of events.

Sweep: sorted list of segments intersecting the sweep line.

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

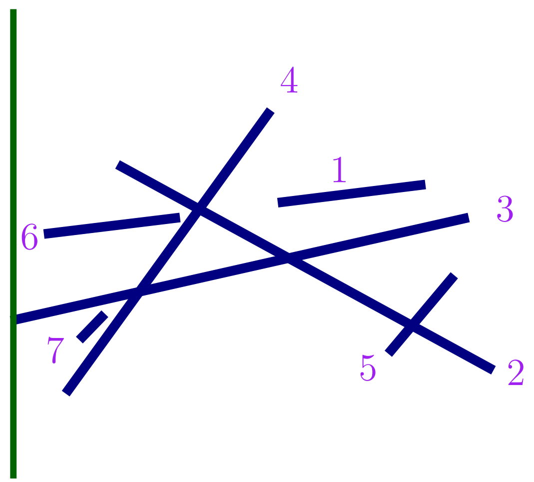
Sweep: sorted list of segments intersecting the sweep line.

Events$= \{L_3, L_6, L_4, L_7, R_7, L_2, R_6, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep$= \{\}$

Output$= \{\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

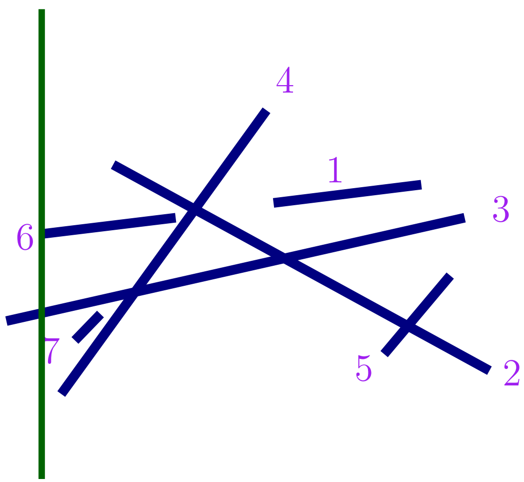
Sweep: sorted list of segments intersecting the sweep line.

Events= $\{L_3, L_6, L_4, L_7, R_7, L_2, R_6, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep= $\{3\}$

Output= $\{\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

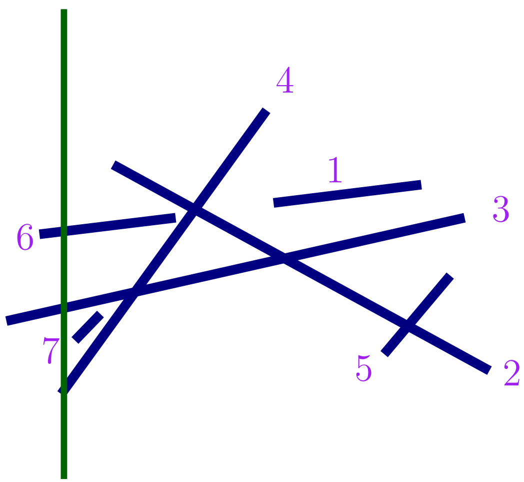
Sweep: sorted list of segments intersecting the sweep line.

Events$= \{L_6, L_4, L_7, R_7, L_2, R_6, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep$= \{6, 3\}$

Output$= \{\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

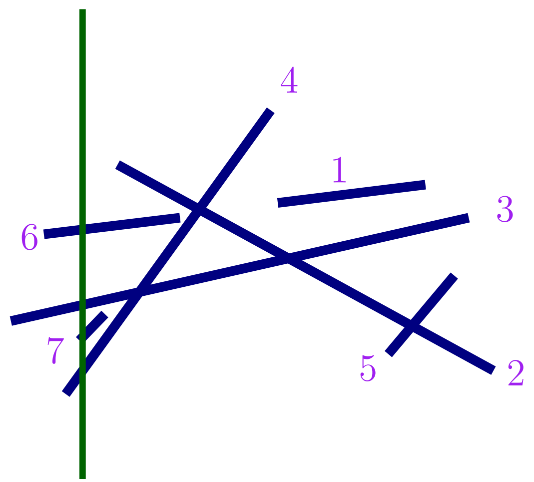
Sweep: sorted list of segments intersecting the sweep line.

Events= $\{L_4, L_7, R_7, L_2, I_{4,3}, R_6, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep= $\{6, 3, 4\}$

Output= $\{(3, 4)\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

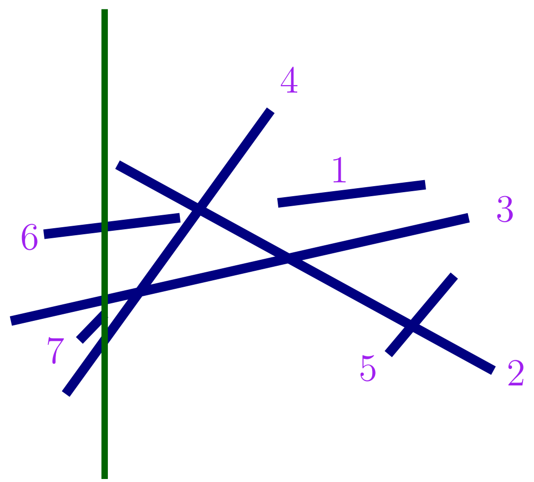
Sweep: sorted list of segments intersecting the sweep line.

Events$= \{L_7, R_7, L_2, I_{4,3}, R_6, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep$= \{6, 3, 7, 4\}$

Output$= \{(3, 4)\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

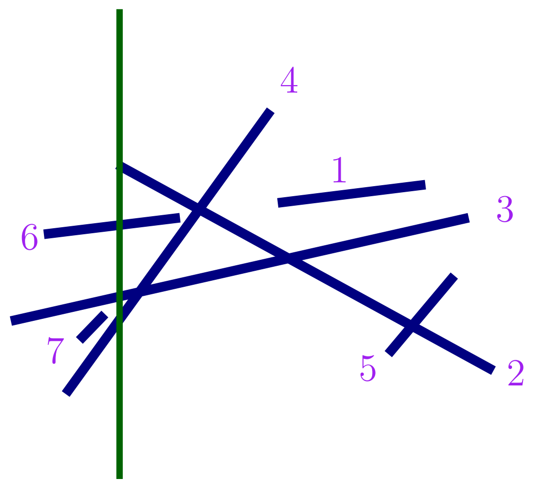
Sweep: sorted list of segments intersecting the sweep line.

Events= $\{L_7, L_2, I_{4,3}, R_6, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep= $\{6, 3, 7, 4\}$

Output= $\{(3, 4)\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

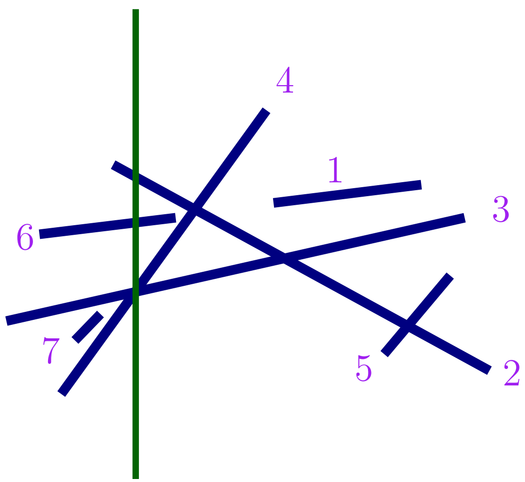
Sweep: sorted list of segments intersecting the sweep line.

Events= $\{L_2, I_{4,3}, R_6, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep= $\{2, 6, 3, 4\}$

Output= $\{(3, 4)\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

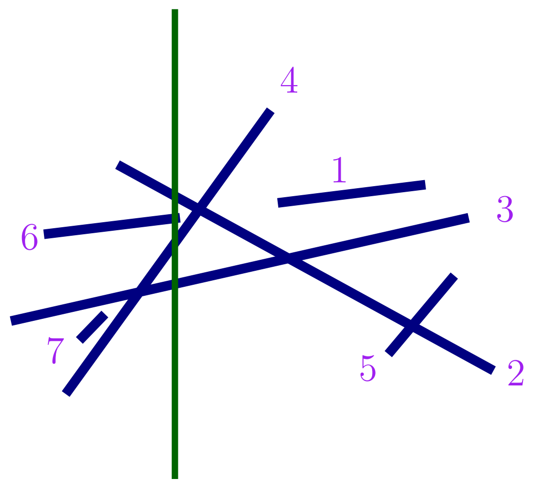
Sweep: sorted list of segments intersecting the sweep line.

Events= $\{I_{4,3}, R_6, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep= $\{2, 6, 4, 3\}$

Output= $\{(3, 4)\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

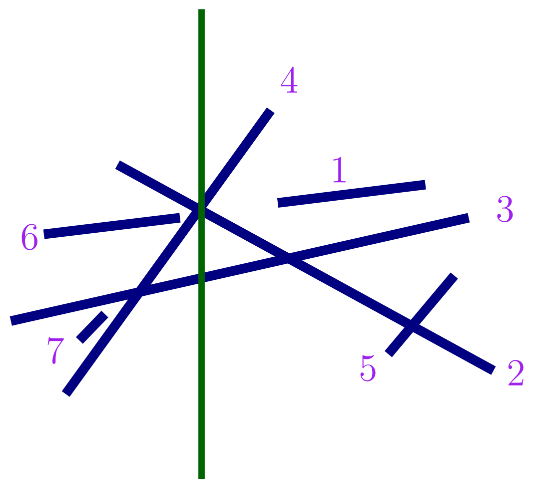
Sweep: sorted list of segments intersecting the sweep line.

Events$= \{R_6, I_{2,4}, R_4, L_1, L_5, R_1, R_5, R_3, R_2\}$

Sweep$= \{2, 6, 4, 3\}$

Output$= \{(3,4), (2,4)\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

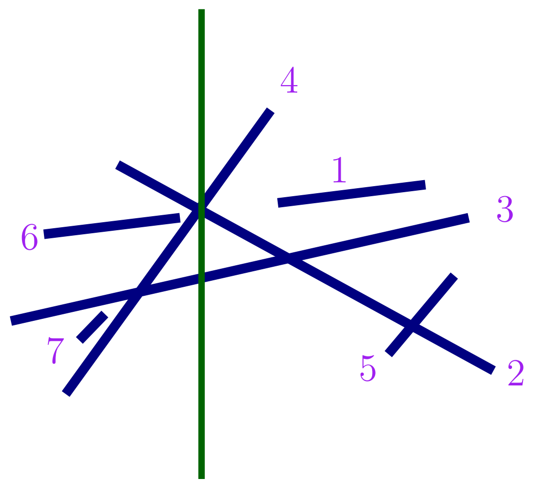
Sweep: sorted list of segments intersecting the sweep line.

Events= $\{I_{2,4}, R_4, L_1, I_{2,3}, L_5, R_1, R_5, R_3, R_2\}$

Sweep= $\{4, 2, 3\}$

Output= $\{(3,4),(2,4),(2,3)\}$

Algorithm



Insert the endpoints of all segments in Events.

Sweep $\leftarrow \emptyset$.

While Events $\neq \emptyset$

Read the next event and remove it from the list.

Insert, delete or swap segments in Sweep.

Check intersections between new neighbors in Sweep.

Add those intersections to the output and to Events.

Events: sorted list of events.

Sweep: sorted list of segments intersecting the sweep line.

Events$= \{I_{2,4}, R_4, L_1, I_{2,3}, L_5, R_1, R_5, R_3, R_2\}$

Sweep$= \{4, 2, 3\}$

Output$= \{(3,4), (2,4), (2,3)\}$

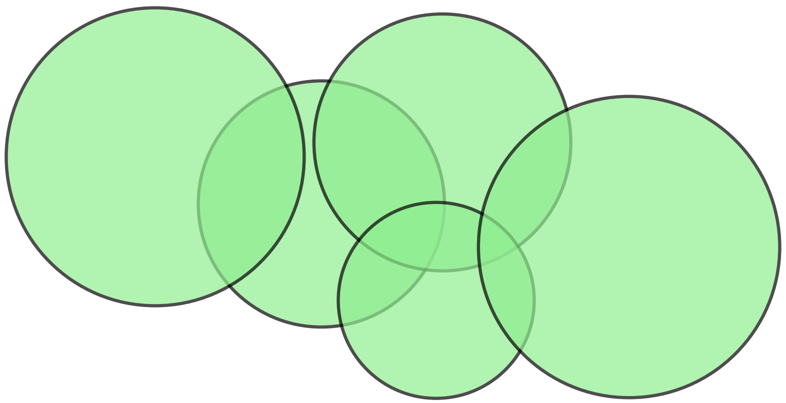
etc...

Correctness? Complexity?

Wrap-up: sweep algorithms

Generic principle, three predicates: x-extreme points, intersection, x-coordinate comparison.

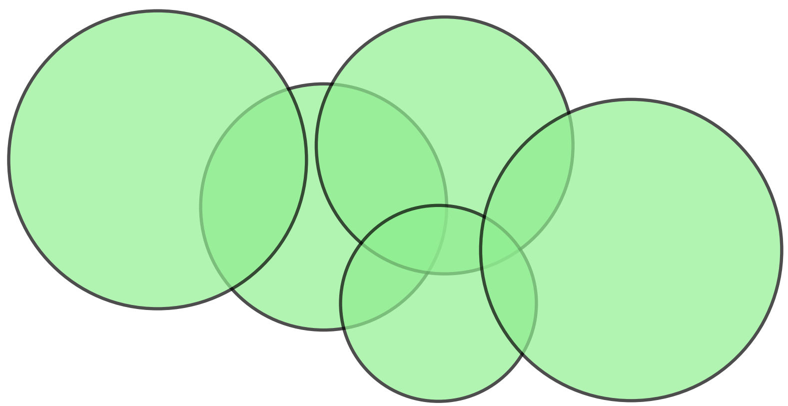
Other objects (polygons, circles, algebraic curves, etc...), other spaces ($\mathbb{S}^2$, $\mathbb{R}^3$, $\mathbb{S}^1 \times \mathbb{S}^1$...).

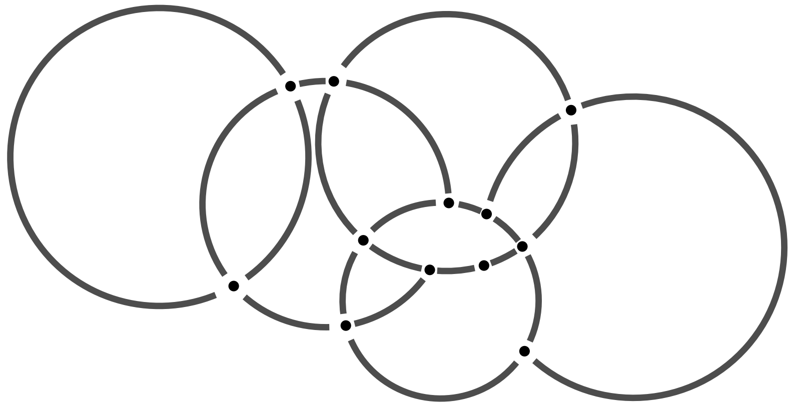


Wrap-up: sweep algorithms

Generic principle, three predicates: x-extreme points, intersection, x-coordinate comparison.

Other objects (polygons, circles, algebraic curves, etc...), other spaces ($\mathbb{S}^2$, $\mathbb{R}^3$, $\mathbb{S}^1 \times \mathbb{S}^1$...).



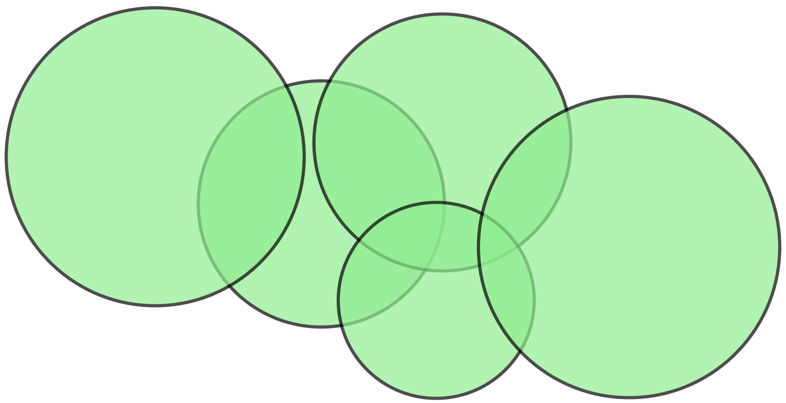


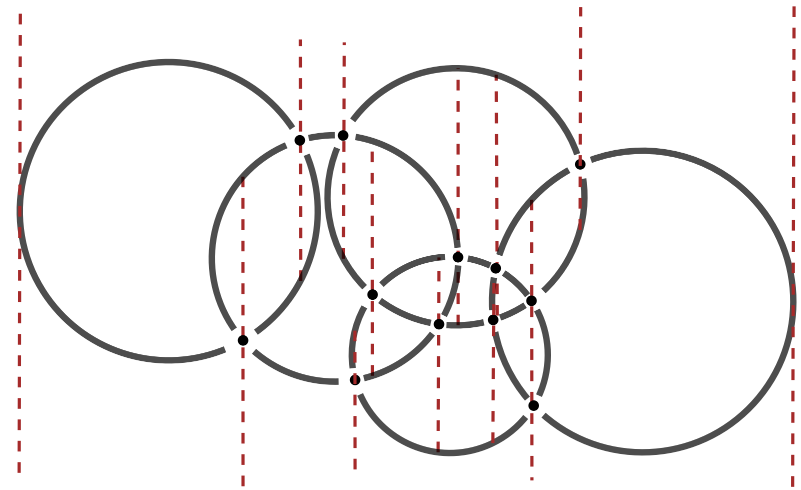
Computing arrangements of geometric objects.

Wrap-up: sweep algorithms

Generic principle, three predicates: x-extreme points, intersection, x-coordinate comparison.

Other objects (polygons, circles, algebraic curves, etc...), other spaces ($\mathbb{S}^2$, $\mathbb{R}^3$, $\mathbb{S}^1 \times \mathbb{S}^1$...).





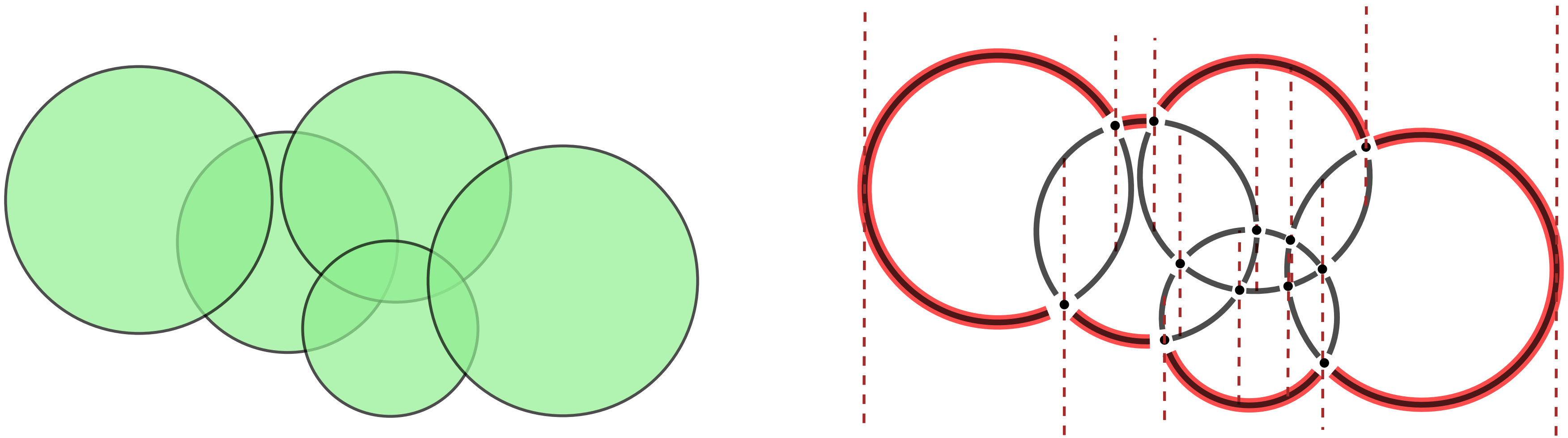
Computing arrangements of geometric objects.

Computing trapezoidal decompositions of arrangements of geometric objects.

Wrap-up: sweep algorithms

Generic principle, three predicates: x-extreme points, intersection, x-coordinate comparison.

Other objects (polygons, circles, algebraic curves, etc...), other spaces ($\mathbb{S}^2$, $\mathbb{R}^3$, $\mathbb{S}^1 \times \mathbb{S}^1$...).



Computing arrangements of geometric objects.

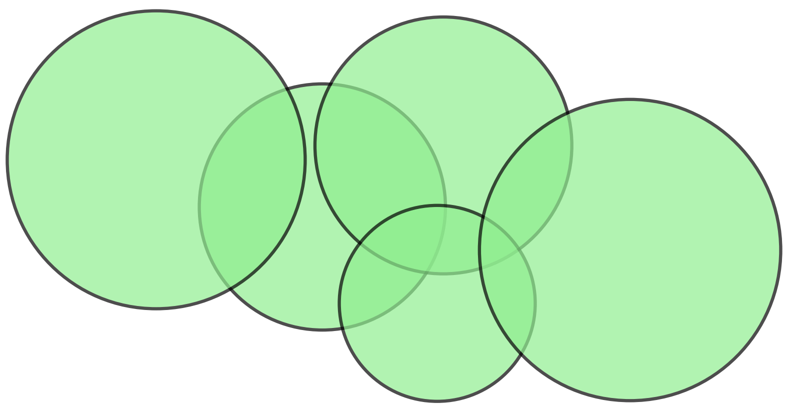
Computing trapezoidal decompositions of arrangements of geometric objects.

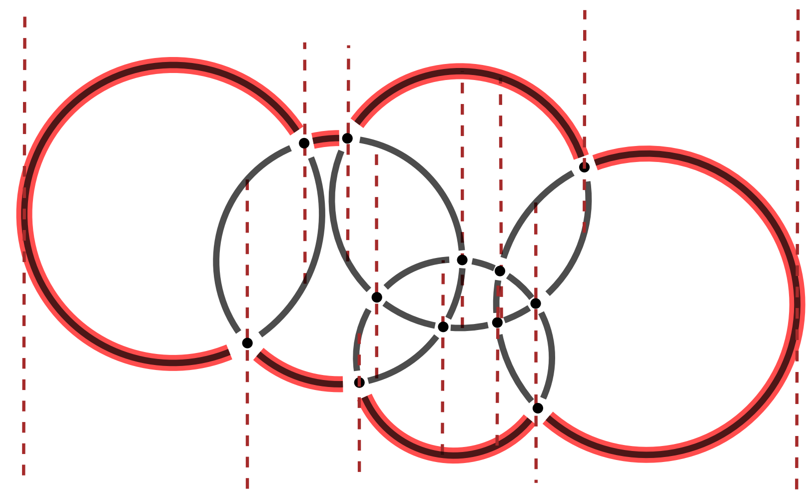
Computing substructures of arrangements of geometric objects.

Wrap-up: sweep algorithms

Generic principle, three predicates: x-extreme points, intersection, x-coordinate comparison.

Other objects (polygons, circles, algebraic curves, etc...), other spaces ($\mathbb{S}^2$, $\mathbb{R}^3$, $\mathbb{S}^1 \times \mathbb{S}^1$...).





Computing arrangements of geometric objects.

Computing trapezoidal decompositions of arrangements of geometric objects.

Computing substructures of arrangements of geometric objects.

All that in $O((n+k)\log n)$.

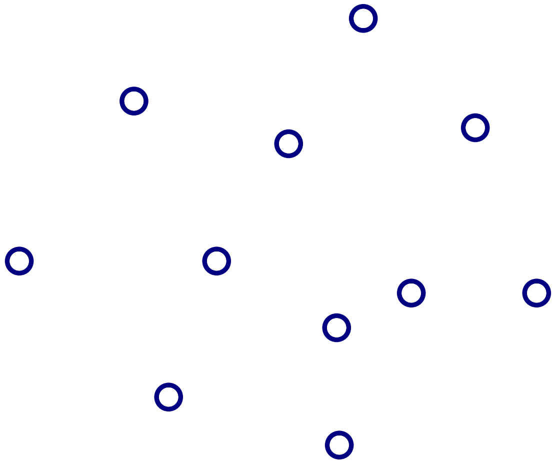
Question 3

How to triangulate a set of points?

Question 3

How to triangulate a set of points?

Input:

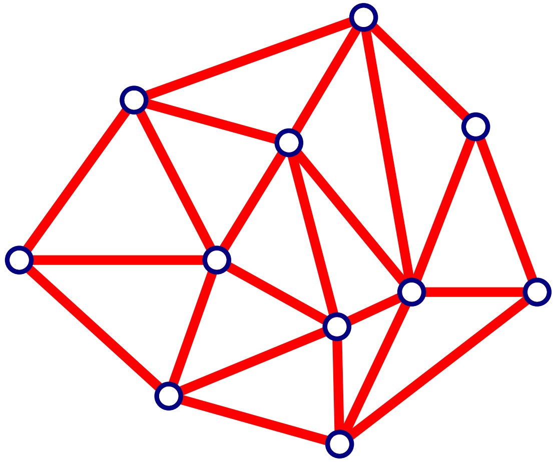


Question 3

How to triangulate a set of points?

Input:

Output:

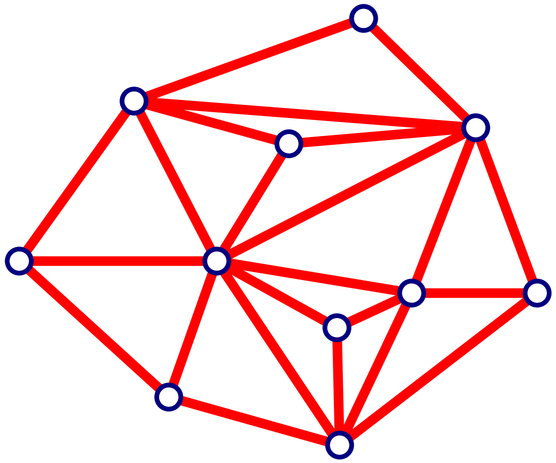


Question 3

How to triangulate a set of points?

Input:

Output:

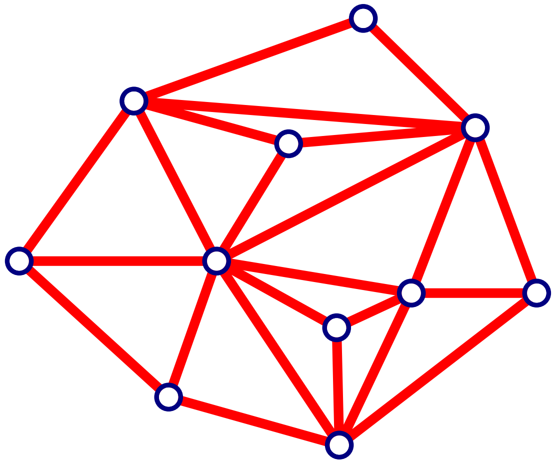


Question 3

How to triangulate a set of points?

Input:

Output:



What is a *good* triangulation?

Number of triangulations

How many triangulations on 4 points? 5 points?

Number of triangulations

How many triangulations on 4 points? 5 points?

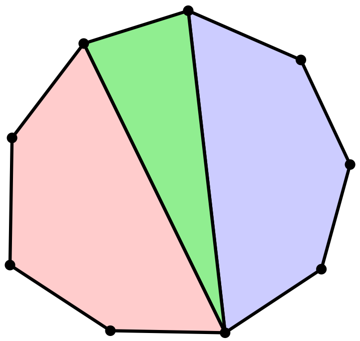
The number of triangulations of n points in convex position is $\frac{1}{n-1}\binom{2n-4}{n-2} = C_{n-2}$.

Number of triangulations

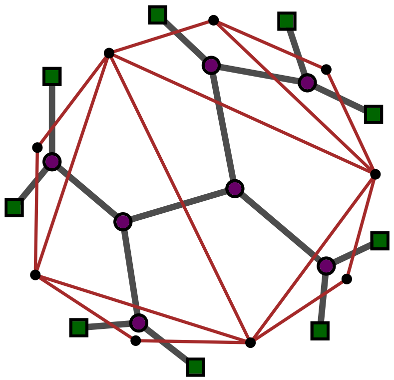
How many triangulations on 4 points? 5 points?

The number of triangulations of n points in convex position is $\frac{1}{n-1}\binom{2n-4}{n-2} = C_{n-2}$.

Direct proof:



"bijective" proof:

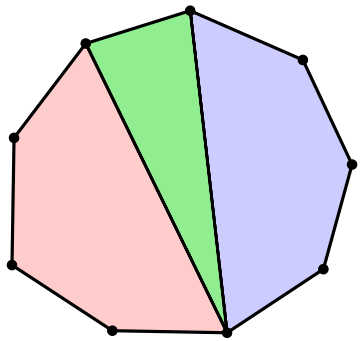


Number of triangulations

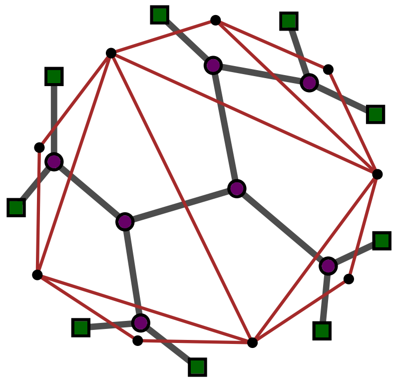
How many triangulations on 4 points? 5 points?

The number of triangulations of n points in convex position is $\frac{1}{n-1}\binom{2n-4}{n-2} = C_{n-2}$.

Direct proof:



"bijective" proof:



Grows fast: $C_n \sim \frac{4^n}{n\sqrt{\pi n}}$

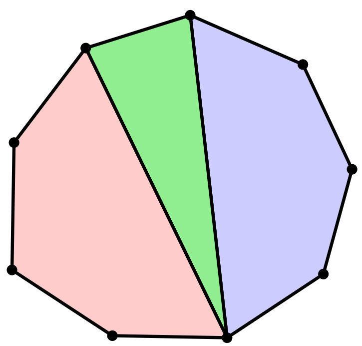
First numbers: $3(1), 4(2), 5(5), 6(14), \ldots, 10(16796), \ldots, 20(6564120420)\ldots$

Number of triangulations

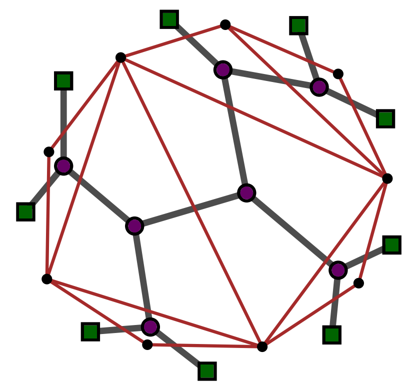
How many triangulations on 4 points? 5 points?

The number of triangulations of n points in convex position is $\frac{1}{n-1}\binom{2n-4}{n-2} = C_{n-2}$.

Direct proof:



"bijective" proof:



Grows fast: $C_n \sim \frac{4^n}{n\sqrt{\pi n}}$

First numbers: $3(1), 4(2), 5(5), 6(14), \ldots, 10(16796), \ldots, 20(6564120420)\ldots$

For points in arbitrary position: $\Omega(8.48^n)$ [2007] and $O(30^n)$ [2009].

Number of edges and triangles

Theorem. Let P be a set of n points in the plane, not all collinear. Let k be the number of points in P that lie on the boundary of the convex hull of P. Any triangulation of P has $2n - 2 - k$ triangles and $3n - 3 - k$ edges.

Number of edges and triangles

Theorem. Let P be a set of n points in the plane, not all collinear. Let k be the number of points in P that lie on the boundary of the convex hull of P. Any triangulation of P has $2n - 2 - k$ triangles and $3n - 3 - k$ edges.

Idea?

Number of edges and triangles

Theorem. Let P be a set of n points in the plane, not all collinear. Let k be the number of points in P that lie on the boundary of the convex hull of P. Any triangulation of P has $2n - 2 - k$ triangles and $3n - 3 - k$ edges.

Proof: $e = \#$ of edges, $t = \#$ of triangles.

Counting edge/face incidences (including unbounded face) $\Rightarrow 2e = 3t + k$

Euler's relation: $n - e + t = 2$

Substitute. $\square$

Number of edges and triangles

Theorem. Let P be a set of n points in the plane, not all collinear. Let k be the number of points in P that lie on the boundary of the convex hull of P. Any triangulation of P has $2n - 2 - k$ triangles and $3n - 3 - k$ edges.

Proof: $e = \#$ of edges, $t = \#$ of triangles.

Counting edge/face incidences (including unbounded face) $\Rightarrow 2e = 3t + k$

Euler's relation: $n - e + t = 2$

Substitute. $\square$

The average degree of a point is ≤ 6.

Which triangulation shall we compute?

"Quality" of a triangulation (mesh) as defined in application areas.

Acute triangles are often considered bad.

Ex: In finite element methods acute triangle make approximation harder.

Which triangulation shall we compute?

"Quality" of a triangulation (mesh) as defined in application areas.

Acute triangles are often considered bad.

Ex: In finite element methods acute triangle make approximation harder.

Not always the case, e.g. in anisotropic meshing (fluid dynamics...)

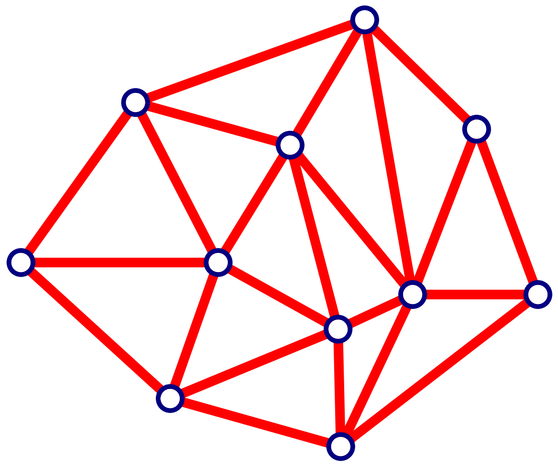
Which triangulation shall we compute?

"Quality" of a triangulation (mesh) as defined in application areas.

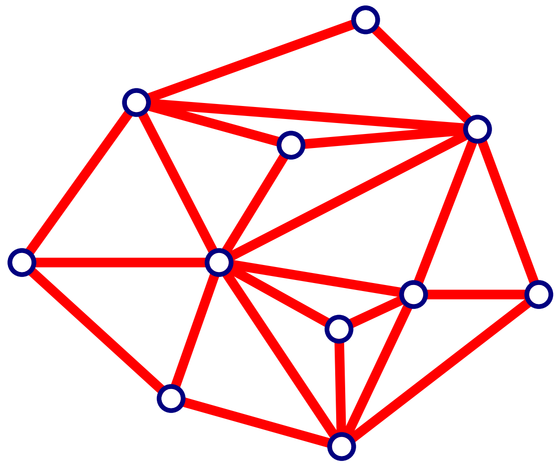
Acute triangles are often considered bad.

Ex: In finite element methods acute triangle make approximation harder.

Not always the case, e.g. in anisotropic meshing (fluid dynamics...)



better than



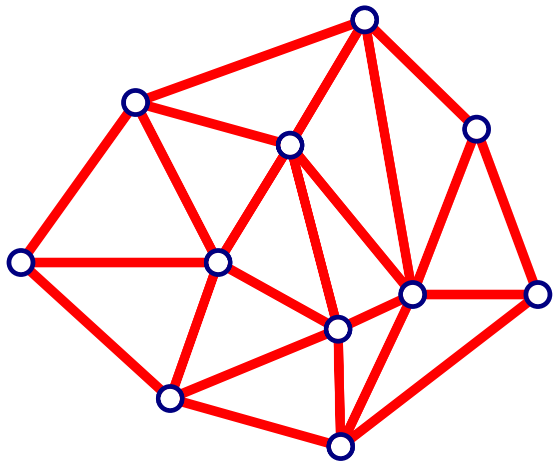
Which triangulation shall we compute?

"Quality" of a triangulation (mesh) as defined in application areas.

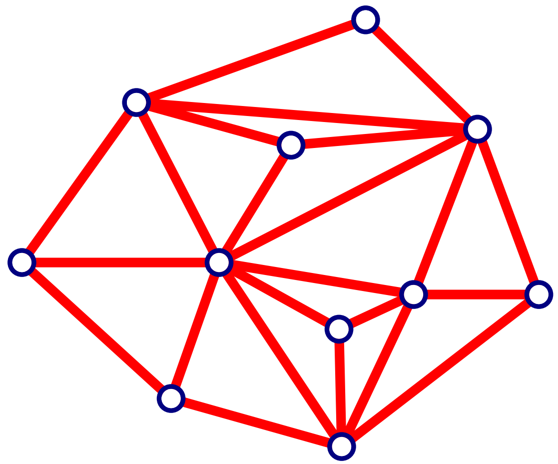
Acute triangles are often considered bad.

Ex: In finite element methods acute triangle make approximation harder.

Not always the case, e.g. in anisotropic meshing (fluid dynamics...)



better than

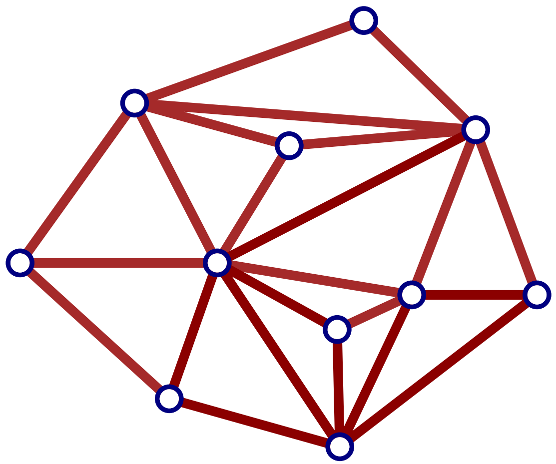


Associate to every triangulation the vector of angles sorted from smallest to largest.

Let's compute the triangulation with lexicographically smallest vector of angles.

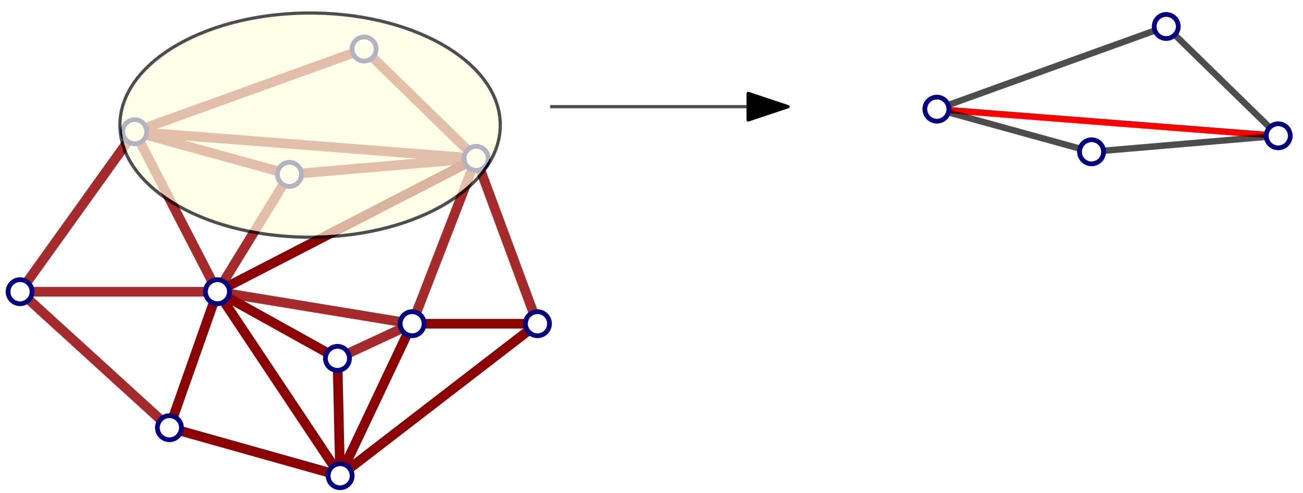
Legal edges

Consider an edge of a triangulation incident to two triangles forming a convex quadrangle.



Legal edges

Consider an edge of a triangulation incident to two triangles forming a convex quadrangle.

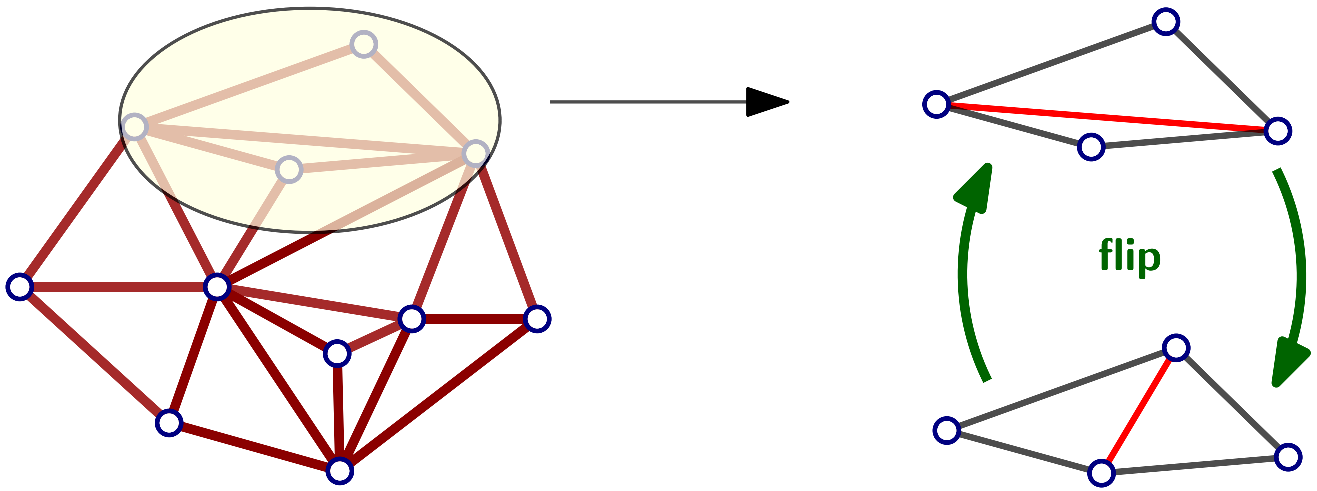


Legal edges

Consider an edge of a triangulation incident to two triangles forming a convex quadrangle.

Exchanging the edge for the other diagonal of the quadrangle yields a new triangulation.

This is called "flipping" the edge.

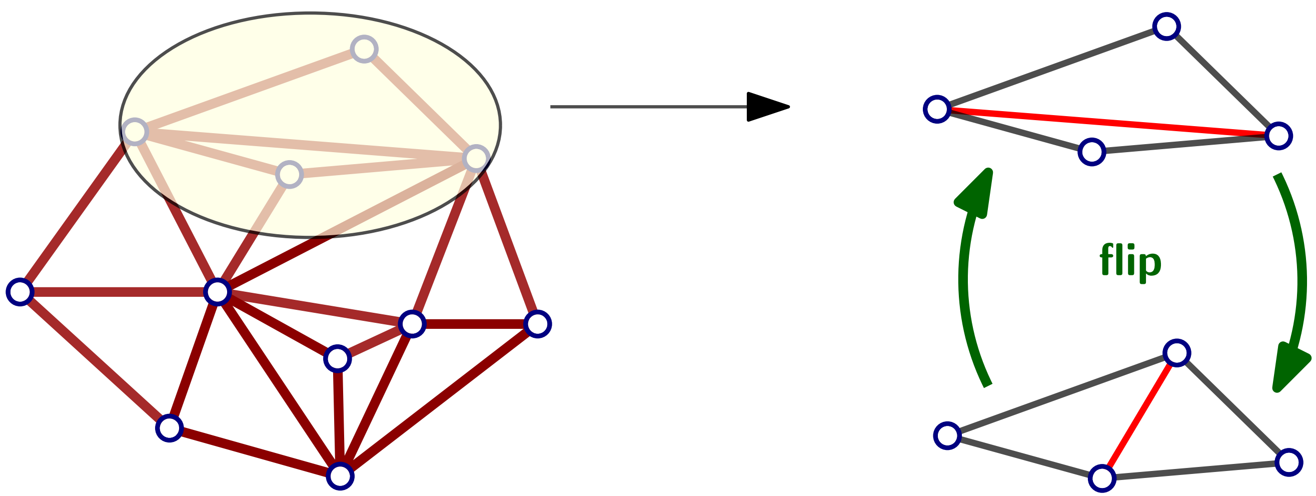


Legal edges

Consider an edge of a triangulation incident to two triangles forming a convex quadrangle.

Exchanging the edge for the other diagonal of the quadrangle yields a new triangulation.

This is called "flipping" the edge.



Call an edge illegal if it can be flipped and flipping it decreases the vector of angles.

Call a triangulation legal if it contains no illegal edge.

Computing a legal triangulation

Termination? Correctness? Complexity?

Start from any triangulation.

While there exists an illegal edge,

flip that edge.

Computing a legal triangulation

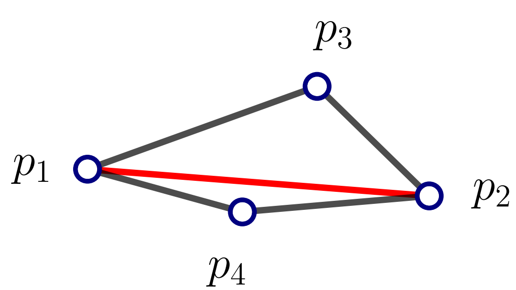
Start from any triangulation.

While there exists an illegal edge,

flip that edge.

Termination? Correctness? Complexity?

An elegant test for edge "illegality"



p_1p_2 is illegal

$\Leftrightarrow$

p_4 lies inside the circle circumscribed to $p_1p_2p_3$

$\Leftrightarrow$

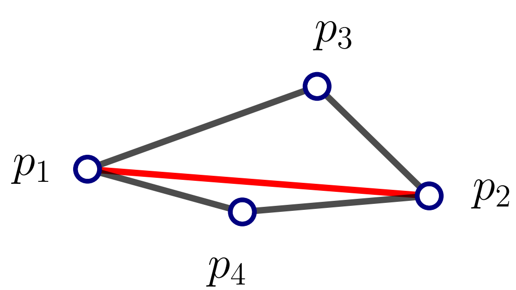
p_3 lies inside the circle circumscribed to $p_1p_2p_4$

Computing a legal triangulation

Termination? Correctness? Complexity?

Start from any triangulation.
While there exists an illegal edge,
flip that edge.

An elegant test for edge "illegality"



p_1p_2 is illegal

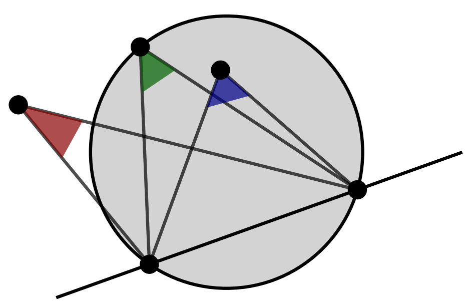
$\Leftrightarrow$

p_4 lies inside the circle circumscribed to $p_1p_2p_3$

$\Leftrightarrow$

p_3 lies inside the circle circumscribed to $p_1p_2p_4$

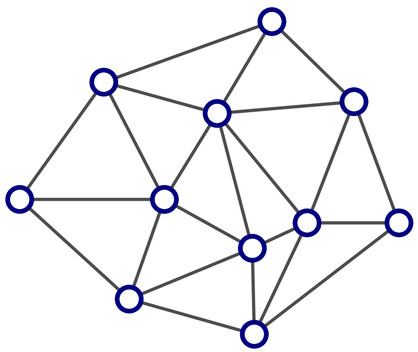
Main ingredient of the proof:



$\leq \quad \leq$

Delaunay triangulations

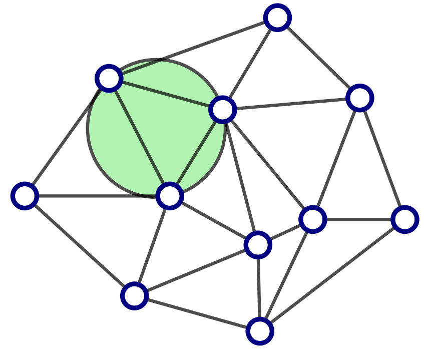
Let P be a set of n points in the plane.



A triangulation is a Delaunay triangulation
$\Leftrightarrow$ the interior of every triangle circumcircle is empty of points of P.

Delaunay triangulations

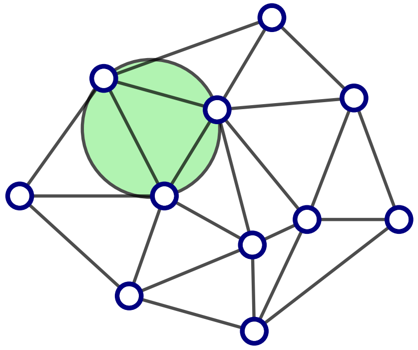
Let P be a set of n points in the plane.



A triangulation is a Delaunay triangulation
$\Leftrightarrow$ the interior of every triangle circumcircle is empty of points of P.

Delaunay triangulations

Let P be a set of n points in the plane.

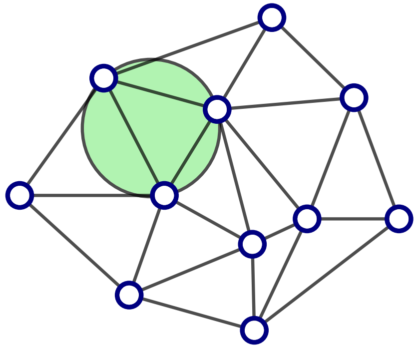


A triangulation is a Delaunay triangulation
$\Leftrightarrow$ the interior of every triangle circumcircle is empty of points of P.

Theorem. A triangulation is legal if and only if it is a Delaunay triangulation.

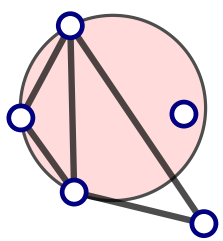
Delaunay triangulations

Let P be a set of n points in the plane.



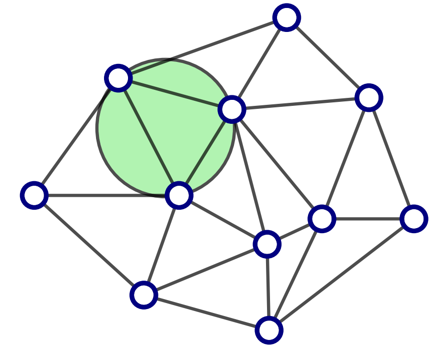
A triangulation is a Delaunay triangulation
$\Leftrightarrow$ the interior of every triangle circumcircle is empty of points of P.

Theorem. A triangulation is legal if and only if it is a Delaunay triangulation.

Proof: Argue that  is impossible in a legal triangulation.

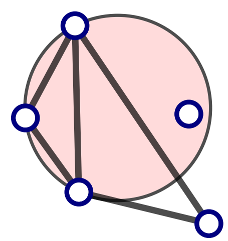
Delaunay triangulations

Let P be a set of n points in the plane.



A triangulation is a Delaunay triangulation
$\Leftrightarrow$ the interior of every triangle circumcircle is empty of points of P.

Theorem. A triangulation is legal if and only if it is a Delaunay triangulation.

Proof: Argue that  is impossible in a legal triangulation.

Theorem. If no 4 points of P are cocircular then P has a unique Delaunay triangulation.

Theorem. All Delaunay triangulations of a point set P have the same minimal angle.

Incremental construction

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in the plane.

For simplicity we assume that P is contained in the triangle $p_1p_2p_3$.

Incremental algorithm:

Add the points one by one.

Maintain a Delaunay triangulation.

Incremental construction

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in the plane.

For simplicity we assume that P is contained in the triangle $p_1 p_2 p_3$.

Incremental algorithm:

Add the points one by one.

Maintain a Delaunay triangulation.

Three sub-problems.

Incremental construction

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in the plane.

For simplicity we assume that P is contained in the triangle $p_1 p_2 p_3$.

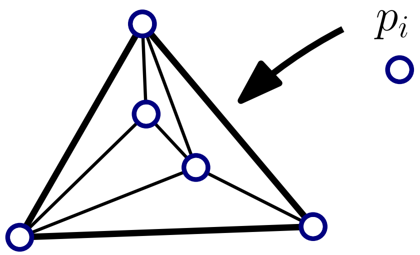
Incremental algorithm:

Add the points one by one.

Maintain a Delaunay triangulation.

Three sub-problems.

Point location



Incremental construction

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in the plane.

For simplicity we assume that P is contained in the triangle $p_1p_2p_3$.

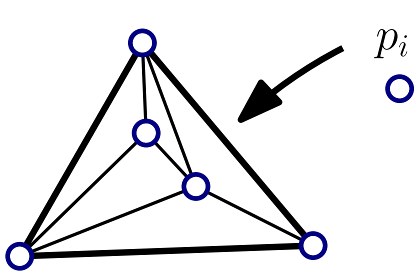
Incremental algorithm:

Add the points one by one.

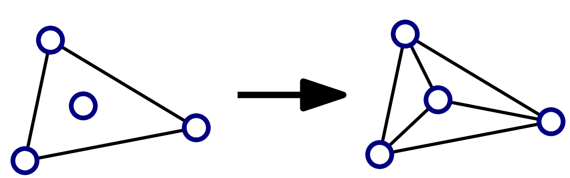
Maintain a Delaunay triangulation.

Three sub-problems.

Point location



Triangle subdivision



Incremental construction

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in the plane.

For simplicity we assume that P is contained in the triangle $p_1p_2p_3$.

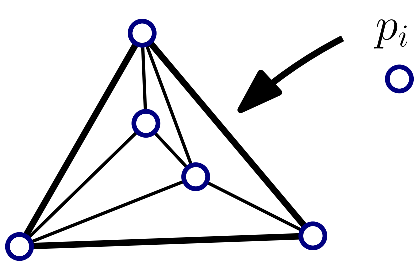
Incremental algorithm:

Add the points one by one.

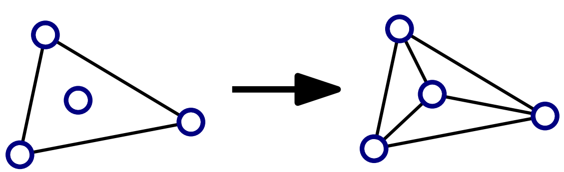
Maintain a Delaunay triangulation.

Three sub-problems.

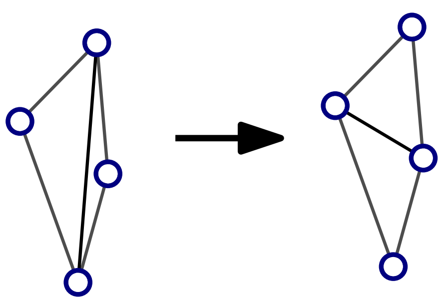
Point location



Triangle subdivision



Correction



Incremental construction

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in the plane.

For simplicity we assume that P is contained in the triangle $p_1p_2p_3$.

Incremental algorithm:

Add the points one by one.

Maintain a Delaunay triangulation.

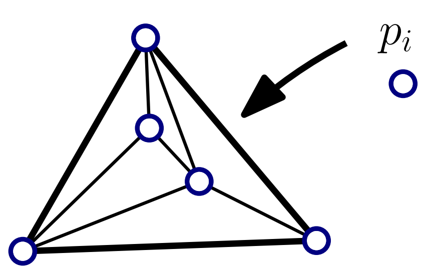
Three sub-problems.

$T \leftarrow \{p_1p_2p_3\}$

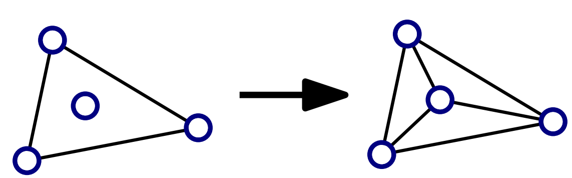
For $i = 4 \ldots n$

- Insert p_i in T.
    - Find a triangle pqr of T containing p_i.
    - $T \leftarrow T \setminus \{pqr\} \cup \{p_ipq, p_iqr, p_irp\}$
    - Make each edge p_ip, p_iq, p_ir legal by successive flips.

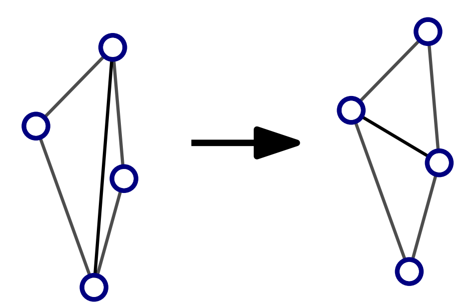
Point location



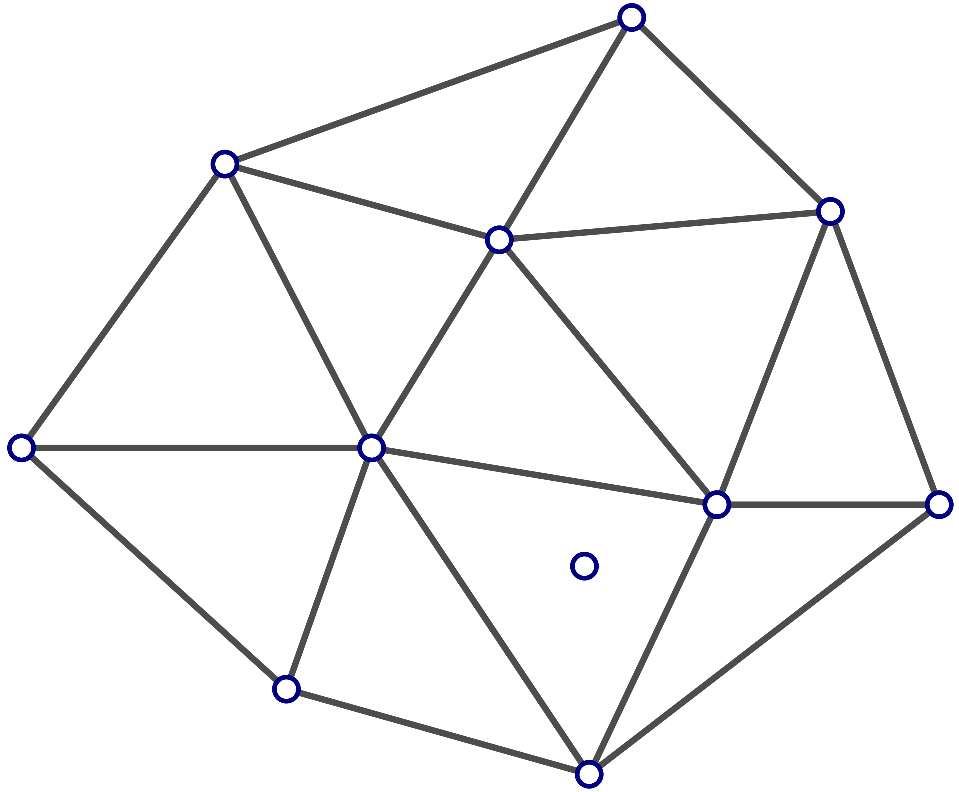
Triangle subdivision



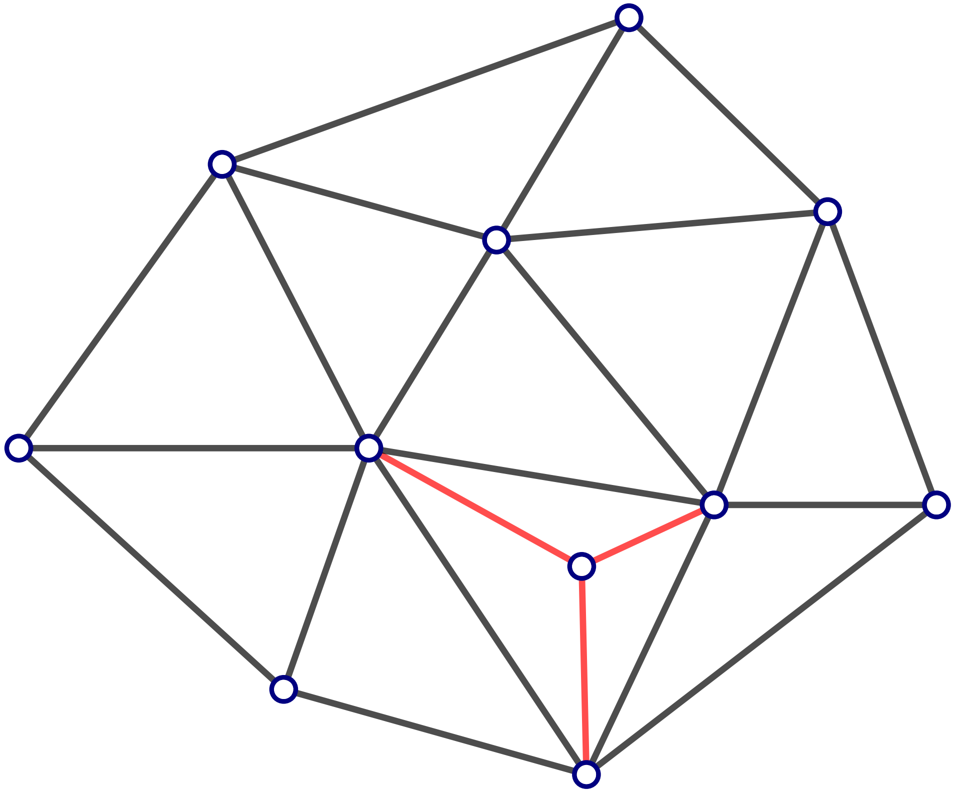
Correction



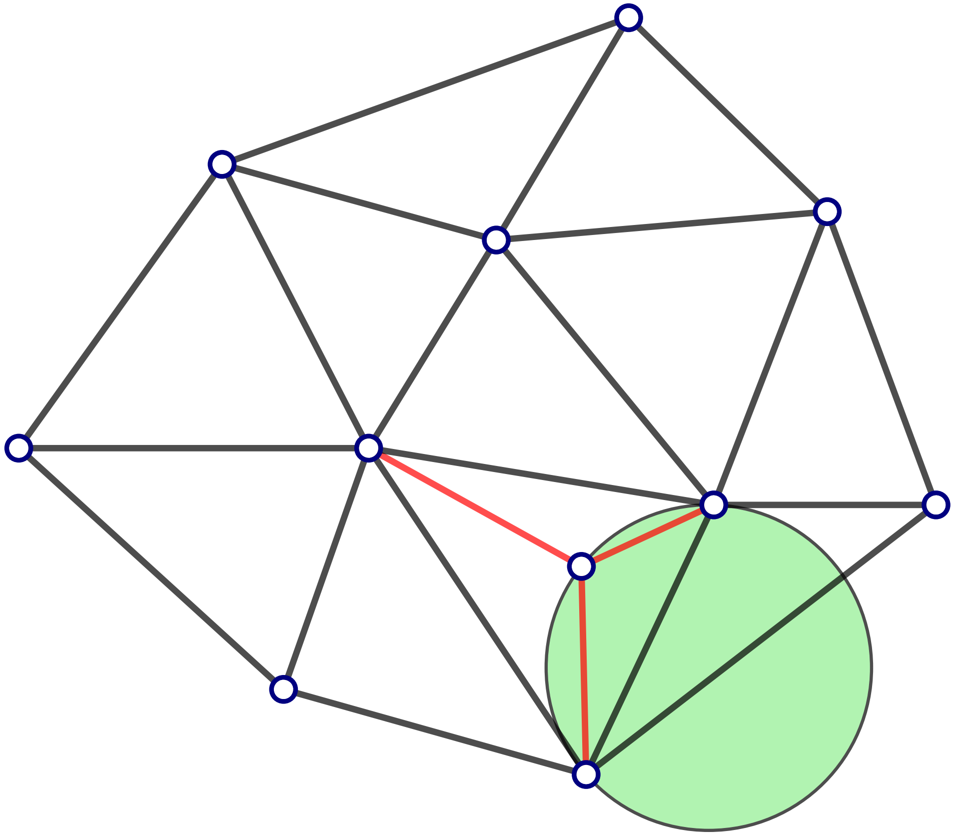
Subdivision - Correction



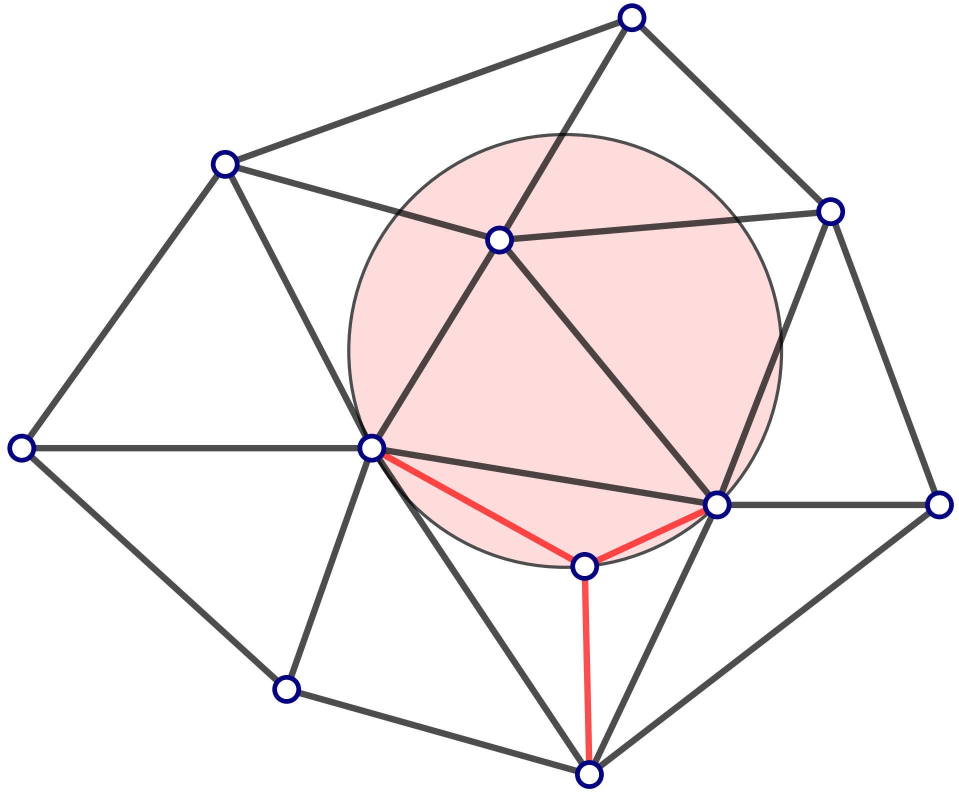
Subdivision - Correction



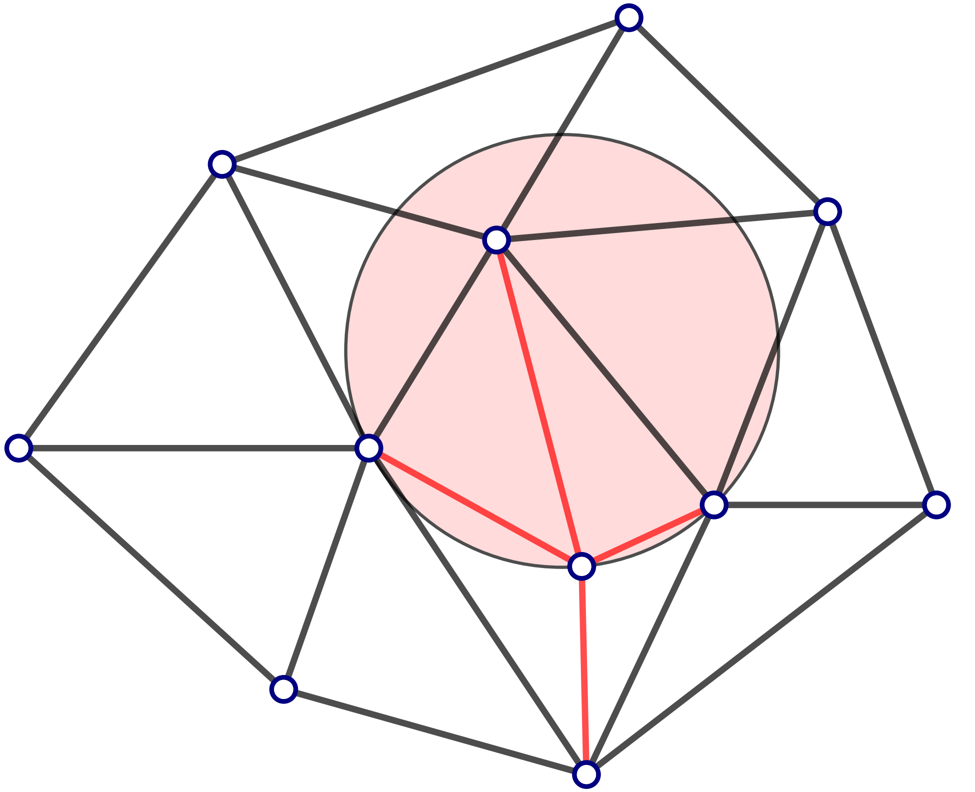
Subdivision - Correction



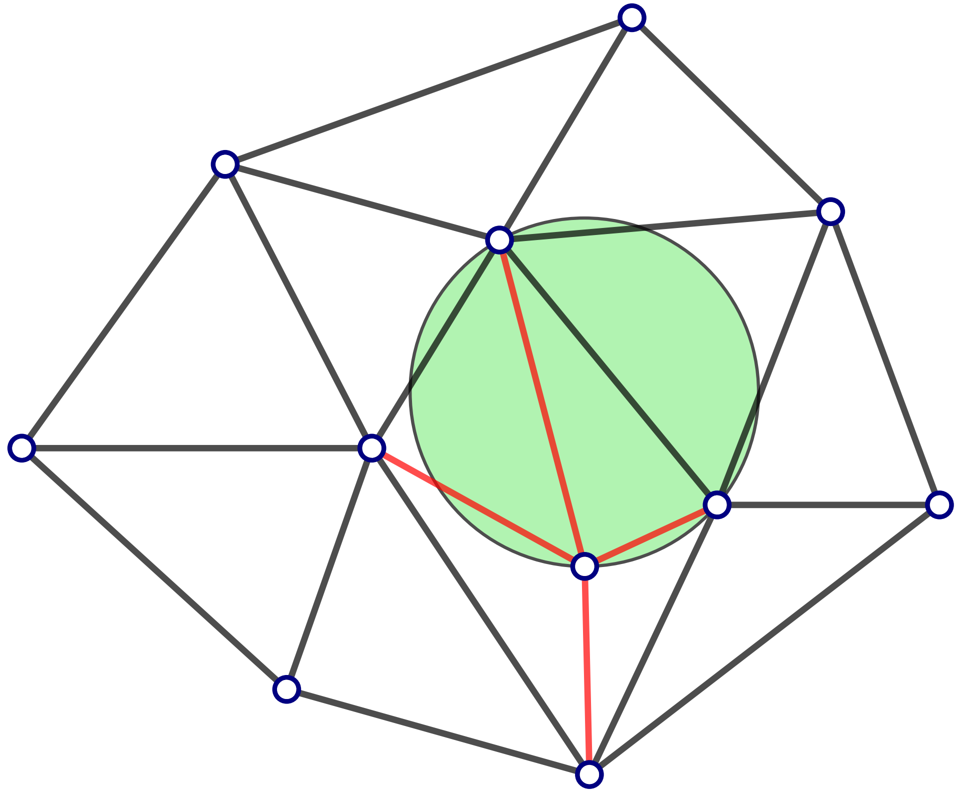
Subdivision - Correction



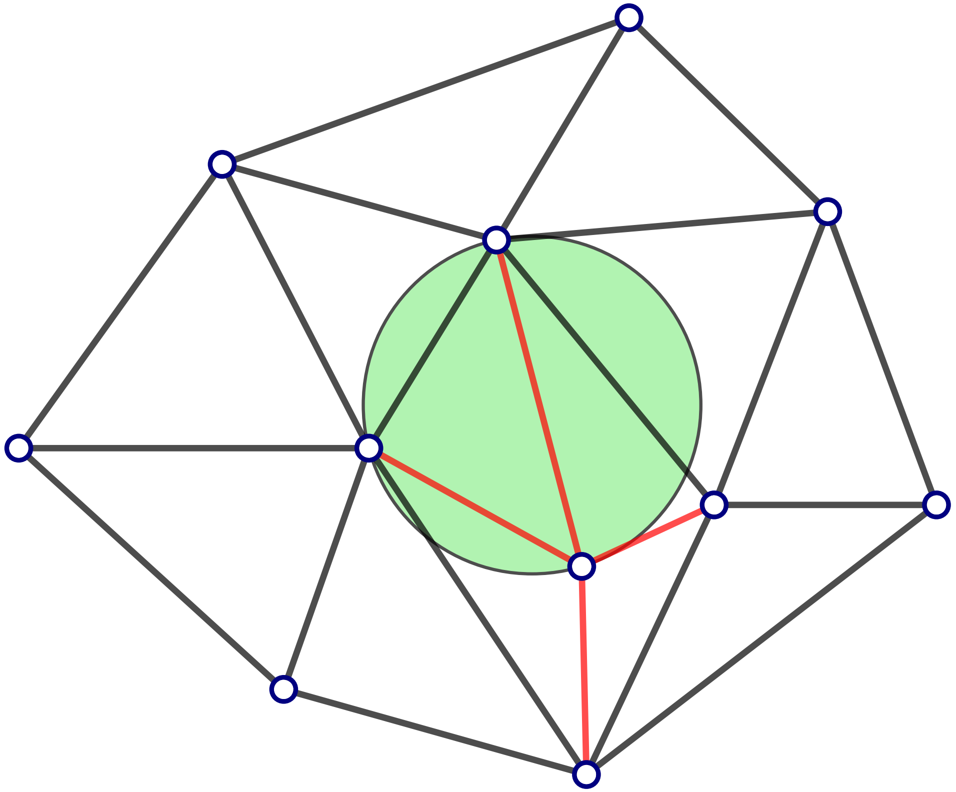
Subdivision - Correction



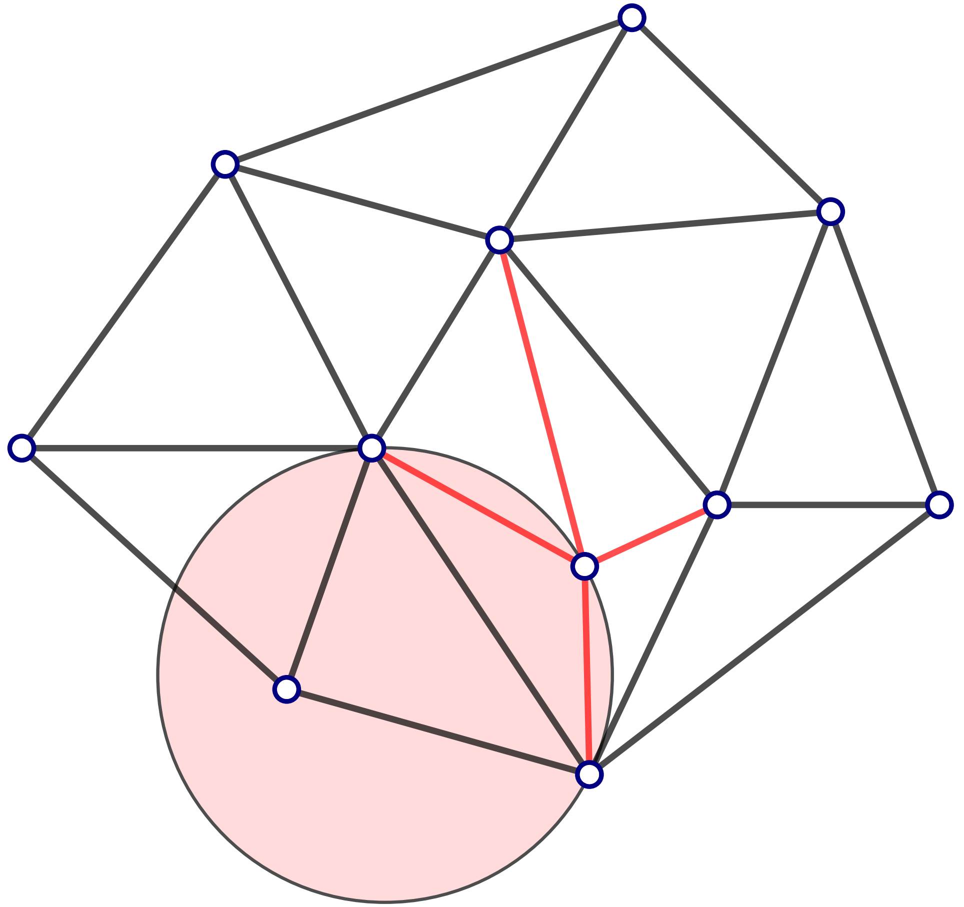
Subdivision - Correction



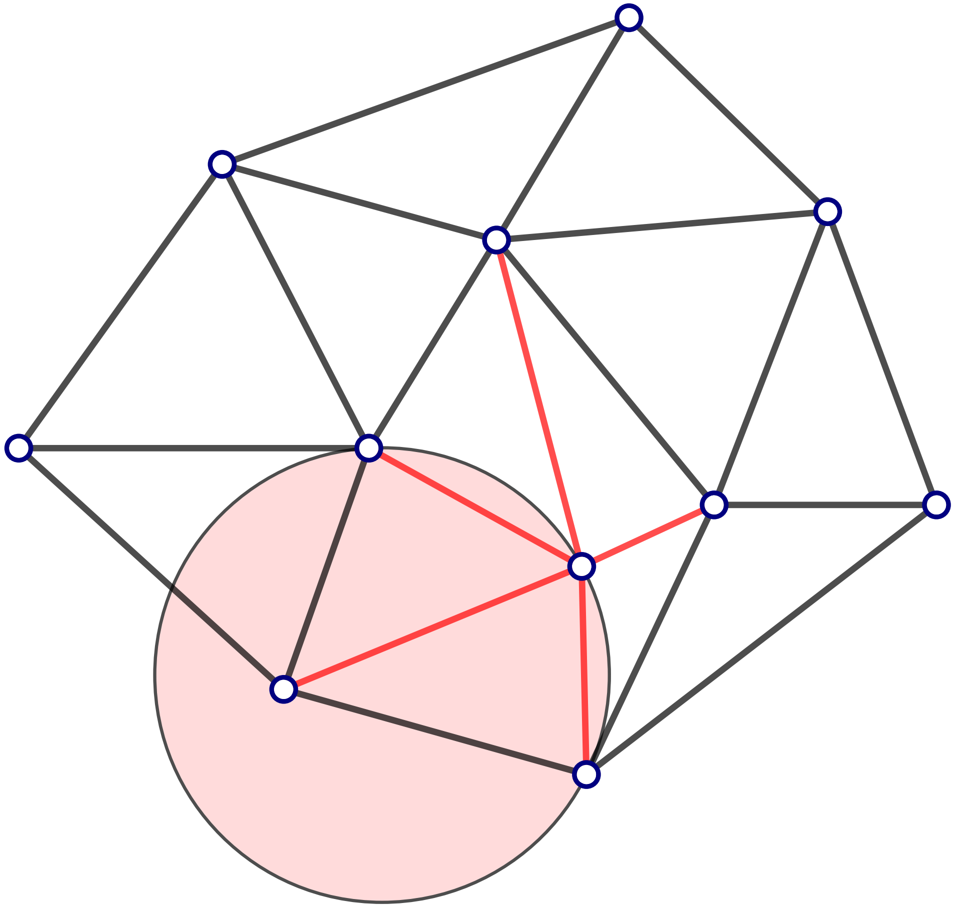
Subdivision - Correction



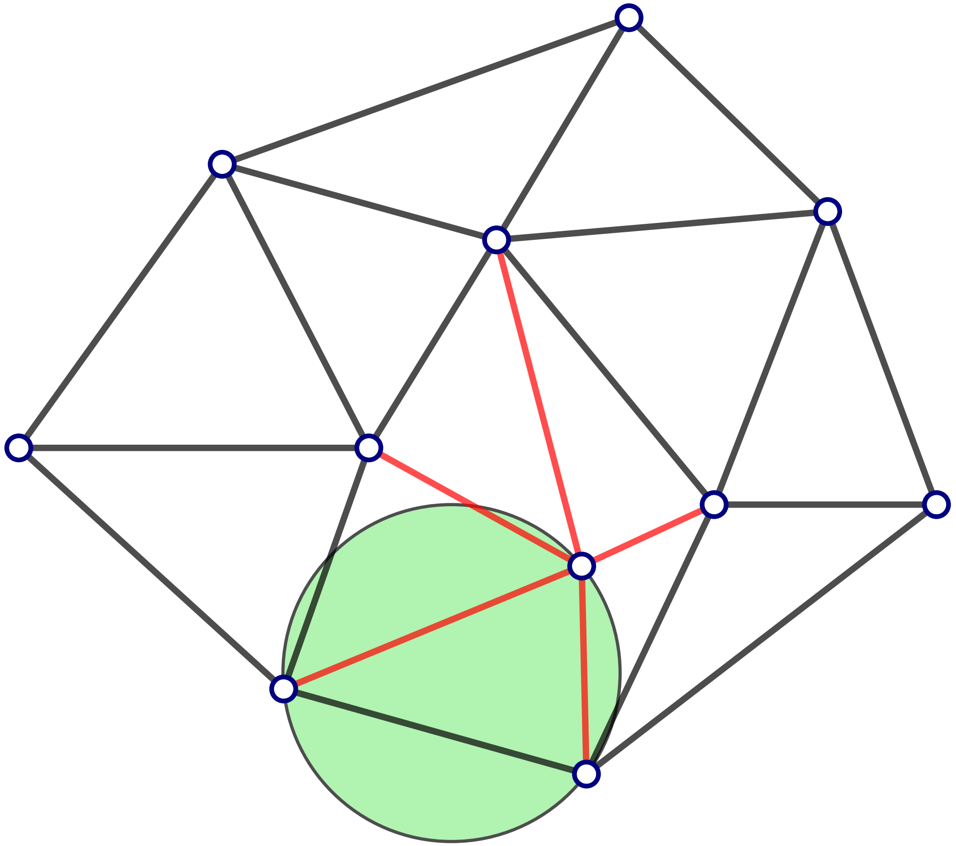
Subdivision - Correction



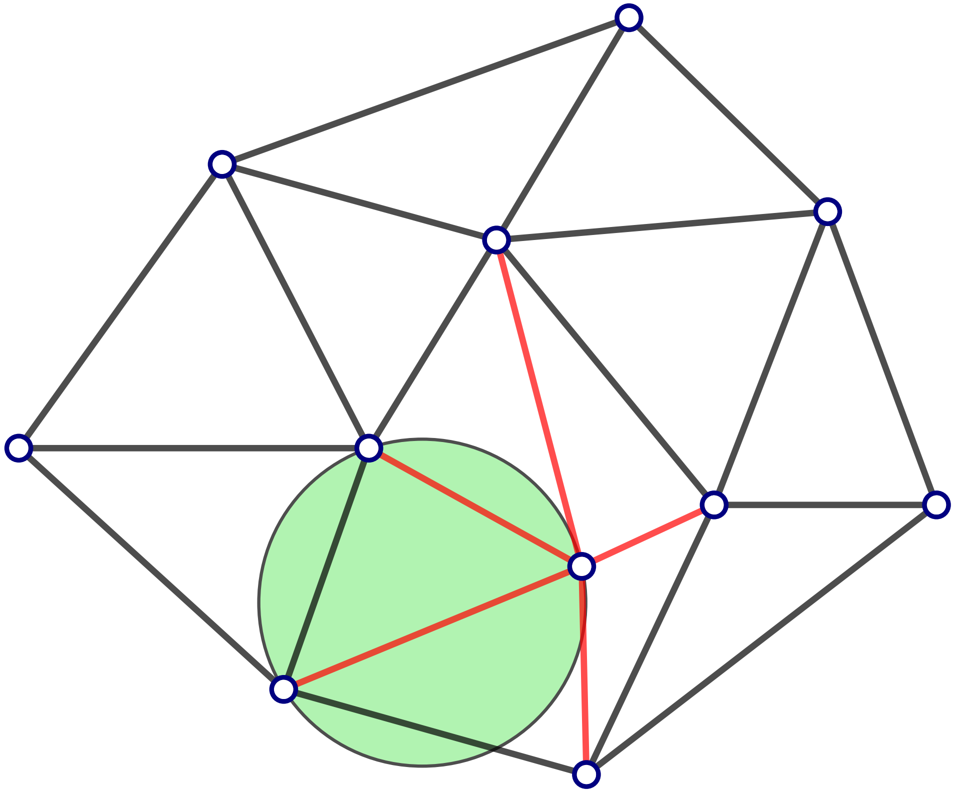
Subdivision - Correction



Subdivision - Correction



Subdivision - Correction



Each flip adds one edge to the new point.

Total cost is $O(d_i)$ where d_i is the degree of p_i in the Delaunay triangulation of $p_1, \ldots, p_i$.

Point location

We use the history of all triangles built to speed up point location.

We maintain a directed acyclic graph during triangle subdivision and edge flips.

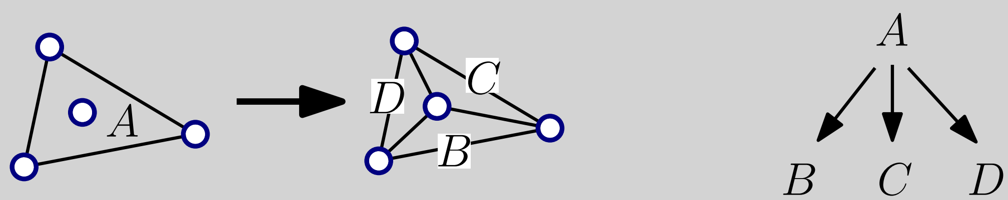
Point location

We use the history of all triangles built to speed up point location.

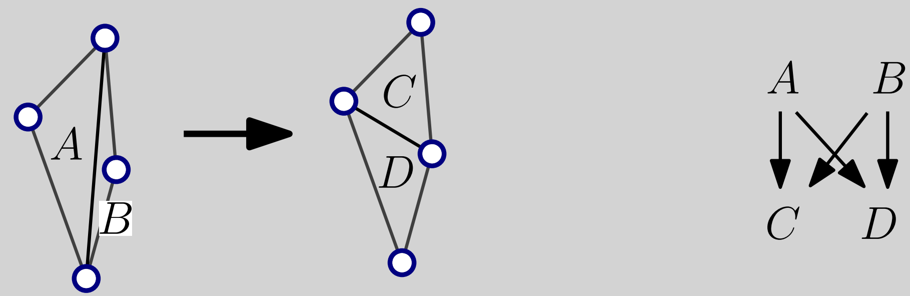
We maintain a directed acyclic graph during triangle subdivision and edge flips.

Initialisation: single node labelled $p_1 p_2 p_3$.

Subdivision:



Flip:



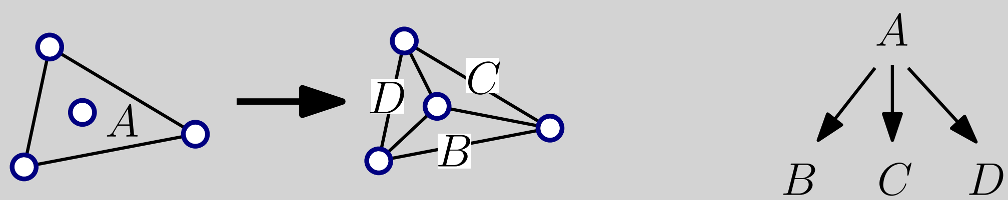
Point location

We use the history of all triangles built to speed up point location.

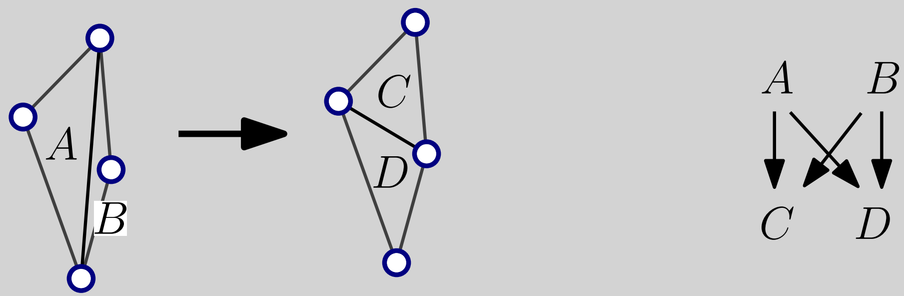
We maintain a directed acyclic graph during triangle subdivision and edge flips.

Initialisation: single node labelled $p_1p_2p_3$.

Subdivision:



Flip:



If a point belongs to the triangle of a node

then it belongs to the triangle to exactly one child of that node.

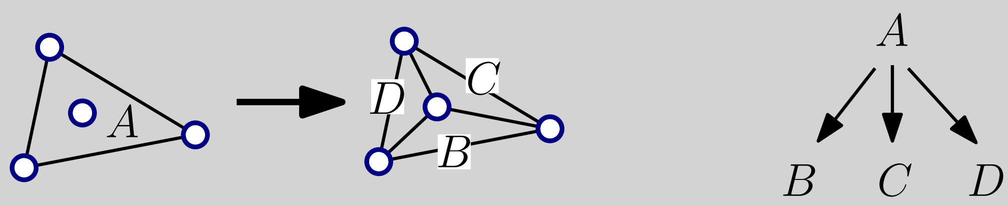
Point location

We use the history of all triangles built to speed up point location.

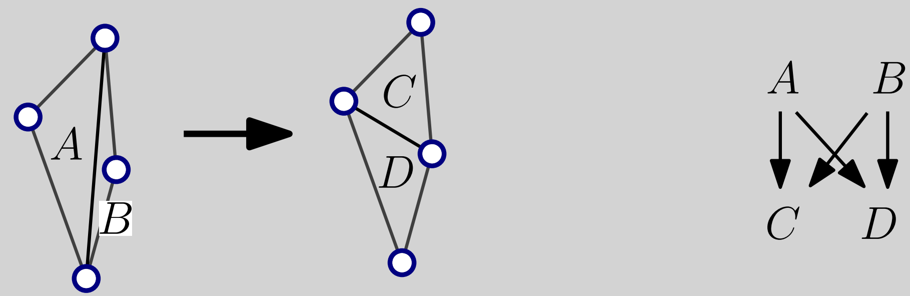
We maintain a directed acyclic graph during triangle subdivision and edge flips.

Initialisation: single node labelled $p_1p_2p_3$.

Subdivision:



Flip:



If a point belongs to the triangle of a node

then it belongs to the triangle to exactly one child of that node.

We start from the root (any p_i belongs to $p_1p_2p_3$) and trickle down

until we find a triangle from the current triangulation (ie a sink of the DAG).

Complexity analysis

Cost $= O\left(\Sigma_i d_i + t_i\right)$.

$T \leftarrow \{p_1 p_2 p_3\}$

For $i = 4 \ldots n$

- Insert p_i in T.
  - Find a triangle pqr of T containing p_i.
  - $T \leftarrow T \setminus \{pqr\} \cup \{p_i pq, p_i qr, p_i rp\}$
  - Make each edge $p_i p$, $p_i q$, $p_i r$ legal by successive flips.

Let d_i be the degree of p_i in the Delaunay triangulation of $p_1, \ldots, p_i$.

Let t_i be the number of triangles to traverse to localize p_i.

Complexity analysis

Cost $= O\left(\Sigma_i d_i + t_i\right)$.

$T \leftarrow \{p_1 p_2 p_3\}$

For $i = 4 \ldots n$

Insert p_i in T.

Find a triangle pqr of T containing p_i.

$T \leftarrow T \setminus \{pqr\} \cup \{p_i pq, p_i qr, p_i rp\}$

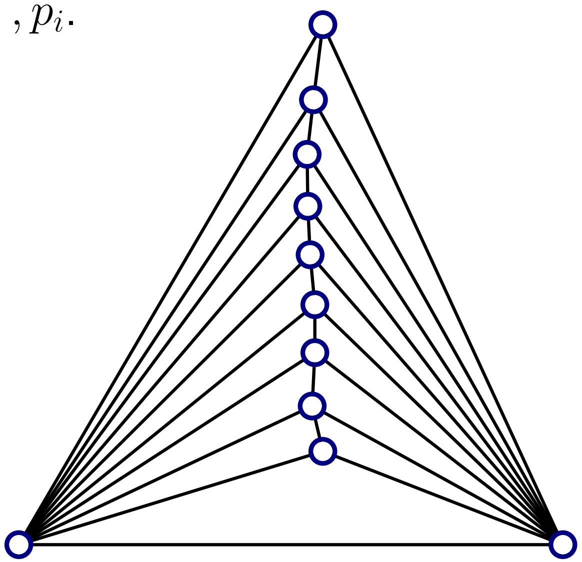
Make each edge $p_i p$, $p_i q$, $p_i r$ legal by successive flips.

Let d_i be the degree of p_i in the Delaunay triangulation of $p_1, \ldots, p_i$.

Let t_i be the number of triangles to traverse to localize p_i.

It can happen that $t_i = i - 1$ for all $i \Rightarrow \Omega(n^2)$ complexity.

The order in which the points are inserted is important.



Complexity analysis

Cost $= O\left(\Sigma_i d_i + t_i\right)$.

$T \leftarrow \{p_1p_2p_3\}$

For $i = 4 \ldots n$

  Insert p_i in T.

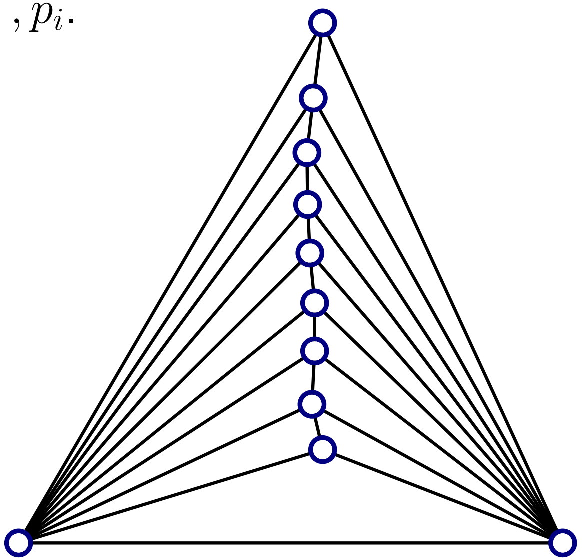
    Find a triangle pqr of T containing p_i.

    $T \leftarrow T \setminus \{pqr\} \cup \{p_ipq, p_iqr, p_irp\}$

    Make each edge p_ip, p_iq, p_ir legal by successive flips.

Let d_i be the degree of p_i in the Delaunay triangulation of $p_1, \ldots, p_i$.

Let t_i be the number of triangles to traverse to localize p_i.



It can happen that $t_i = i - 1$ for all i $\Rightarrow \Omega(n^2)$ complexity.

The order in which the points are inserted is important.

Computing a good order is hard. What about a random order?

Complexity analysis

Cost $= O\left(\Sigma_i d_i + t_i\right)$.

$T \leftarrow \{p_1p_2p_3\}$

Renumber the points $p_4, \ldots, p_n$ randomly.

For $i = 4 \ldots n$

Insert p_i in T.

Find a triangle pqr of T containing p_i.

$T \leftarrow T \setminus \{pqr\} \cup \{p_ipq, p_iqr, p_irp\}$

Make each edge p_ip, p_iq, p_ir legal by successive flips.

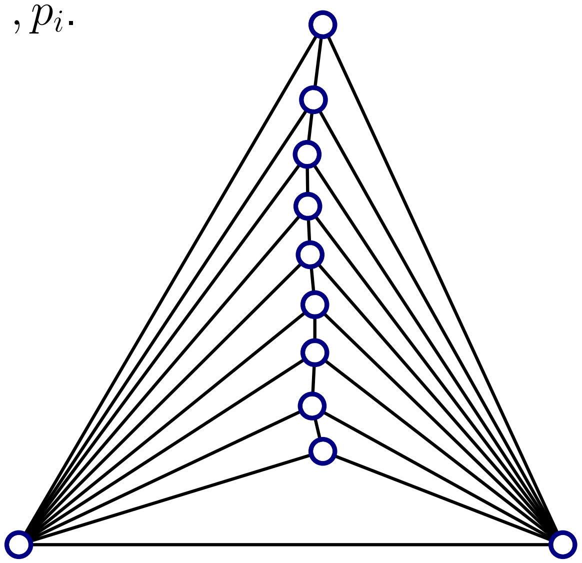
Let d_i be the degree of p_i in the Delaunay triangulation of $p_1, \ldots, p_i$.

Let t_i be the number of triangles to traverse to localize p_i.

It can happen that $t_i = i - 1$ for all $i \Rightarrow \Omega(n^2)$ complexity.

The order in which the points are inserted is important.

Computing a good order is hard. What about a random order?



Randomized incremental construction - complexity analysis

We bound the expected complexity of the algorithm.

Expectation is taken with respect to random internal choices. The input is arbitrary.

$$\textbf{Cost} = E\left[O\left(\Sigma_i d_i + t_i\right)\right] = O\left(\Sigma_i E[d_i] + E[t_i]\right).$$

Randomized incremental construction - complexity analysis

We bound the expected complexity of the algorithm.

Expectation is taken with respect to random internal choices. The input is arbitrary.

$$\textbf{Cost} = E\left[O\left(\Sigma_i d_i + t_i\right)\right] = O\left(\Sigma_i E[d_i] + E[t_i]\right).$$

Condition over the choice of the first i points

Use the bound on the average degree.

$\Rightarrow E[d_i] \leq 6.$

Randomized incremental construction - complexity analysis

We bound the expected complexity of the algorithm.

Expectation is taken with respect to random internal choices. The input is arbitrary.

$$\textbf{Cost} = E\left[O\left(\Sigma_i d_i + t_i\right)\right] = O\left(\Sigma_i E[d_i] + E[t_i]\right).$$

Condition over the choice of the first i points

Use the bound on the average degree.

$\Rightarrow E[d_i] \leq 6.$

Charge deleted triangles to point/triangle incidences.

Amortization theorem (not trivial).

$\Rightarrow E[\Sigma_i t_i] = O(n \log n).$

Randomized incremental construction - complexity analysis

We bound the expected complexity of the algorithm.

Expectation is taken with respect to random internal choices. The input is arbitrary.

$$\textbf{Cost} = E\left[O\left(\Sigma_i d_i + t_i\right)\right] = O\left(\Sigma_i E[d_i] + E[t_i]\right).$$

Condition over the choice of the first i points

Use the bound on the average degree.

$\Rightarrow E[d_i] \leq 6.$

Charge deleted triangles to point/triangle incidences.

Amortization theorem (not trivial).

$\Rightarrow E[\Sigma_i t_i] = O(n \log n).$

Generating a random permutation on $\{3, \ldots, n\}$ can be done in $O(n \log n)$ time.

Randomized incremental construction - complexity analysis

We bound the expected complexity of the algorithm.

Expectation is taken with respect to random internal choices. The input is arbitrary.

$$\textbf{Cost} = E\left[O\left(\Sigma_i d_i + t_i\right)\right] = O\left(\Sigma_i E[d_i] + E[t_i]\right).$$

Condition over the choice of the first i points

Use the bound on the average degree.

$\Rightarrow E[d_i] \leq 6.$

Charge deleted triangles to point/triangle incidences.

Amortization theorem (not trivial).

$\Rightarrow E[\Sigma_i t_i] = O(n \log n).$

Generating a random permutation on $\{3, \ldots, n\}$ can be done in $O(n \log n)$ time.

Theorem. We can compute a Delaunay triangulation of n points in $\mathbb{R}^2$ in $O(n \log n)$ time.

Expected running time of a randomized algorithm.

Higher dimension

Delaunay triangulations generalize to arbitrary dimension (empty circumscribed hypersphere).

Higher dimension

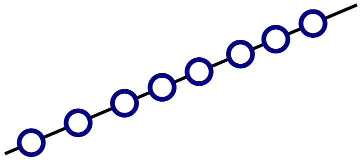
Delaunay triangulations generalize to arbitrary dimension (empty circumscribed hypersphere).

What is the complexity of a Delaunay triangulation of...

Higher dimension

Delaunay triangulations generalize to arbitrary dimension (empty circumscribed hypersphere).

What is the complexity of a Delaunay triangulation of...

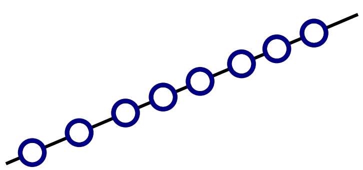


?

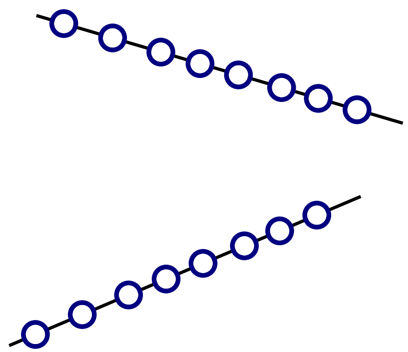
Higher dimension

Delaunay triangulations generalize to arbitrary dimension (empty circumscribed hypersphere).

What is the complexity of a Delaunay triangulation of...



?

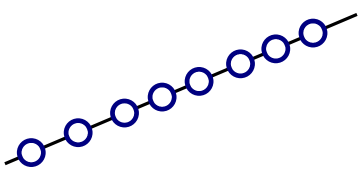


?

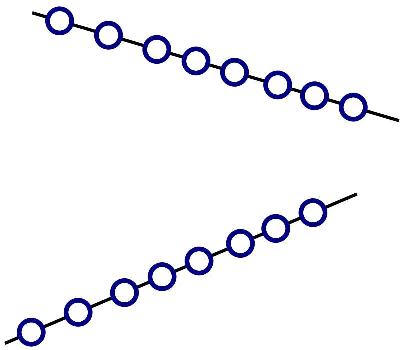
Higher dimension

Delaunay triangulations generalize to arbitrary dimension (empty circumscribed hypersphere).

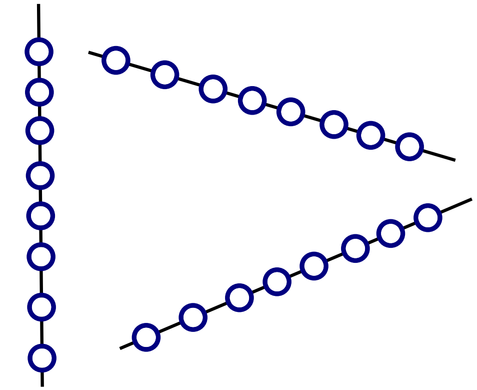
What is the complexity of a Delaunay triangulation of...



?



?

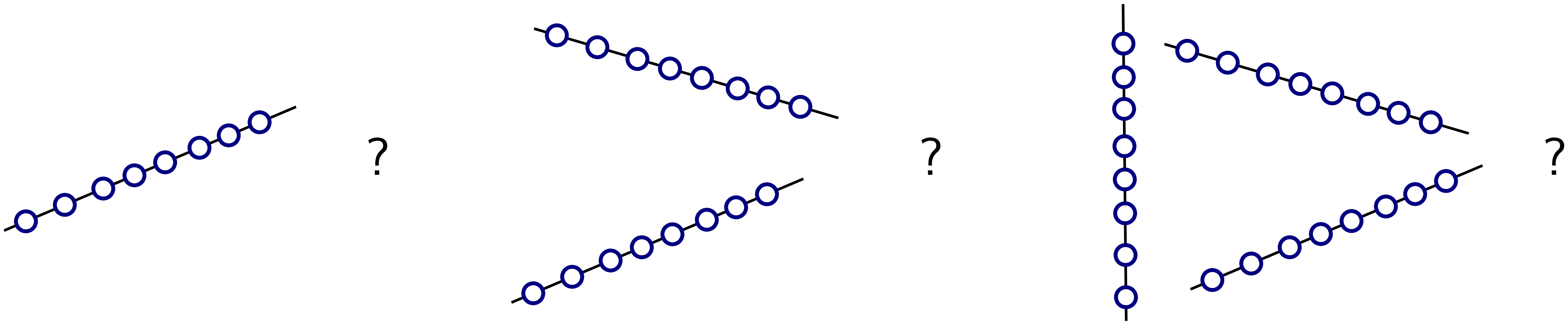


?

Higher dimension

Delaunay triangulations generalize to arbitrary dimension (empty circumscribed hypersphere).

What is the complexity of a Delaunay triangulation of...



$\Theta\left(n^{\lceil \frac{d}{2} \rceil}\right)$ worst-case complexity in dimension $d \geq 3$.

Can be constructed in expected $O\left(n^{\lceil \frac{d}{2} \rceil}\right)$ time (similar approach).

Wrapping up: Delaunay triangulations

Particular triangulation with good geometric properties.

Efficient flip-based algorithm to compute it.

Optimized & flexible implementations available in CGAL.

Randomization is a powerful technique.

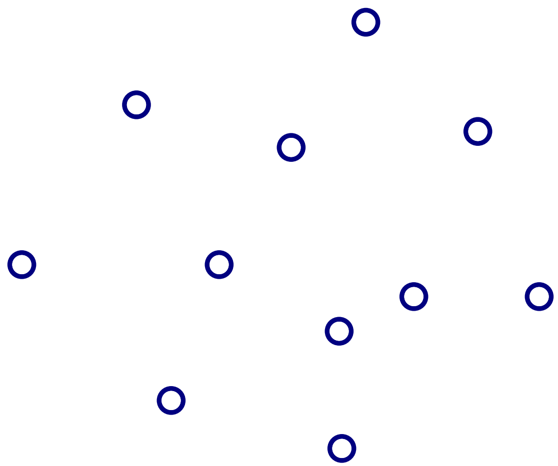
Generic setup: randomized incremental construction.

Many variations: higher dimension, constrained DT, etc...

Question 4

How do you find the nearest post-office?

Input:

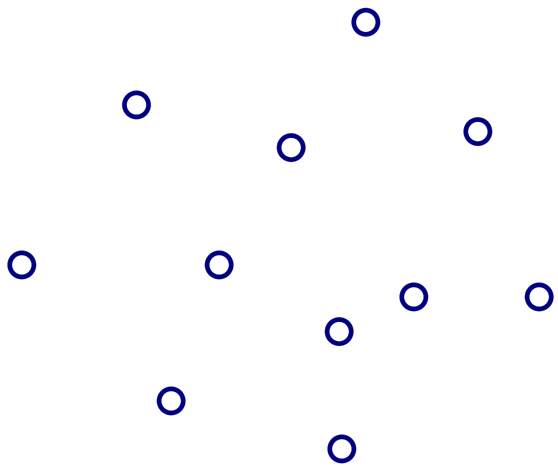


Question 4

How do you find the nearest post-office?

Repeatedly?

Input:



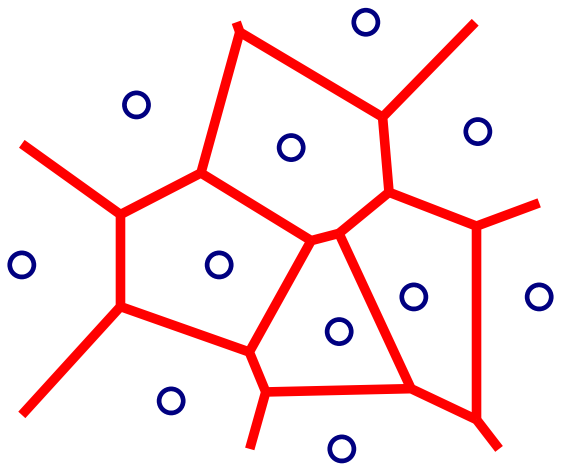
Question 4

How do you find the nearest post-office?

Repeatedly?

Input:

Output

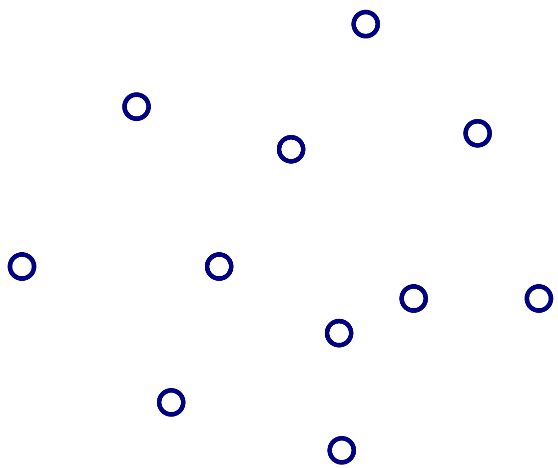


Voronoi diagram - definition

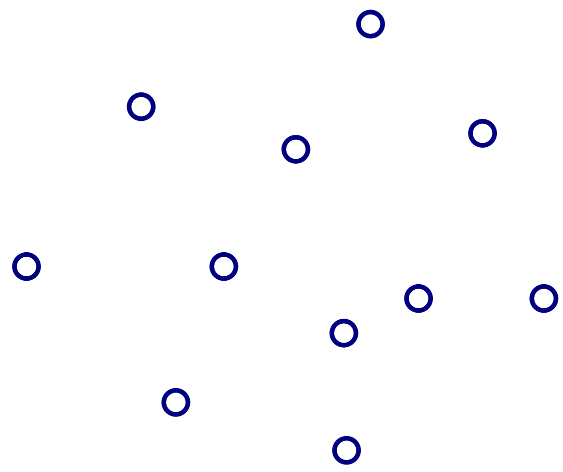
Given a family of sites $p_1, \dots p_n$ in a space with a distance.

Partition the space into regions $R_1, \dots R_n$.

$R_i =$ set of points closer to p_i than to p_j for any $j \neq i$.



Voronoi diagram - definition



Given a family of sites $p_1, \ldots p_n$ in a space with a distance.

Partition the space into regions $R_1, \ldots R_n$.

$R_i =$ set of points closer to p_i than to p_j for any $j \neq i$.

The space can be $\mathbb{R}^2$, $\mathbb{R}^3$, a surface, etc...

The points can be points, disks, polygons, etc...

We focus on the case of sites in the plane.

Voronoi diagram - applications

Voronoi diagrams capture notions of area of influence appearing in many natural sciences.

Voronoi diagram - applications

Voronoi diagrams capture notions of area of influence appearing in many natural sciences.

Voronoi diagram - applications

Voronoi diagrams capture notions of area of influence appearing in many natural sciences.

Used by Descartes to study cosmic fragmentation.

Voronoi diagram - applications

Voronoi diagrams capture notions of area of influence appearing in many natural sciences.

Used by Descartes to study cosmic fragmentation.

Also known as Dirichlet tesselations.

Voronoi diagram - applications

Voronoi diagrams capture notions of area of influence appearing in many natural sciences.

Used by Descartes to study cosmic fragmentation.

Also known as Dirichlet tesselations.

Known to meteorologists as Thiessen polygons.

Voronoi diagram - applications

Voronoi diagrams capture notions of area of influence appearing in many natural sciences.

Used by Descartes to study cosmic fragmentation.

Also known as Dirichlet tesselations.

Known to meteorologists as Thiessen polygons.

Known to chemists as Wigner-Seitz cell.

Voronoi diagram - applications

Voronoi diagrams capture notions of area of influence appearing in many natural sciences.

Used by Descartes to study cosmic fragmentation.

Also known as Dirichlet tesselations.

Known to meteorologists as Thiessen polygons.

Known to chemists as Wigner-Seitz cell.

Called area potentially available to a tree (Brown) and plant polygons (Mead) by biologists.

Voronoi diagram - applications

Voronoi diagrams capture notions of area of influence appearing in many natural sciences.

Used by Descartes to study cosmic fragmentation.

Also known as Dirichlet tesselations.

Known to meteorologists as Thiessen polygons.

Known to chemists as Wigner-Seitz cell.

Called area potentially available to a tree (Brown)
and plant polygons (Mead) by biologists.

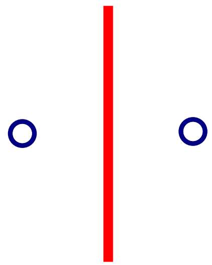
Can be used for natural neighbors interpolation (more later), facility positionning (Voronoi game)...

Structure of a Voronoi diagram

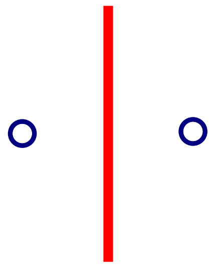
Bisector of p_i and p_j = locus of points at equal distance from p_i and p_j.

Structure of a Voronoi diagram

Bisector of p_i and p_j = locus of points at equal distance from p_i and p_j.



Structure of a Voronoi diagram



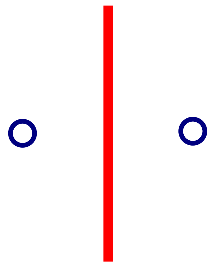
Bisector of p_i and p_j = locus of points at equal distance from p_i and p_j.

The set of points closer to p_i than to p_j is a halfplane bounded by their bissector.

Each region R_i is an intersection of closed halfspaces: a convex (not necessarily bounded) polygon.

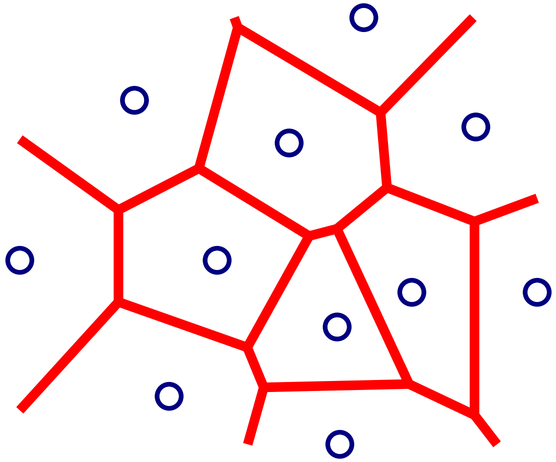
Structure of a Voronoi diagram

Bisector of p_i and p_j = locus of points at equal distance from p_i and p_j.



The set of points closer to p_i than to p_j is a halfplane bounded by their bissector.

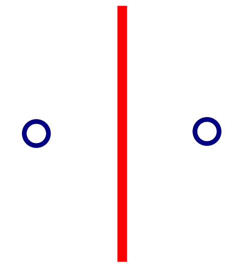
Each region R_i is an intersection of closed halfspaces: a convex (not necessarily bounded) polygon.



Theorem. The Voronoi diagram of n points in the plane has at most $2n-5$ vertices and $3n-6$ edges/ray/lines.

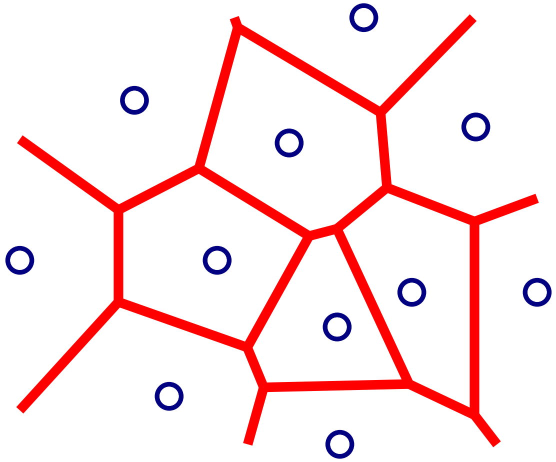
Structure of a Voronoi diagram

Bisector of p_i and p_j = locus of points at equal distance from p_i and p_j.



The set of points closer to p_i than to p_j is a halfplane bounded by their bissector.

Each region R_i is an intersection of closed halfspaces: a convex (not necessarily bounded) polygon.



Theorem. The Voronoi diagram of n points in the plane has at most $2n-5$ vertices and $3n-6$ edges/ray/lines.

Proof. Let v and e be the number of vertices and edges of the VD.

Add a point at infinity where all rays meet.

Euler's formula: $(v+1)-e+n=2$.

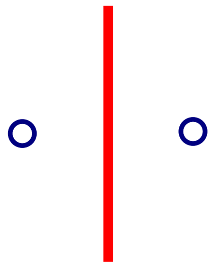
Edge/vertex incidences + every vertex has degree ≥ 3.

$$2e \geq 3(v+1)$$

$\square$

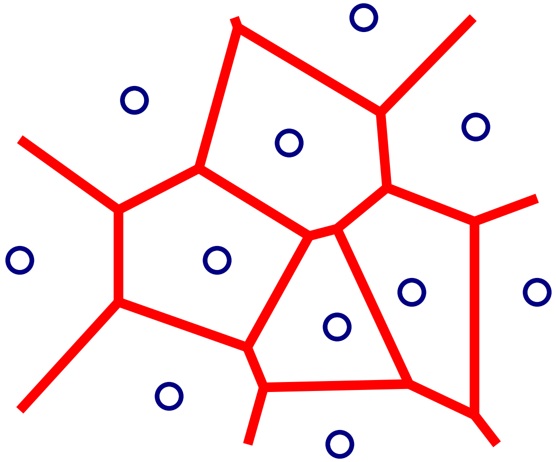
Structure of a Voronoi diagram

Bisector of p_i and p_j = locus of points at equal distance from p_i and p_j.



The set of points closer to p_i than to p_j is a halfplane bounded by their bissector.

Each region R_i is an intersection of closed halfspaces: a convex (not necessarily bounded) polygon.



Theorem. The Voronoi diagram of n points in the plane has at most $2n-5$ vertices and $3n-6$ edges/ray/lines.

Proof. Let v and e be the number of vertices and edges of the VD.

Add a point at infinity where all rays meet.

Euler's formula: $(v+1)-e+n=2$.

Edge/vertex incidences + every vertex has degree ≥ 3.

$$2e \geq 3(v+1)$$

$\square$

The Voronoi diagram of n points has $O(n)$ complexity.

Direct algorithm

Naive algorithm to compute the Voronoi Diagram.

Complexity?

For $i = 1 \dots n$

Compute R_i as intersection of $n-1$ half-planes.

Reconnect everything...

Direct algorithm

Naive algorithm to compute the Voronoi Diagram.

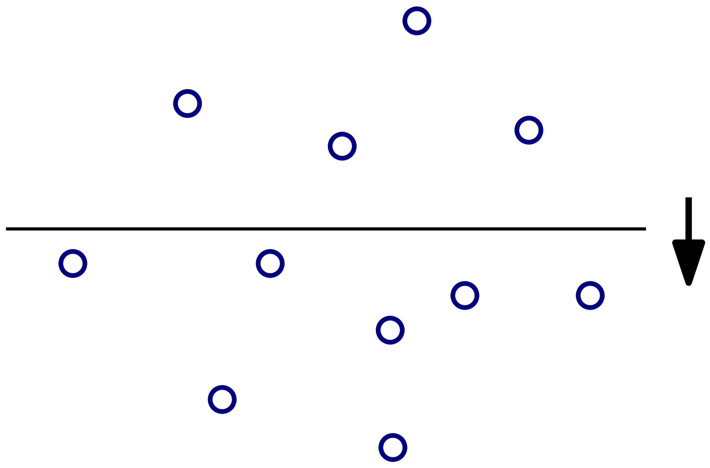
Complexity?

For $i = 1 \dots n$

Compute R_i as intersection of $n-1$ half-planes.

Reconnect everything...

What about a sweep-plane algorithm?



Direct algorithm

Naive algorithm to compute the Voronoi Diagram.

Complexity?

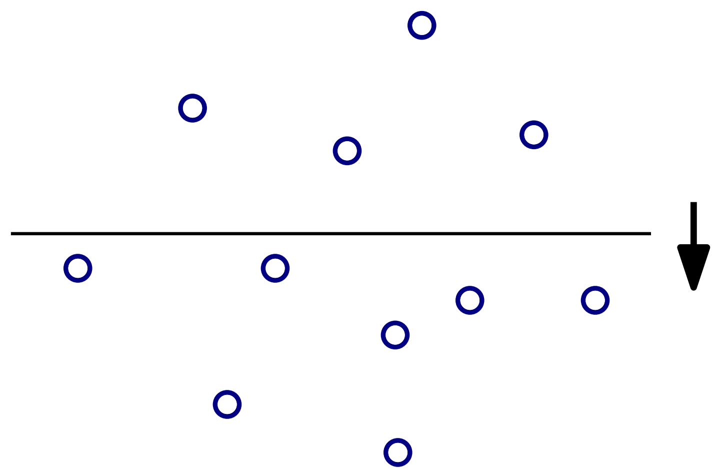
For $i = 1 \ldots n$

 Compute R_i as intersection of $n-1$ half-planes.

Reconnect everything...

What about a sweep-plane algorithm?

Problem: part of the VD above the sweep line is influenced by the sites below the sweep line.



Direct algorithm

Naive algorithm to compute the Voronoi Diagram.

Complexity?

For $i = 1 \ldots n$

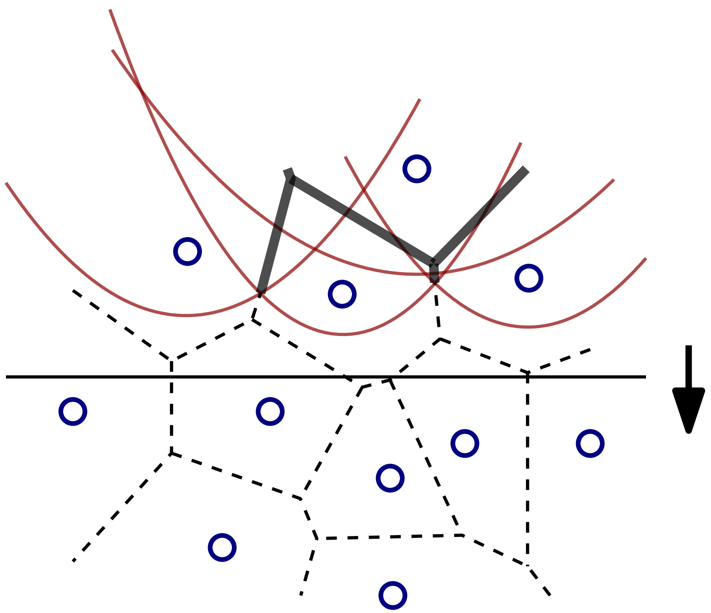
Compute R_i as intersection of $n-1$ half-planes.

Reconnect everything...

What about a sweep-plane algorithm?

Problem: part of the VD above the sweep line is influenced by the sites below the sweep line.

Part of the VD above the sweep line can no longer be influenced.



Direct algorithm

Naive algorithm to compute the Voronoi Diagram.

Complexity?

For $i = 1 \ldots n$

Compute R_i as intersection of $n-1$ half-planes.

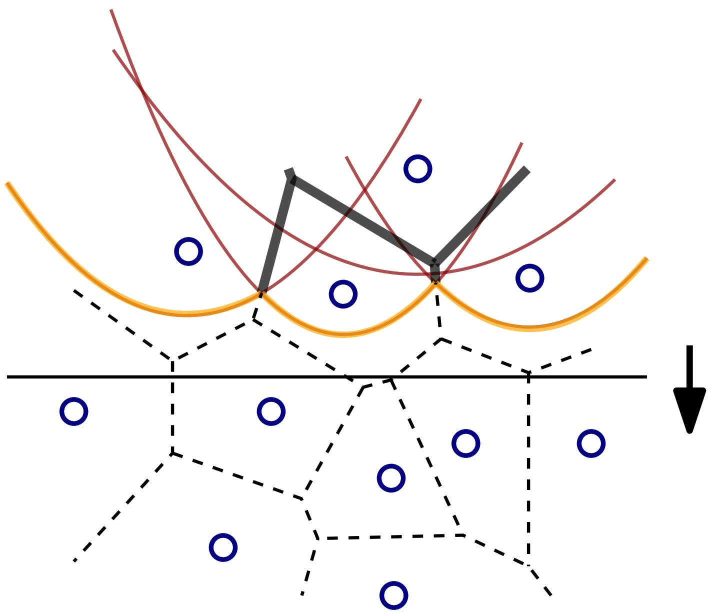
Reconnect everything...

What about a sweep-plane algorithm?

Problem: part of the VD above the sweep line is influenced by the sites below the sweep line.

Part of the VD above the sweep line can no longer be influenced.

Maintain a beach line (lower enveloppe of parabolas).



Direct algorithm

Naive algorithm to compute the Voronoi Diagram.

Complexity?

For $i = 1 \ldots n$
  Compute R_i as intersection of $n-1$ half-planes.
Reconnect everything...

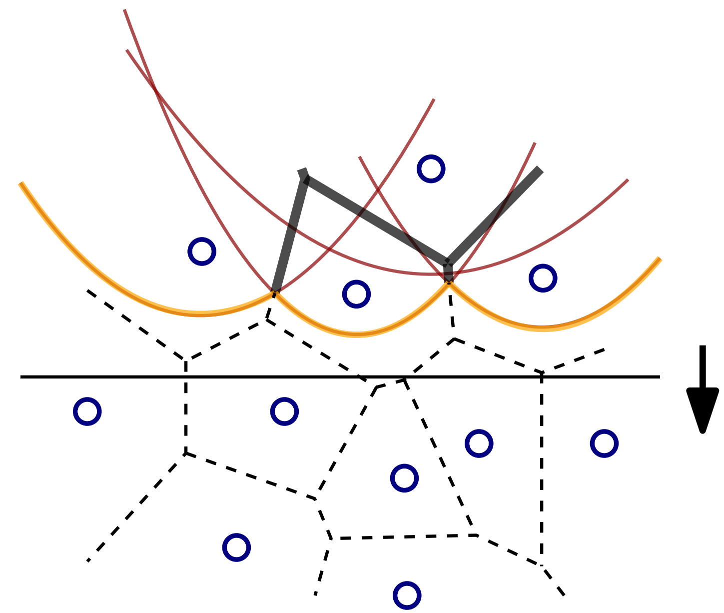
What about a sweep-plane algorithm?

Problem: part of the VD above the sweep line is influenced by the sites below the sweep line.

Part of the VD above the sweep line can no longer be influenced.

Maintain a beach line (lower enveloppe of parabolas).

Characterize the events (non-trivial).



Direct algorithm

Naive algorithm to compute the Voronoi Diagram.

Complexity?

For $i = 1 \ldots n$

  Compute R_i as intersection of $n-1$ half-planes.

Reconnect everything...

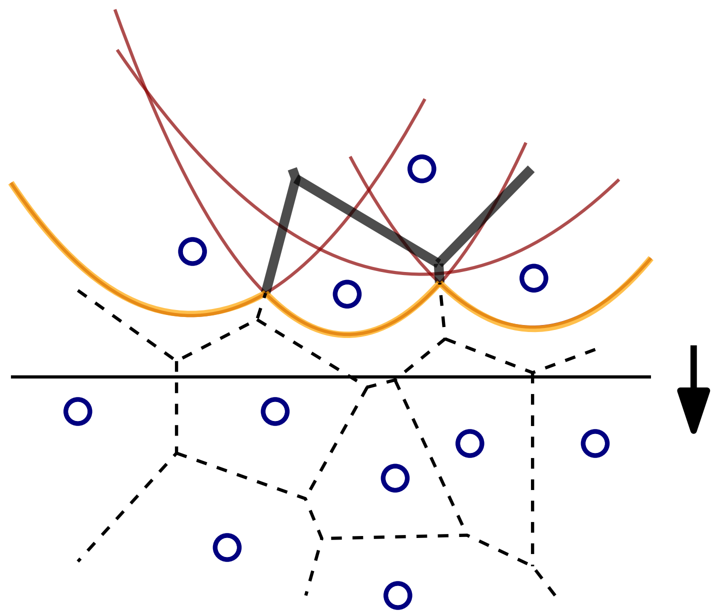
What about a sweep-plane algorithm?

Problem: part of the VD above the sweep line is influenced by the sites below the sweep line.

Part of the VD above the sweep line can no longer be influenced.

Maintain a beach line (lower enveloppe of parabolas).

Characterize the events (non-trivial).

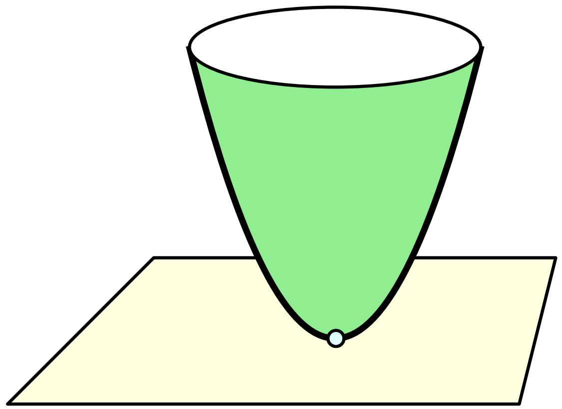


This algorithm takes $O(n \log n)$ time.

Fortune's algorithm.

Lifting $\mathbb{R}^2 \to \mathbb{R}^3$

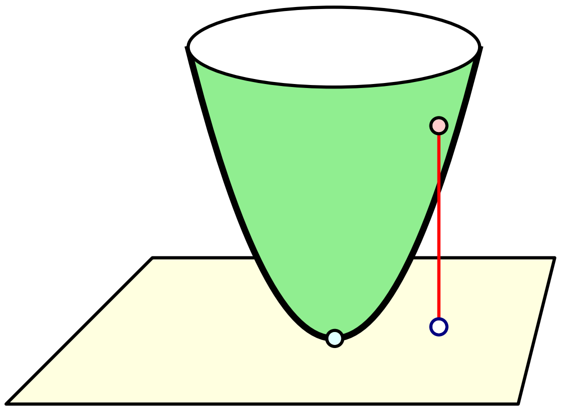
Consider the unit paraboloid $(P) : z = x^2 + y^2$ in $\mathbb{R}^3$.



Lifting $\mathbb{R}^2 \to \mathbb{R}^3$

Consider the unit paraboloid $(P) : z = x^2 + y^2$ in $\mathbb{R}^3$.

"Lift" $q = (x, y) \mapsto q' = (x, y, x^2 + y^2) \in (P)$.

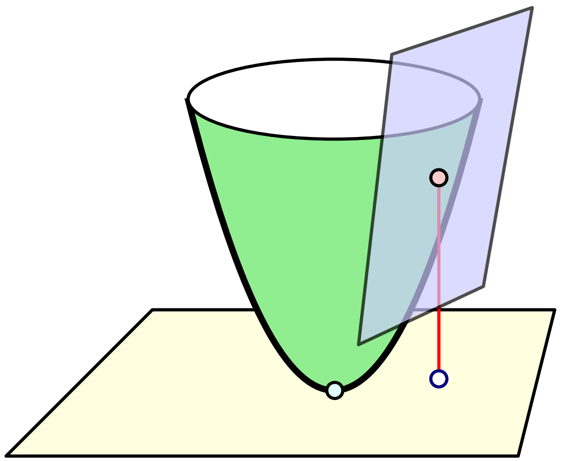


Lifting $\mathbb{R}^2 \to \mathbb{R}^3$

Consider the unit paraboloid $(P) : z = x^2 + y^2$ in $\mathbb{R}^3$.

"Lift" $q = (x, y) \mapsto q' = (x, y, x^2 + y^2) \in (P)$.

Associate to $q' \in (P)$ the plane $h(q')$ tangent to (P) in q'.



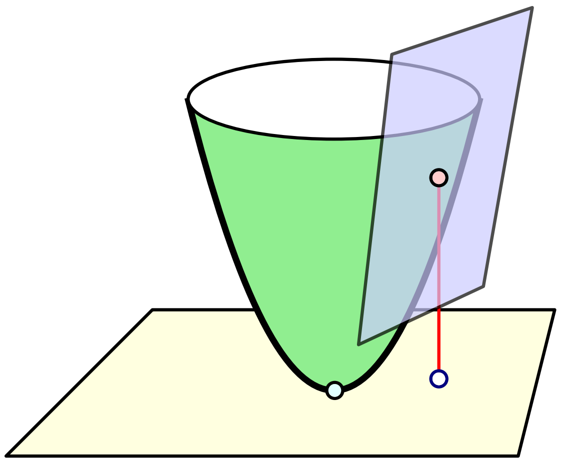
Lifting $\mathbb{R}^2 \to \mathbb{R}^3$

Consider the unit paraboloid $(P) : z = x^2 + y^2$ in $\mathbb{R}^3$.

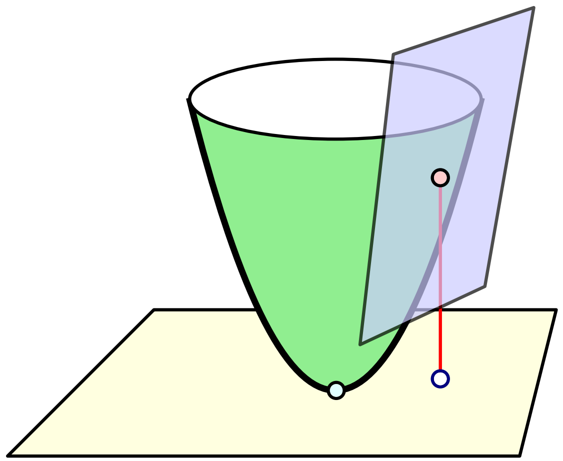
"Lift" $q = (x, y) \mapsto q' = (x, y, x^2 + y^2) \in (P)$.

Associate to $q' \in (P)$ the plane $h(q')$ tangent to (P) in q'.

$q' \mapsto h(q')$ is the duality with respect to (P).



Lifting $\mathbb{R}^2 \to \mathbb{R}^3$



Consider the unit paraboloid $(P) : z = x^2 + y^2$ in $\mathbb{R}^3$.

"Lift" $q = (x, y) \mapsto q' = (x, y, x^2 + y^2) \in (P)$.

Associate to $q' \in (P)$ the plane $h(q')$ tangent to (P) in q'.

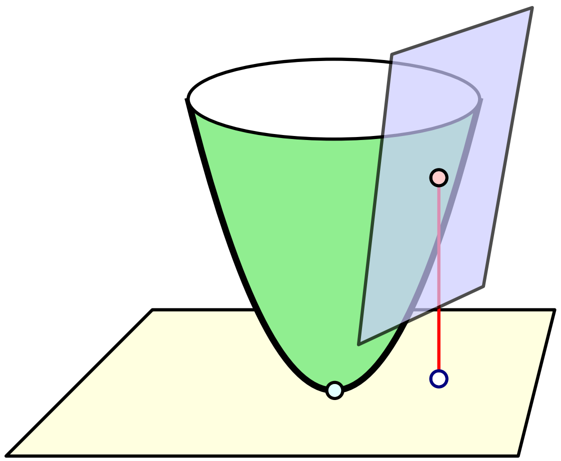
$q' \mapsto h(q')$ is the duality with respect to (P).

Let a and b be two points in the (yellow) plane.

Let a' and b' be their lifts on (P).

Let b'' be the lift of b on $h(a')$.

Lifting $\mathbb{R}^2 \to \mathbb{R}^3$



Consider the unit paraboloid $(P) : z = x^2 + y^2$ in $\mathbb{R}^3$.

"Lift" $q = (x, y) \mapsto q' = (x, y, x^2 + y^2) \in (P)$.

Associate to $q' \in (P)$ the plane $h(q')$ tangent to (P) in q'.

$q' \mapsto h(q')$ is the duality with respect to (P).

Let a and b be two points in the (yellow) plane.

Let a' and b' be their lifts on (P).

Let b'' be the lift of b on $h(a')$.

Equation of $h(a) : 2x_a(x - x_a) + 2y_a(y - y_a) - z + x_a^2 + y_a^2 = 0$

$b'' = (b_x, b_y, 2b_x a_x + 2b_y a_y - (a_x^2 + a_y^2)) \Rightarrow b'b'' = ab^2$

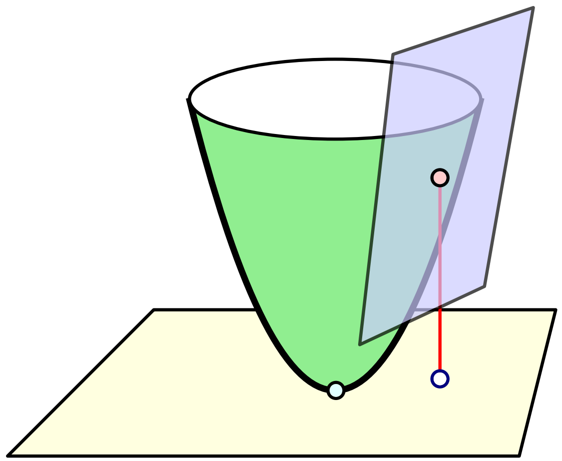
Lifting $\mathbb{R}^2 \to \mathbb{R}^3$

Consider the unit paraboloid $(P) : z = x^2 + y^2$ in $\mathbb{R}^3$.

"Lift" $q = (x, y) \mapsto q' = (x, y, x^2 + y^2) \in (P)$.

Associate to $q' \in (P)$ the plane $h(q')$ tangent to (P) in q'.

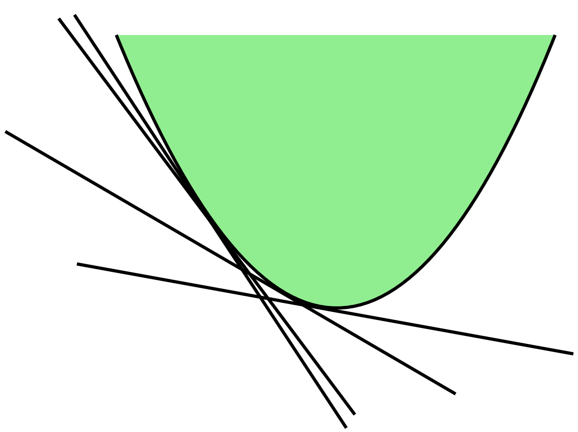
$q' \mapsto h(q')$ is the duality with respect to (P).



Let a and b be two points in the (yellow) plane.

Let a' and b' be their lifts on (P).

Let b'' be the lift of b on $h(a')$.



Equation of $h(a) : 2x_a(x - x_a) + 2y_a(y - y_a) - z + x_a^2 + y_a^2 = 0$

$$b'' = (b_x, b_y, 2b_x a_x + 2b_y a_y - (a_x^2 + a_y^2)) \Rightarrow b'b'' = ab^2$$

Lifting each point of R_i to $h(p'_i)$

lifts the VD to the upper enveloppe of $h(p'_1), \ldots, h(p'_n)$

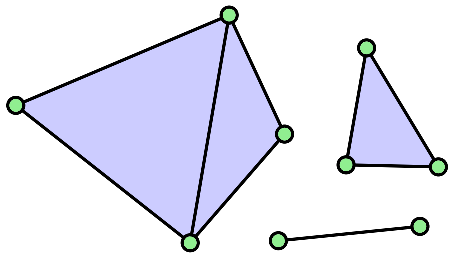
$\rightsquigarrow$ Compute the convex hull of n halfspaces in $\mathbb{R}^3$.

Incidences and combinatorial duality

Consider a set of vertices, edges and polygons.

A polygon P and an edge e are incident if e is on the boundary of P.

Same for edge/vertex and polygon/vertex incidences.

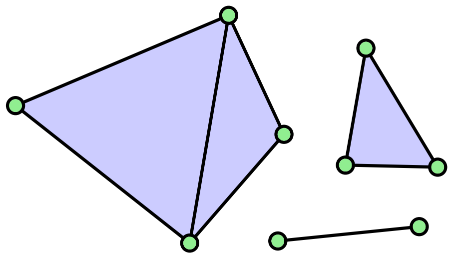


Incidences and combinatorial duality

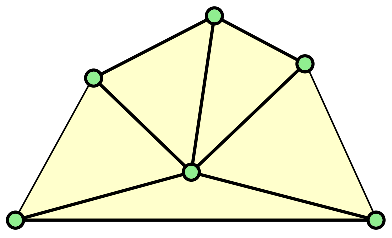
Consider a set of vertices, edges and polygons.

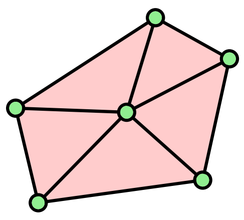
A polygon P and an edge e are incident if e is on the boundary of P.

Same for edge/vertex and polygon/vertex incidences.



Consider two families F_1 and F_2 of polygons/edges/vertices.



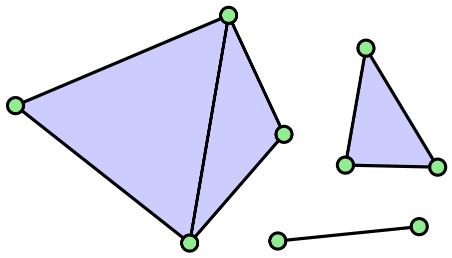


Incidences and combinatorial duality

Consider a set of vertices, edges and polygons.

A polygon P and an edge e are incident if e is on the boundary of P.

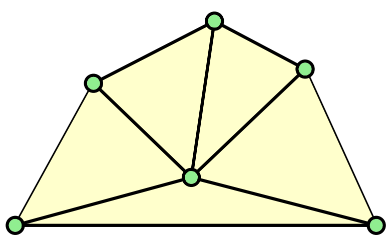
Same for edge/vertex and polygon/vertex incidences.

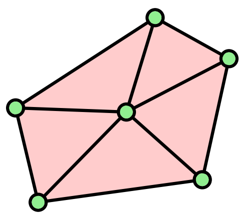


Consider two families F_1 and F_2 of polygons/edges/vertices.

F_1 and F_2 are combinatorially equivalent if there is a map $F_1 \to F_2$ that preserve incidences and...

maps a vertex to a vertex, an edge to an edge, a polygon to a polygon.



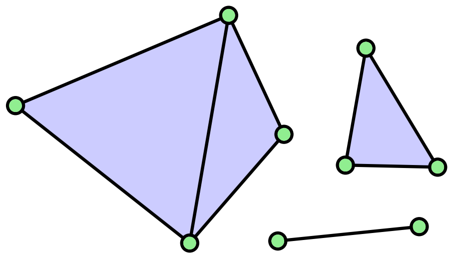


Incidences and combinatorial duality

Consider a set of vertices, edges and polygons.

A polygon P and an edge e are incident if e is on the boundary of P.

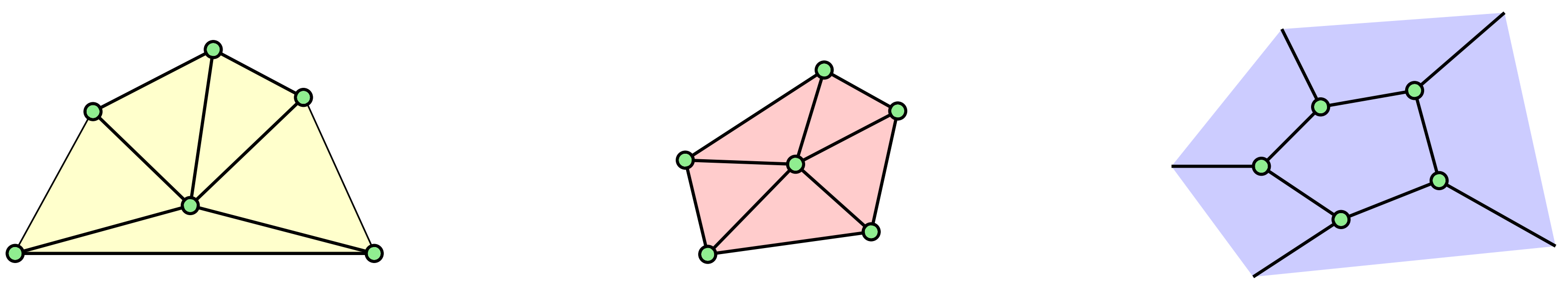
Same for edge/vertex and polygon/vertex incidences.



Consider two families F_1 and F_2 of polygons/edges/vertices.

F_1 and F_2 are combinatorially equivalent if there is a map $F_1 \to F_2$ that preserve incidences and...

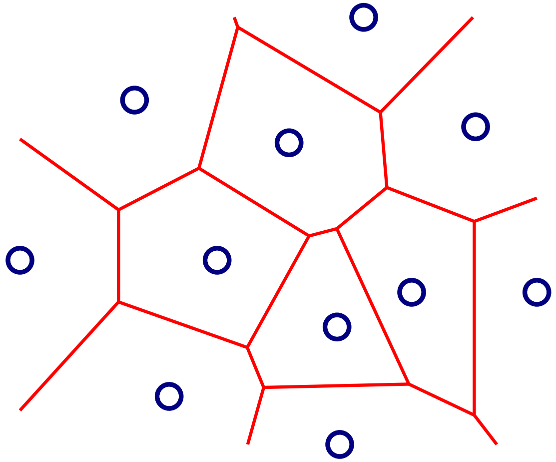
maps a vertex to a vertex, an edge to an edge, a polygon to a polygon.



F_1 and F_2 are combinatorially dual if there is a map $F_1 \to F_2$ that preserve incidences and...

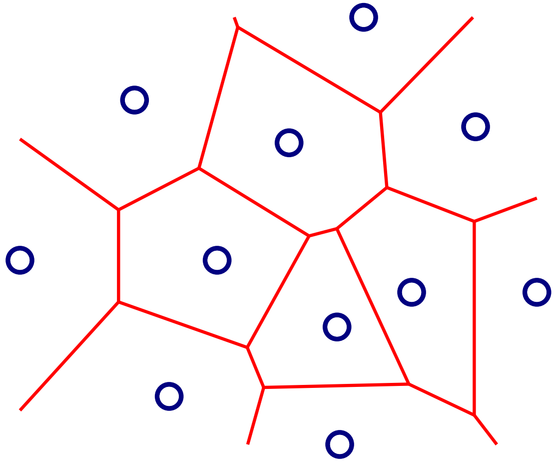
maps a vertex to a polygon, an edge to an edge, a polygon to a vertex.

Voronoi & Delaunay



There is a combinatorial duality between
the Voronoi diagram and the Delaunay triangulation of a point set.

Voronoi & Delaunay



There is a combinatorial duality between the Voronoi diagram and the Delaunay triangulation of a point set.

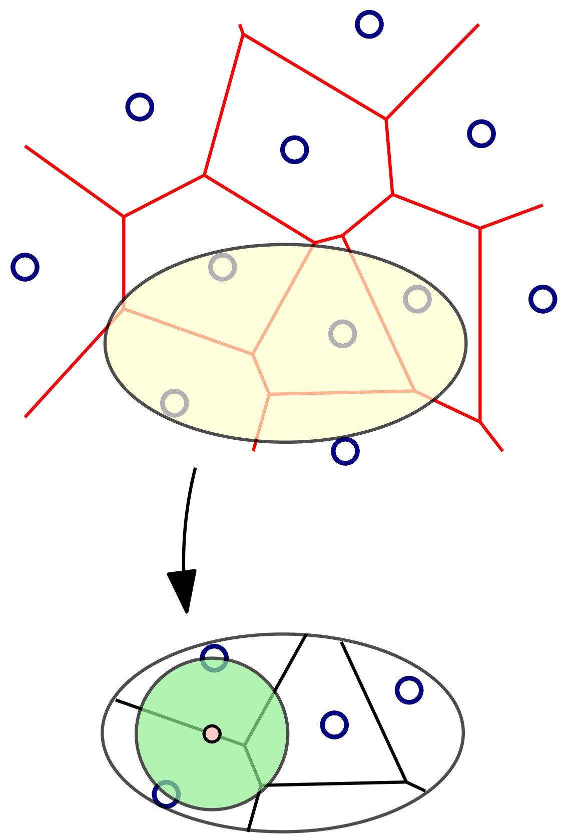
Voronoi regions

sites

Voronoi & Delaunay

There is a combinatorial duality between
the Voronoi diagram and the Delaunay triangulation of a point set.

A point on a Voronoi edge is at equal distance
from its two closest sites.



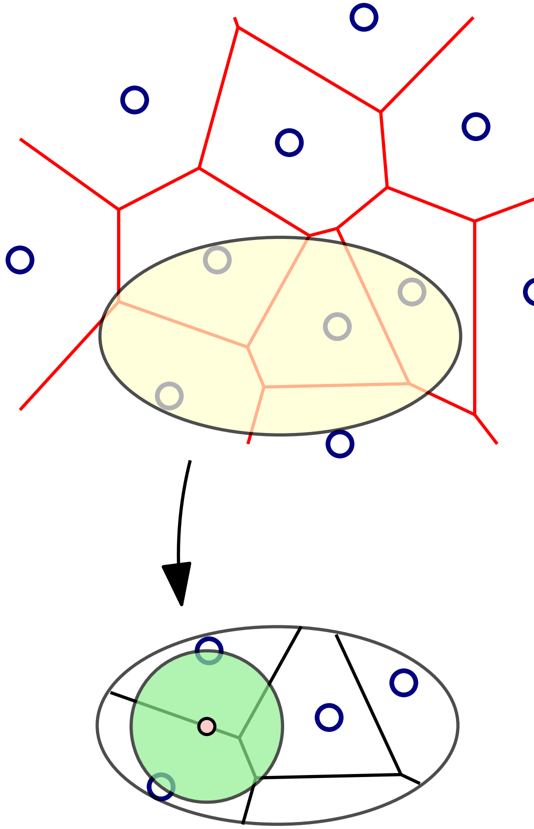
Voronoi regions

sites

Voronoi & Delaunay

There is a combinatorial duality between
the Voronoi diagram and the Delaunay triangulation of a point set.

A point on a Voronoi edge is at equal distance
from its two closest sites.



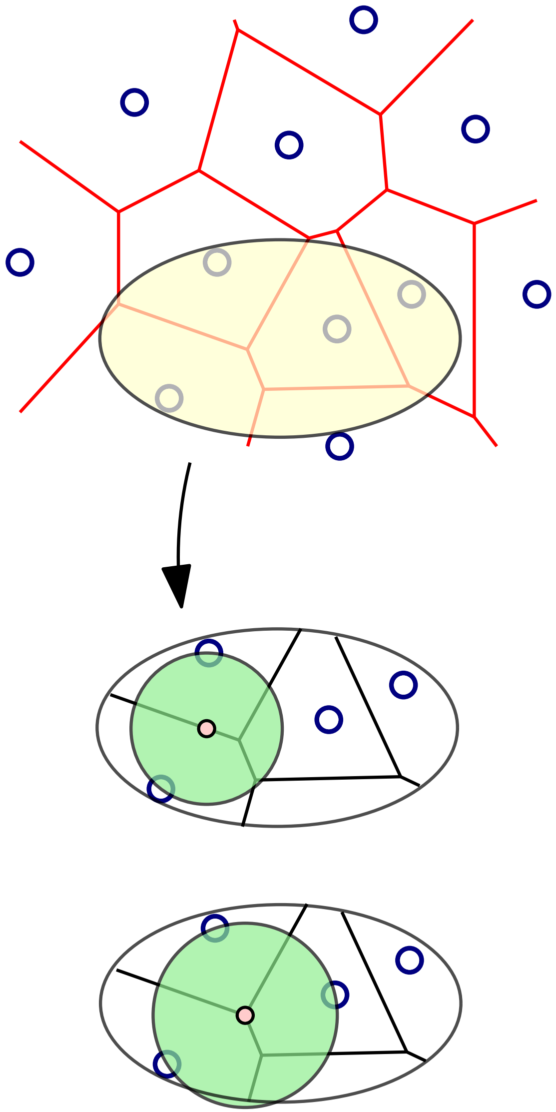
Voronoi regions	Voronoi edges
↕	↕
sites	Delaunay edges

Voronoi & Delaunay

There is a combinatorial duality between
the Voronoi diagram and the Delaunay triangulation of a point set.

A point on a Voronoi edge is at equal distance
from its two closest sites.

The Voronoi vertices are the points at equal distance
from their three closest sites.



Voronoi regions $\updownarrow$ sites

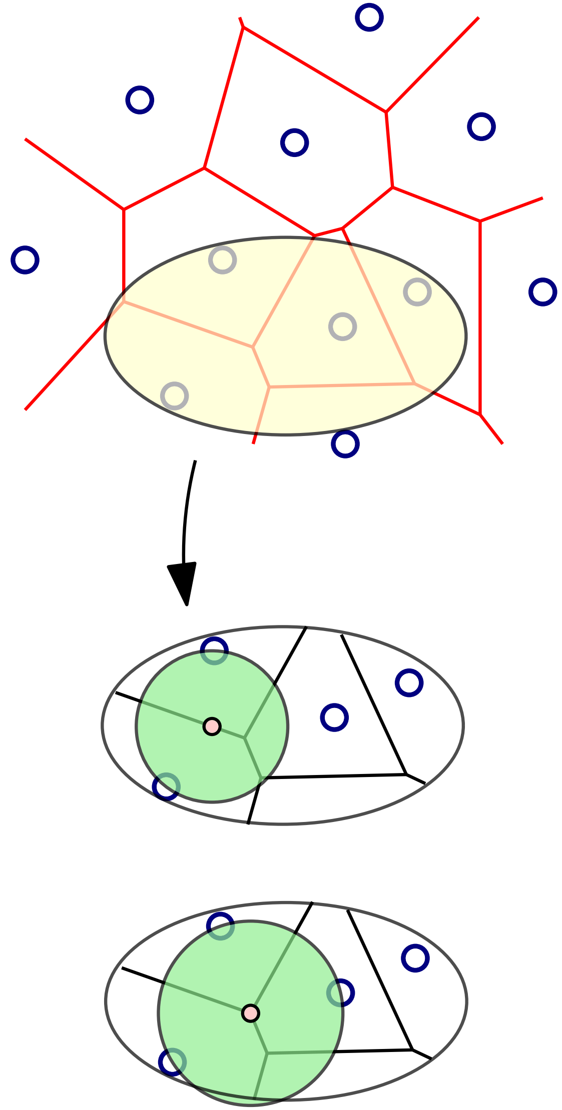
Voronoi edges $\updownarrow$ Delaunay edges

Voronoi & Delaunay

There is a combinatorial duality between
the Voronoi diagram and the Delaunay triangulation of a point set.

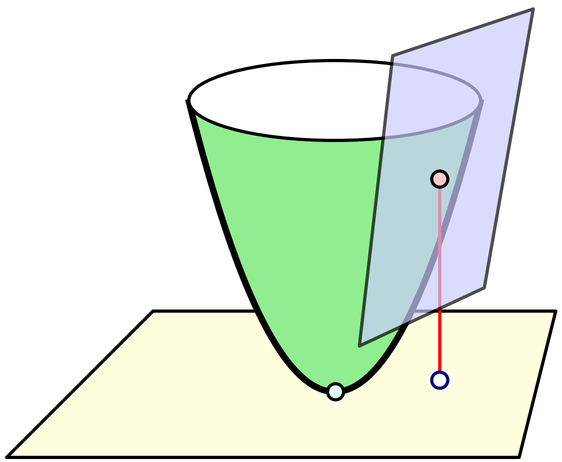
A point on a Voronoi edge is at equal distance
from its two closest sites.

The Voronoi vertices are the points at equal distance
from their three closest sites.



Voronoi regions	Voronoi edges	Voronoi vertices
↕	↕	↕
sites	Delaunay edges	(circumcenters of) Delaunay triangles

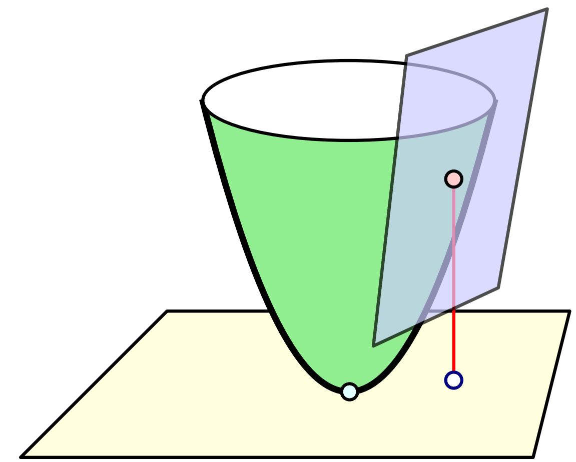
Lifting (bis)



The lift to the paraboloid maps every site p_i to

$$\begin{cases} \text{a point } p_i' \text{ on } (P), \\ \text{the plane } h(p_i) \text{ tangent to } (P) \text{ in } p_i. \end{cases}$$

Lifting (bis)



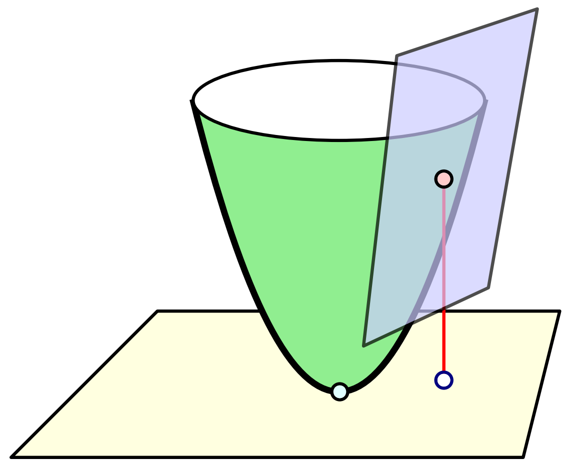
The lift to the paraboloid maps every site p_i to

$$\begin{cases} \text{a point } p_i' \text{ on } (P), \\ \text{the plane } h(p_i) \text{ tangent to } (P) \text{ in } p_i. \end{cases}$$

The Voronoi diagram of $p_1, \ldots, p_n$ is the projection of the convex hull of $h(p_1'), \ldots, h(p_n')$

The Delaunay triangulation of $p_1, \ldots, p_n$ is the projection of the convex hull of $p_1', \ldots, p_n'$

Lifting (bis)

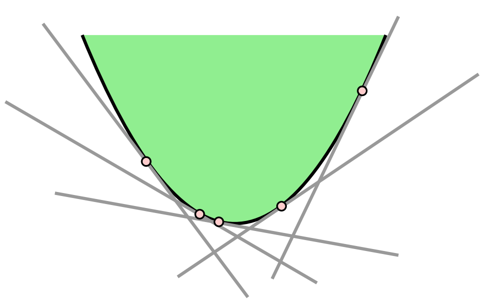


The lift to the paraboloid maps every site p_i to

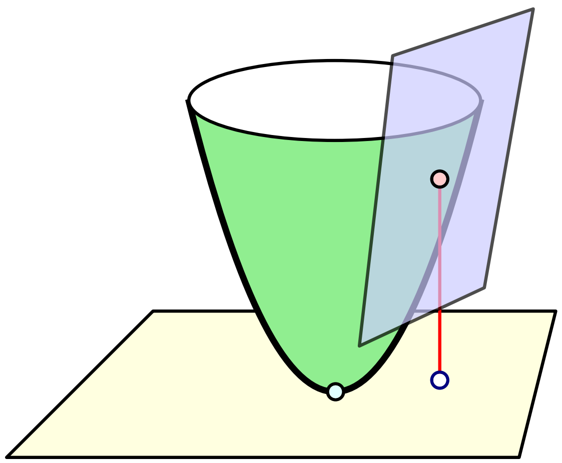
$$\begin{cases} \text{a point } p'_i \text{ on } (P), \\ \text{the plane } h(p_i) \text{ tangent to } (P) \text{ in } p_i. \end{cases}$$

The Voronoi diagram of $p_1, \ldots, p_n$ is the projection of the convex hull of $h(p'_1), \ldots, h(p'_n)$

The Delaunay triangulation of $p_1, \ldots, p_n$ is the projection of the convex hull of $p'_1, \ldots, p'_n$



Lifting (bis)

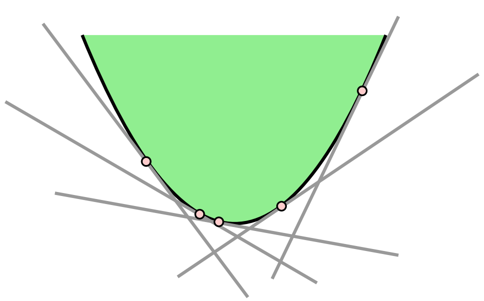


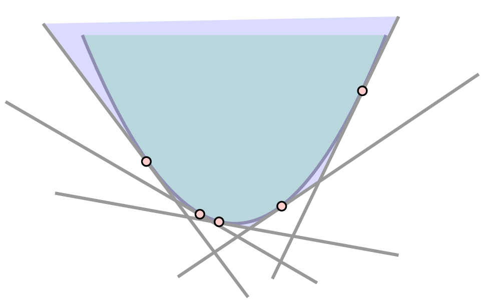
The lift to the paraboloid maps every site p_i to

$$\begin{cases} \text{a point } p'_i \text{ on } (P), \\ \text{the plane } h(p_i) \text{ tangent to } (P) \text{ in } p_i. \end{cases}$$

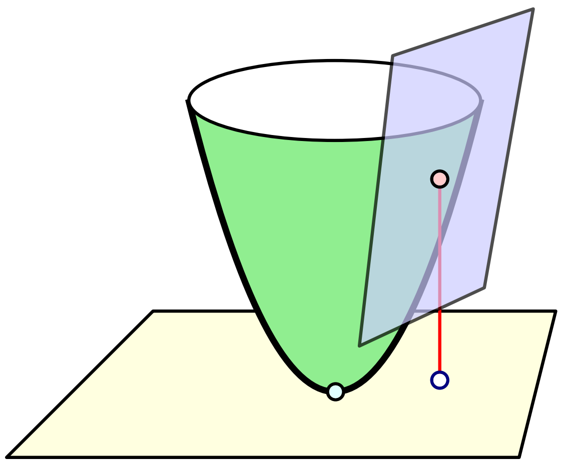
The Voronoi diagram of $p_1, \ldots, p_n$ is the projection of the convex hull of $h(p'_1), \ldots, h(p'_n)$

The Delaunay triangulation of $p_1, \ldots, p_n$ is the projection of the convex hull of $p'_1, \ldots, p'_n$





Lifting (bis)

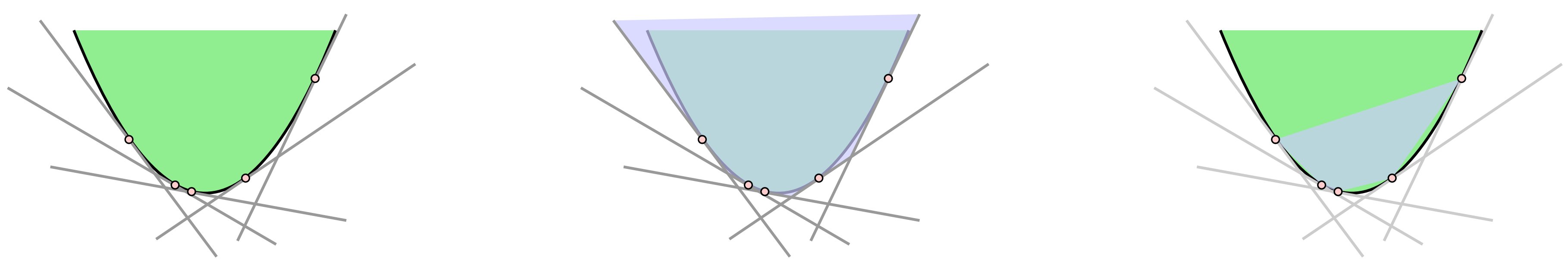


The lift to the paraboloid maps every site p_i to

$$\begin{cases} \text{a point } p'_i \text{ on } (P), \\ \text{the plane } h(p_i) \text{ tangent to } (P) \text{ in } p_i. \end{cases}$$

The Voronoi diagram of $p_1, \ldots, p_n$ is the projection of the convex hull of $h(p'_1), \ldots, h(p'_n)$

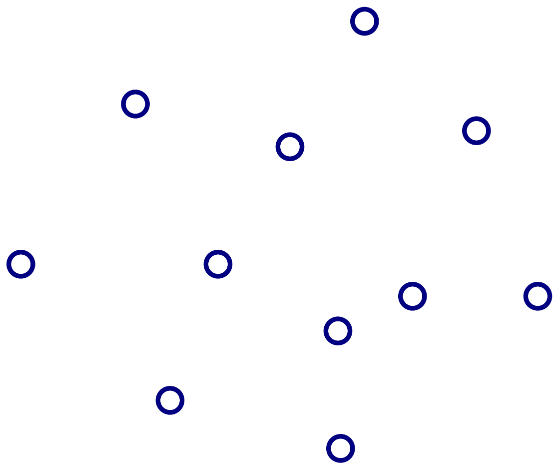
The Delaunay triangulation of $p_1, \ldots, p_n$ is the projection of the convex hull of $p'_1, \ldots, p'_n$



Natural neighbors interpolation

Assume we have a function f whose value is known in $p_1, \ldots, p_n$.

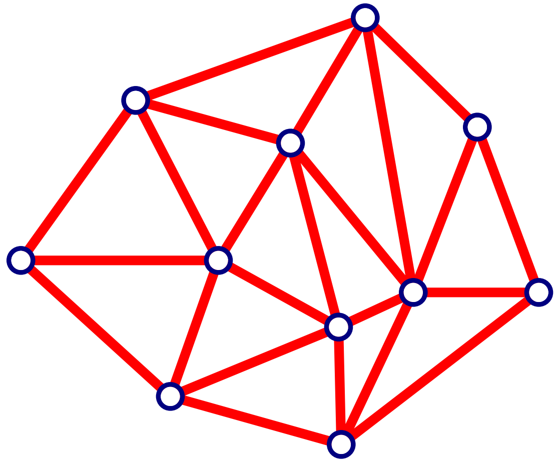
How can we interpolate f?



Natural neighbors interpolation

Assume we have a function f whose value is known in $p_1, \ldots, p_n$.

How can we interpolate f?



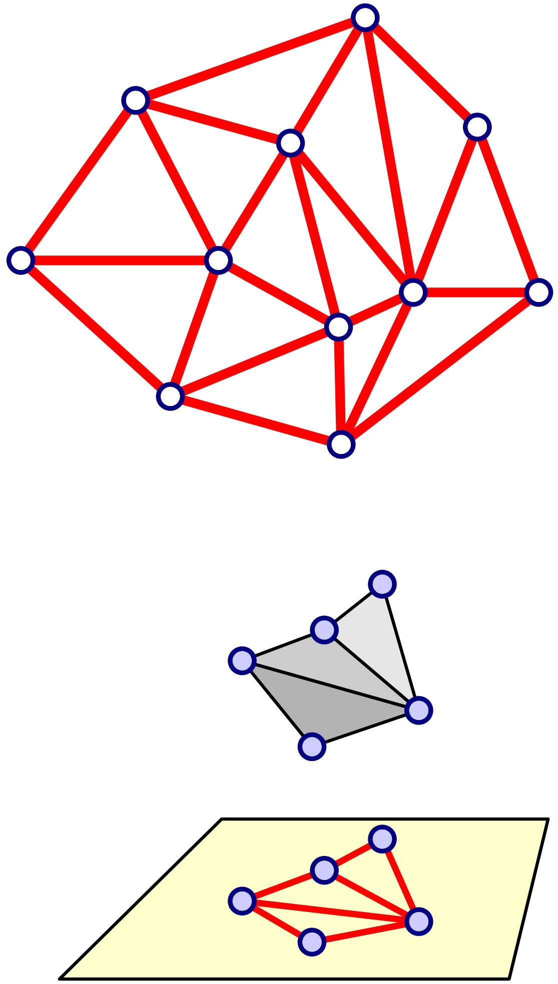
Solution 1: triangulate and interpolate linearly.

Natural neighbors interpolation

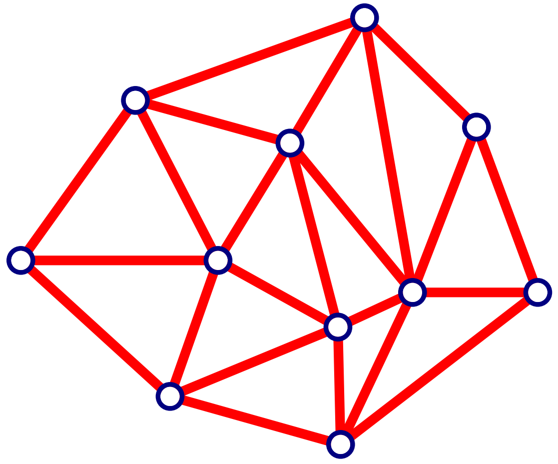
Assume we have a function f whose value is known in $p_1, \ldots, p_n$.

How can we interpolate f?

Solution 1: triangulate and interpolate linearly.



Natural neighbors interpolation

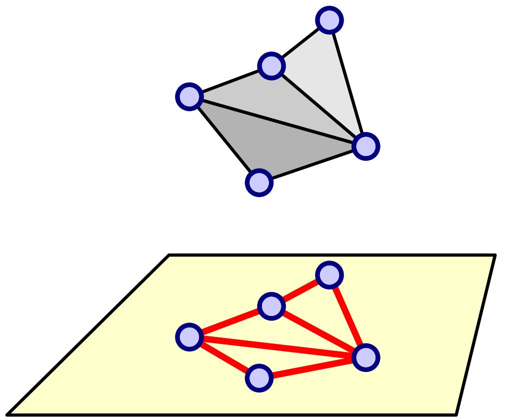


Assume we have a function f whose value is known in $p_1, \ldots, p_n$.

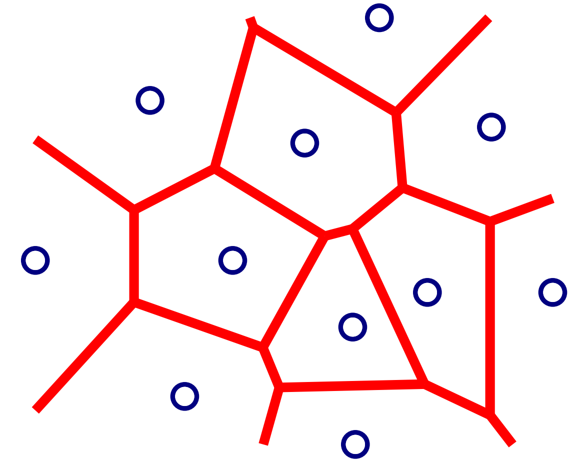
How can we interpolate f?

Solution 1: triangulate and interpolate linearly.

Pb: the interpolation $\tilde{f}$ is not smooth along the edges.



Natural neighbors interpolation

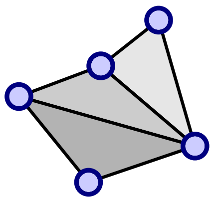


Assume we have a function f whose value is known in $p_1, \ldots, p_n$.

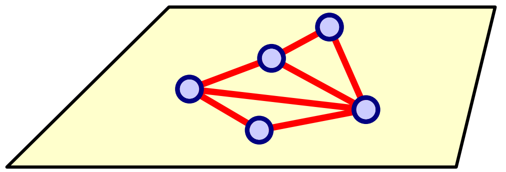
How can we interpolate f?

Solution 1: triangulate and interpolate linearly.

Pb: the interpolation $\tilde{f}$ is not smooth along the edges.



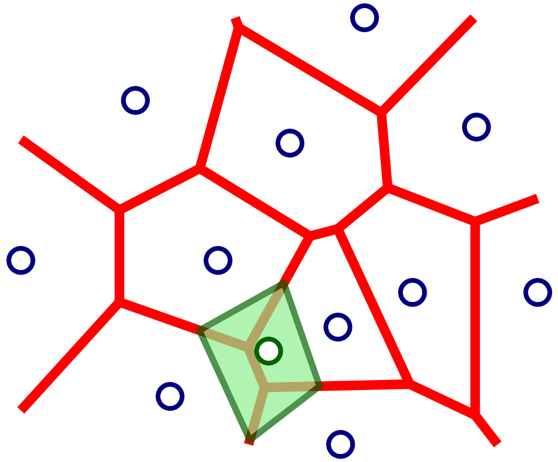
Solution 2: use the Voronoi diagram of the points



Natural neighbors interpolation

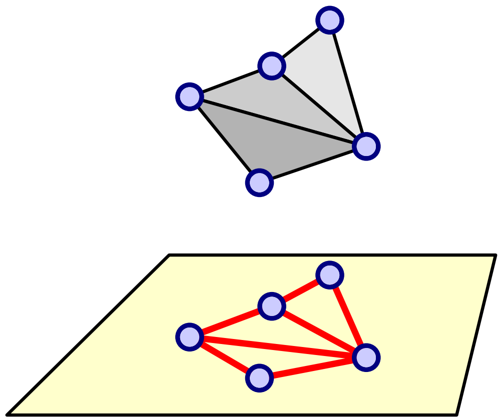
Assume we have a function f whose value is known in $p_1, \ldots, p_n$.

How can we interpolate f?



Solution 1: triangulate and interpolate linearly.

Pb: the interpolation $\tilde{f}$ is not smooth along the edges.



Solution 2: use the Voronoi diagram of the points

$$\tilde{f}(p) = \Sigma_{i=1}^{n} w_i f(p_i)$$

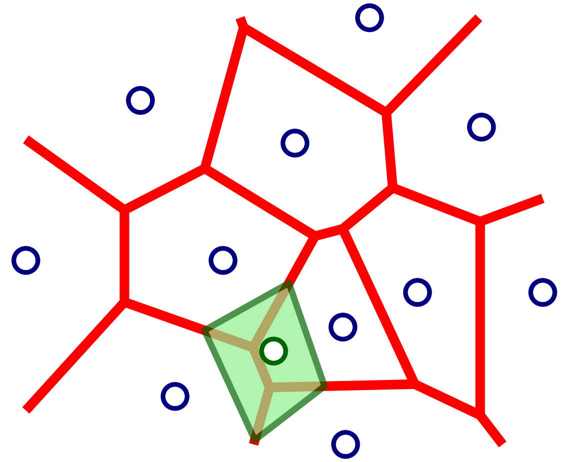
$$w_i = \frac{|R(p) \cap R_i|}{|R_i|}$$

$R(p) =$ Voronoi region of p in $\{p_1, \ldots, p_n\} \cup \{p\}$.

Natural neighbors interpolation

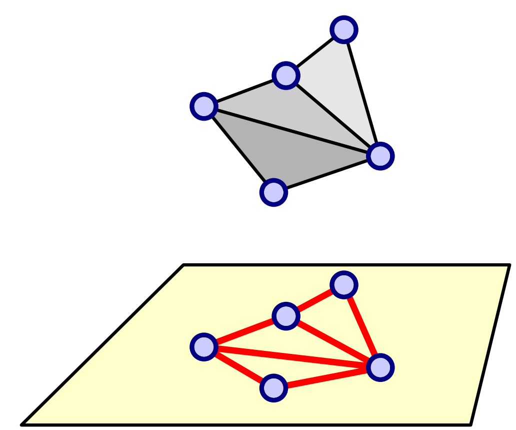
Assume we have a function f whose value is known in $p_1, \ldots, p_n$.

How can we interpolate f?



Solution 1: triangulate and interpolate linearly.

Pb: the interpolation $\tilde{f}$ is not smooth along the edges.



Solution 2: use the Voronoi diagram of the points

$$\tilde{f}(p) = \Sigma_{i=1}^{n} w_i f(p_i)$$

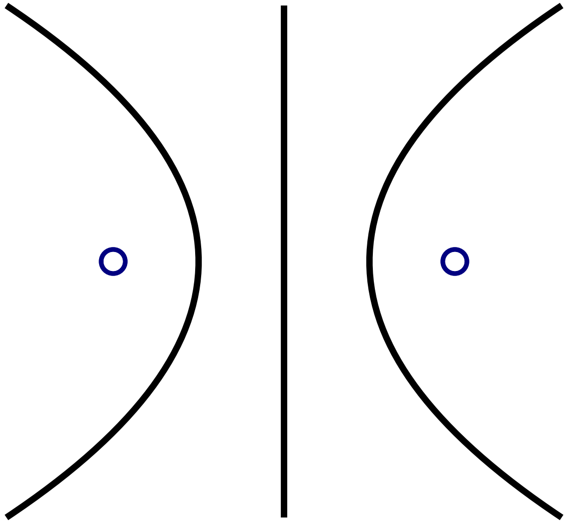
$$w_i = \frac{|R(p) \cap R_i|}{|R_i|}$$

$R(p) =$ Voronoi region of p in $\{p_1, \ldots, p_n\} \cup \{p\}$.

Now $\tilde{f}$ is smooth everywhere but in the points $p_1, \ldots p_n$.

Going a bit further

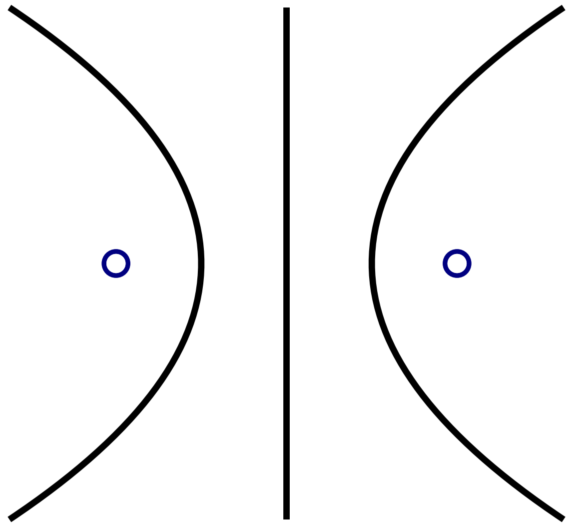
bisectors $\rightsquigarrow$ k-sector?



Going a bit further

bisectors $\rightsquigarrow$ k-sector?

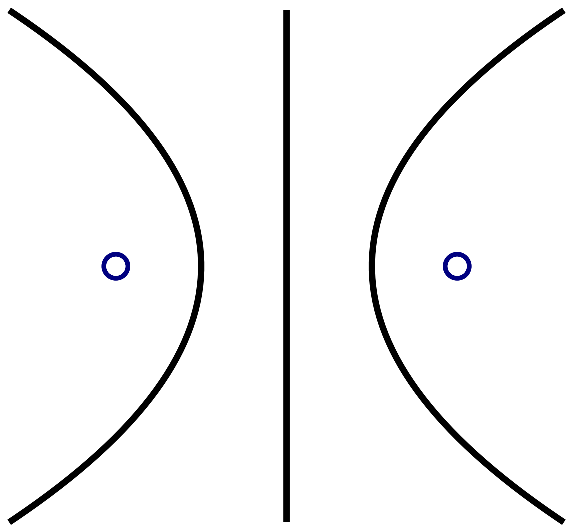
Existence? Uniqueness? Computation?

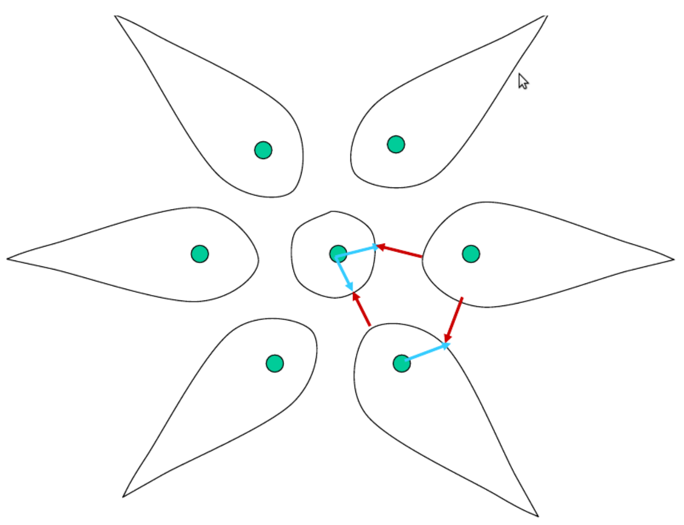


Going a bit further

bisectors $\rightsquigarrow$ k-sector?

Existence? Uniqueness? Computation?





Voronoi diagram with a neutral region.

Wrapping-up: Voronoi diagram

Natural structure: decomposition of space according to the closest site.

"Natural enough" that it was rediscovered over and over.

(combinatorially and geometrically) dual to Delaunay triangulation.

Combinatorial / geometric transforms help.

Question 5

Why are geometric algorithms hard to implement correctly?

First issue: degeneracies

Many algorithms are described assuming general position of the input.

$$\left.\begin{array}{l}\text{No two points have the same } x\text{-coordinate.}\\ \text{No three segments intersect in the same point.}\\ \text{No four points lie on the same circle.}\end{array}\right\}\quad \text{properties that hold generically.}$$

First issue: degeneracies

Many algorithms are described assuming general position of the input.

$$\left.\begin{array}{l}\text{No two points have the same } x\text{-coordinate.}\\ \text{No three segments intersect in the same point.}\\ \text{No four points lie on the same circle.}\end{array}\right\} \quad \text{properties that hold generically.}$$

A property that is true only for a subset of measure 0 of the space of inputs is a degeneracy.

First issue: degeneracies

Many algorithms are described assuming general position of the input.

$$\left.\begin{array}{l}\text{No two points have the same } x\text{-coordinate.}\\ \text{No three segments intersect in the same point.}\\ \text{No four points lie on the same circle.}\end{array}\right\}\quad \text{properties that hold generically.}$$

A property that is true only for a subset of measure 0 of the space of inputs is a degeneracy.

Degeneracy are common. They are often there by design.

Objects in contact are tangent.

Try asking an architect to avoid quadruple of coplanar points when designing a CAD model.

First issue: degeneracies

Many algorithms are described assuming general position of the input.

$$\left.\begin{array}{l}\text{No two points have the same } x\text{-coordinate.}\\ \text{No three segments intersect in the same point.}\\ \text{No four points lie on the same circle.}\end{array}\right\} \quad \text{properties that hold generically.}$$

A property that is true only for a subset of measure 0 of the space of inputs is a degeneracy.

Degeneracy are common. They are often there by design.

Objects in contact are tangent.

Try asking an architect to avoid quadruple of coplanar points when designing a CAD model.

Some degeneracies come from the problem, others from the algorithm.

First issue: degeneracies

Many algorithms are described assuming general position of the input.

$$\left.\begin{array}{l}\text{No two points have the same } x\text{-coordinate.}\\ \text{No three segments intersect in the same point.}\\ \text{No four points lie on the same circle.}\end{array}\right\}\quad \text{properties that hold generically.}$$

A property that is true only for a subset of measure 0 of the space of inputs is a degeneracy.

Degeneracy are common. They are often there by design.

Objects in contact are tangent.

Try asking an architect to avoid quadruple of coplanar points when designing a CAD model.

Some degeneracies come from the problem, others from the algorithm.

Can we handle degeneracies without treating each one separately?

Can we at least detect them efficiently?

Hardness of testing degeneracies

The 3-sum problem: Given n numbers, decide if three of them sum to 0.

What is the best algorithm you can come-up with?

Hardness of testing degeneracies

The 3-**sum problem**: Given n numbers, decide if three of them sum to 0.

What is the best algorithm you can come-up with?

Known bounds: $O(n^2)$ and $\Omega(n \log n)$.

Hardness of testing degeneracies

The 3-sum problem: Given n numbers, decide if three of them sum to 0.

What is the best algorithm you can come-up with?

Known bounds: $O(n^2)$ and $\Omega(n \log n)$.

15 years old conjecture: any algorithm solving 3-sum has $\Omega(n^2)$ time complexity.

Hardness of testing degeneracies

The 3**-sum problem**: Given n numbers, decide if three of them sum to 0.

What is the best algorithm you can come-up with?

Known bounds: $O(n^2)$ and $\Omega(n \log n)$.

15 years old conjecture: any algorithm solving 3-sum has $\Omega(n^2)$ time complexity.

If we can detect triples of aligned 2D points in $o(n^2)$ time then we can solve 3-sum in $o(n^2)$ time.

Hardness of testing degeneracies

The 3-sum problem: Given n numbers, decide if three of them sum to 0.

What is the best algorithm you can come-up with?

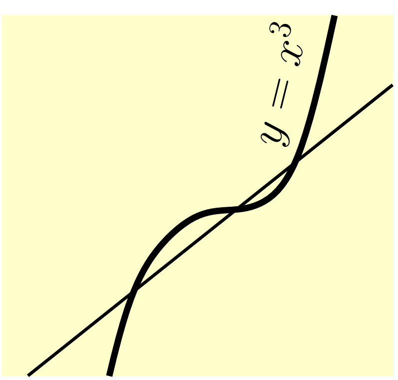
Known bounds: $O(n^2)$ and $\Omega(n \log n)$.

15 years old conjecture: any algorithm solving 3-sum has $\Omega(n^2)$ time complexity.

If we can detect triples of aligned 2D points in $o(n^2)$ time then we can solve 3-sum in $o(n^2)$ time.

numbers $t_1, \ldots, t_n \to$ points $p_1, \ldots, p_n$ with $p_i = (t_i, t_i^3)$.

$t_i + t_j + t_k = 0 \Leftrightarrow p_i, p_j, p_k$ are aligned.



p, q and r are aligned $\Leftrightarrow \begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix} = 0.$

$$\begin{vmatrix} t_i & t_j & t_k \\ t_i^3 & t_j^3 & t_k^3 \\ 1 & 1 & 1 \end{vmatrix} = (t_j - t_i)(t_k - t_i)(t_k - t_j)(t_i + t_j + t_k).$$

Hardness of testing degeneracies

The 3**-sum problem**: Given n numbers, decide if three of them sum to 0.

What is the best algorithm you can come-up with?

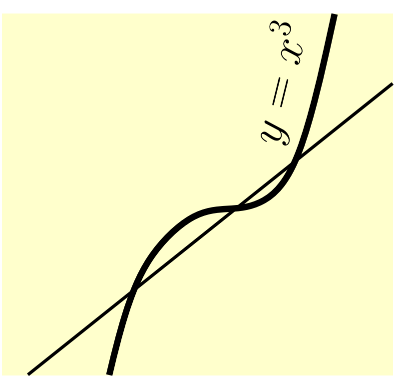
Known bounds: $O(n^2)$ and $\Omega(n \log n)$.

15 years old conjecture: any algorithm solving 3-sum has $\Omega(n^2)$ time complexity.

If we can detect triples of aligned 2D points in $o(n^2)$ time then we can solve 3-sum in $o(n^2)$ time.

numbers $t_1, \ldots, t_n \to$ points $p_1, \ldots, p_n$ with $p_i = (t_i, t_i^3)$.

$t_i + t_j + t_k = 0 \Leftrightarrow p_i, p_j, p_k$ are aligned.



p, q and r are aligned $\Leftrightarrow \begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix} = 0.$

$$\begin{vmatrix} t_i & t_j & t_k \\ t_i^3 & t_j^3 & t_k^3 \\ 1 & 1 & 1 \end{vmatrix} = (t_j - t_i)(t_k - t_i)(t_k - t_j)(t_i + t_j + t_k).$$

Testing if $d+1$ points lie on a common hyperplane in $\mathbb{R}^d$ is $\lceil \frac{d}{2} \rceil$-sum hard.

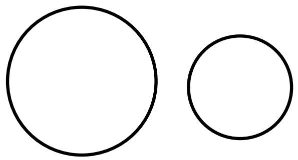
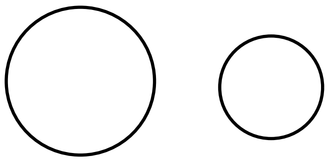
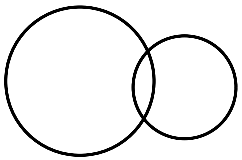
Perturbing? Easier said than done

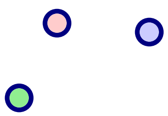
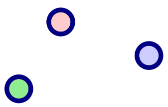
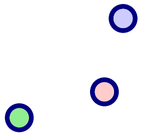
In principle, perturbing the points eliminate degeneracies.

Perturbing? Easier said than done

In principle, perturbing the points eliminate degeneracies.

First issue: the perturbation should preserves non-degenerate inputs.

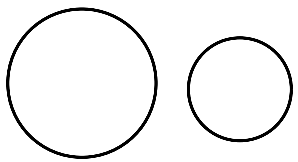
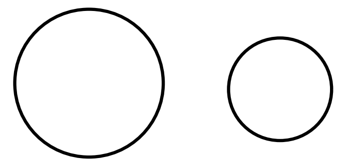
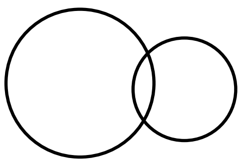
 can be perturbed in  but not in 

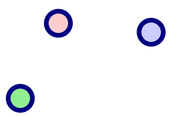
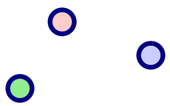
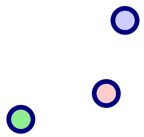
 can be perturbed in  but not in 

Perturbing? Easier said than done

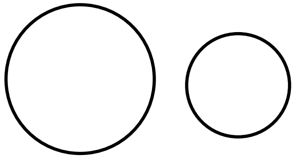
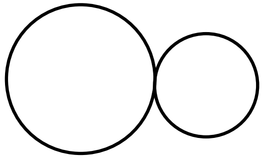
In principle, perturbing the points eliminate degeneracies.

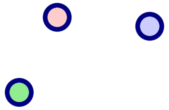
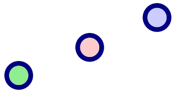
First issue: the perturbation should preserves non-degenerate inputs.

 can be perturbed in  but not in 

 can be perturbed in  but not in 

Second issue: the perturbation should not create new degeneracies.

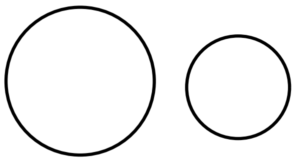
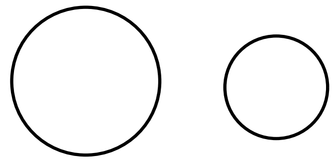
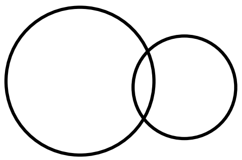
 should not be perturbed in 

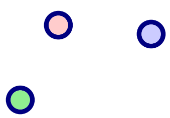
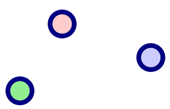
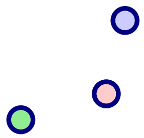
 should not be perturbed in 

Perturbing? Easier said than done

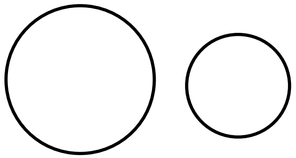
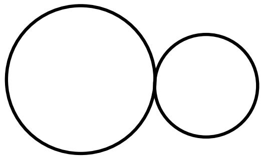
In principle, perturbing the points eliminate degeneracies.

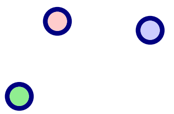
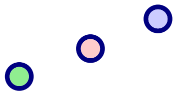
First issue: the perturbation should preserves non-degenerate inputs.

 can be perturbed in  but not in 

 can be perturbed in  but not in 

Second issue: the perturbation should not create new degeneracies.

 should not be perturbed in 

 should not be perturbed in 

Bottom line: "Epsilon=10^-12" is not an option if we want any kind of guarantee.

Systematic approach: simulation of simplicity

Degeneracy correspond to vanishing of some of the polynomials evaluating the geometric predicates.

Systematic approach: simulation of simplicity

Degeneracy correspond to vanishing of some of the polynomials evaluating the geometric predicates.

Consider a geometric object as a function of one variable t [1990].
The input we are interested in is the value when $t = 0$.
Ex: the point $p = (3, 12)$ becomes $p = (3 + t, 12 + t^2)$.

Make all computations for "$t > 0$ sufficiently small" then take $\lim_{t \to 0}$.

Systematic approach: simulation of simplicity

Degeneracy correspond to vanishing of some of the polynomials evaluating the geometric predicates.

Consider a geometric object as a function of one variable t [1990].
The input we are interested in is the value when $t = 0$.

Ex: the point $p = (3, 12)$ becomes $p = (3 + t, 12 + t^2)$.

Make all computations for "$t > 0$ sufficiently small" then take $\lim_{t \to 0}$.

Choose the functions so that the relevant polynomials do not identically vanish.
Example: convex hull computation, point-in-polygon.

Predicates are x-*coordinates comparison* and *orientation*.

Replacing p_i by $p_i + (t^{2^{2i}}, t^{2^{2i+1}})$ handles all degeneracies for these predicates.

Systematic approach: simulation of simplicity

Degeneracy correspond to vanishing of some of the polynomials evaluating the geometric predicates.

Consider a geometric object as a function of one variable t [1990].
The input we are interested in is the value when $t = 0$.

Ex: the point $p = (3, 12)$ becomes $p = (3 + t, 12 + t^2)$.

Make all computations for "$t > 0$ sufficiently small" then take $\lim_{t \to 0}$.

Choose the functions so that the relevant polynomials do not identically vanish.
Example: convex hull computation, point-in-polygon.

Predicates are x-*coordinates comparison* and *orientation*.

Replacing p_i by $p_i + (t^{2^{2i}}, t^{2^{2i+1}})$ handles all degeneracies for these predicates.

Heavy machinery, important slow-down, ignore voluntary degeneracies.

Systematic approach: simulation of simplicity

Degeneracy correspond to vanishing of some of the polynomials evaluating the geometric predicates.

Consider a geometric object as a function of one variable t [1990].
The input we are interested in is the value when $t = 0$.

Ex: the point $p = (3, 12)$ becomes $p = (3 + t, 12 + t^2)$.

Make all computations for "$t > 0$ sufficiently small" then take $\lim_{t\to 0}$.

Choose the functions so that the relevant polynomials do not identically vanish.
Example: convex hull computation, point-in-polygon.

Predicates are x*-coordinates comparison* and *orientation*.

Replacing p_i by $p_i + (t^{2^{2i}}, t^{2^{2i+1}})$ handles all degeneracies for these predicates.

Heavy machinery, important slow-down, ignore voluntary degeneracies.

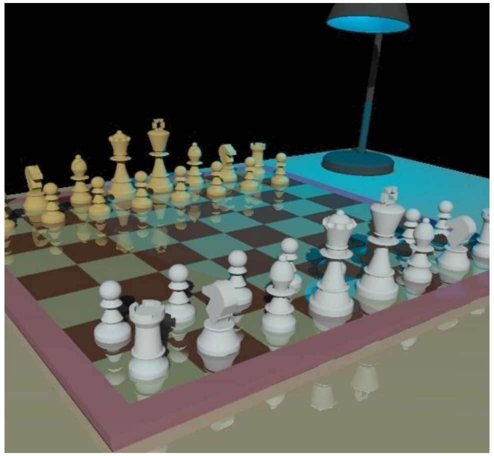
Partial perturbation: shearing $(x, y) \mapsto (x + ty, y)$

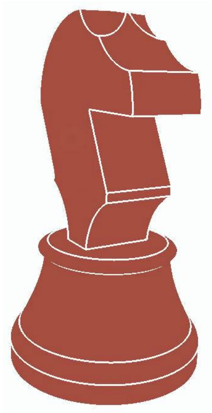
The hard way: zillions of cases

The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

Example: intersection of two quadric surfaces in $\mathbb{R}^3$

Fundamental problem in solid modelling.



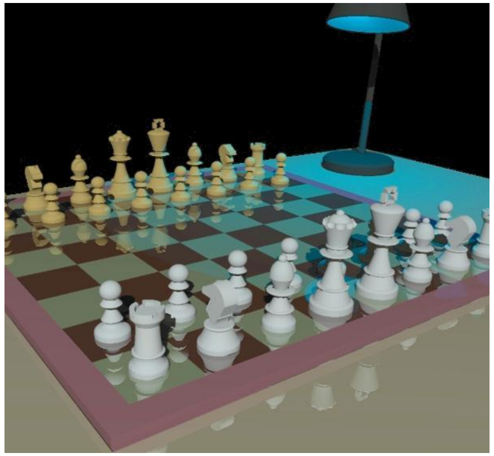


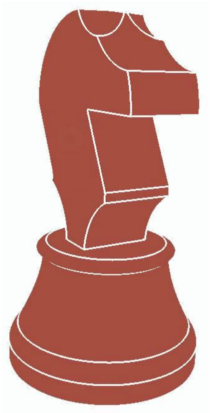
The hard way: zillions of cases

The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

Example: intersection of two quadric surfaces in $\mathbb{R}^3$

Fundamental problem in solid modelling.





Many degenerate cases...

A published algorithm missed several.

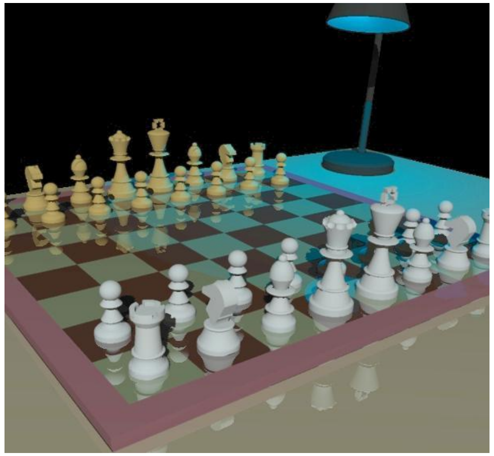
Classification is non-trivial

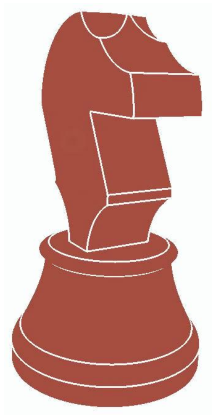
The hard way: zillions of cases

The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

Example: intersection of two quadric surfaces in $\mathbb{R}^3$

Fundamental problem in solid modelling.

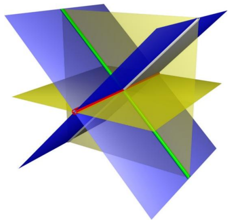




Many degenerate cases...

A published algorithm missed several.

Classification is non-trivial

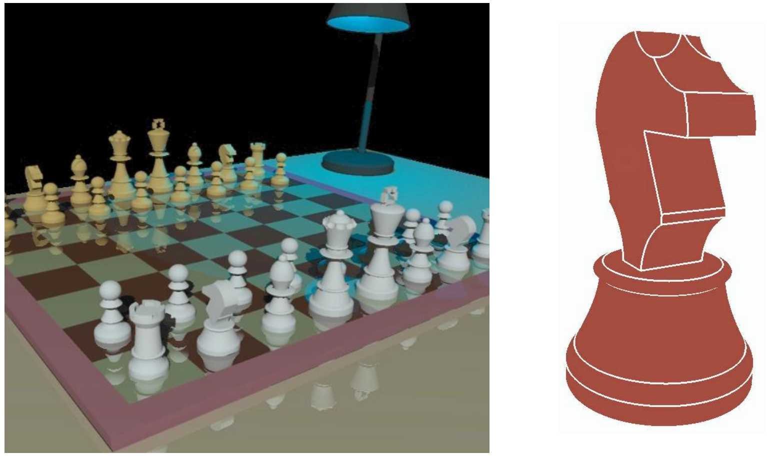


The hard way: zillions of cases

The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

Example: intersection of two quadric surfaces in $\mathbb{R}^3$

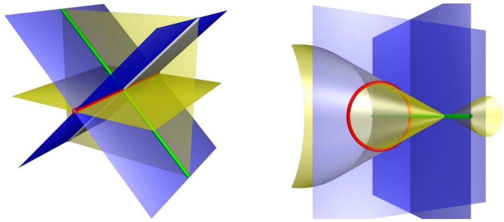
Fundamental problem in solid modelling.



Many degenerate cases...

A published algorithm missed several.

Classification is non-trivial

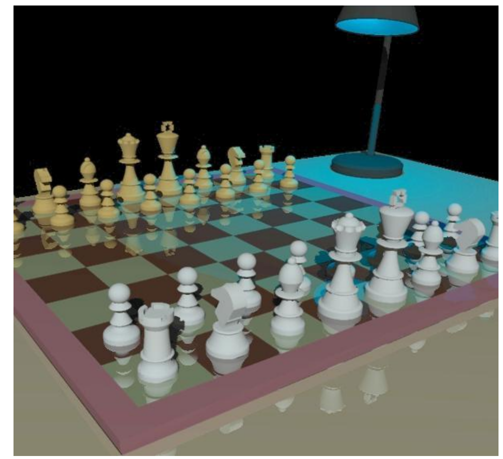


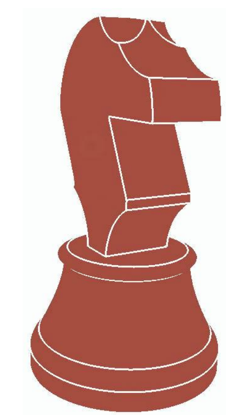
The hard way: zillions of cases

The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

Example: intersection of two quadric surfaces in $\mathbb{R}^3$

Fundamental problem in solid modelling.

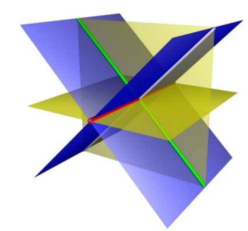


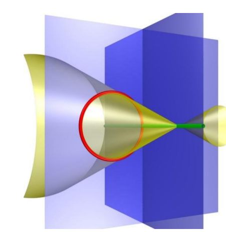


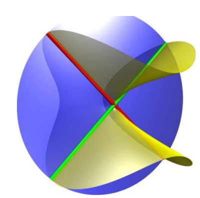
Many degenerate cases...

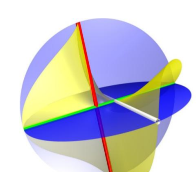
A published algorithm missed several.

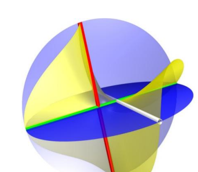
Classification is non-trivial









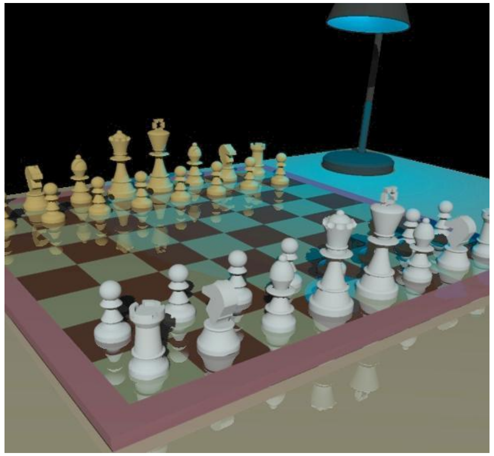


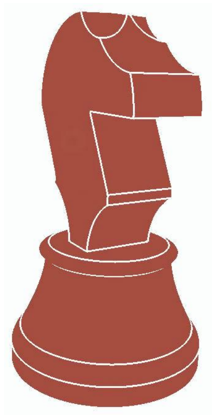
The hard way: zillions of cases

The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

Example: intersection of two quadric surfaces in $\mathbb{R}^3$

Fundamental problem in solid modelling.

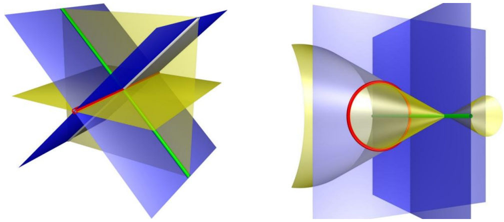


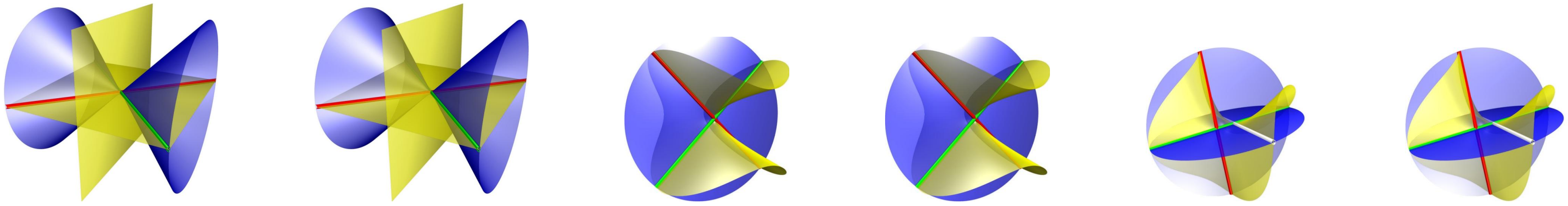


Many degenerate cases...

A published algorithm missed several.

Classification is non-trivial



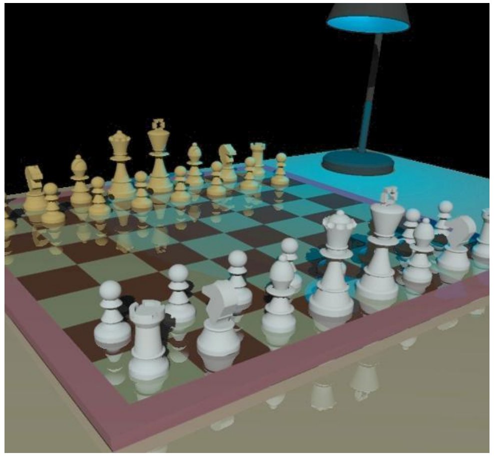


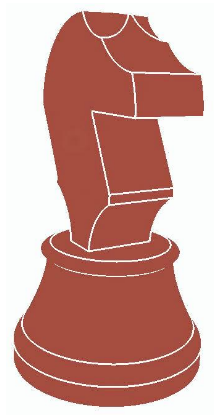
The hard way: zillions of cases

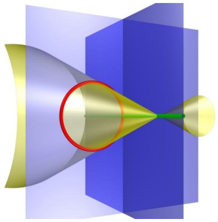
The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

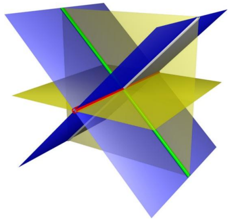
Example: intersection of two quadric surfaces in $\mathbb{R}^3$

Fundamental problem in solid modelling.





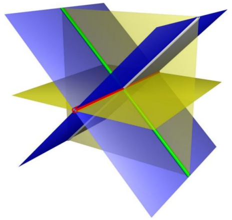


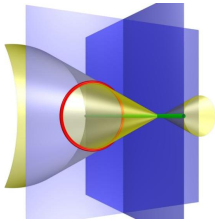


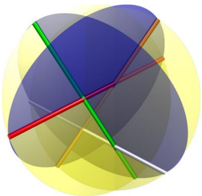
Many degenerate cases...

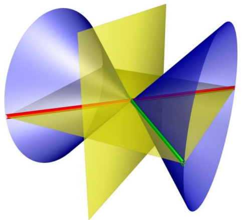
A published algorithm missed several.

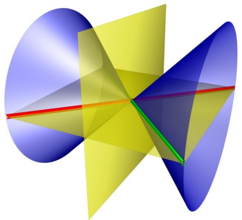
Classification is non-trivial

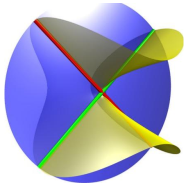


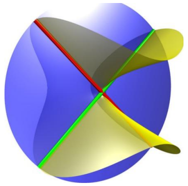


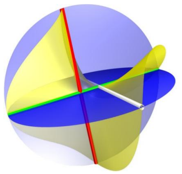


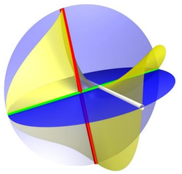




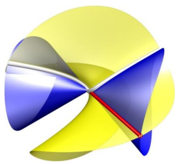






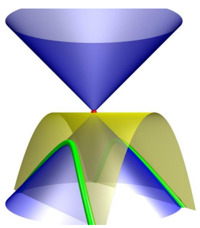


The hard way: zillions of cases

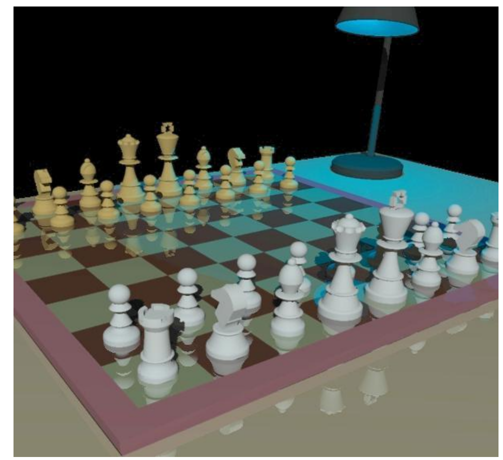


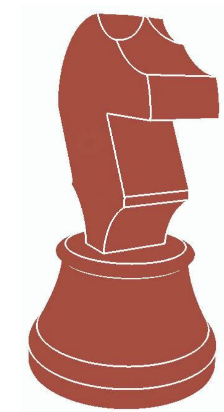
The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

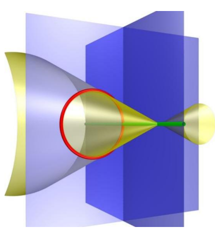
Example: intersection of two quadric surfaces in $\mathbb{R}^3$

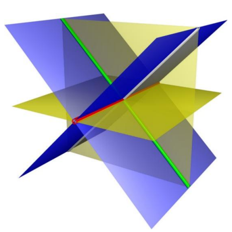


Fundamental problem in solid modelling.





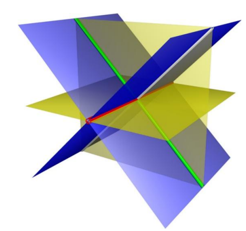


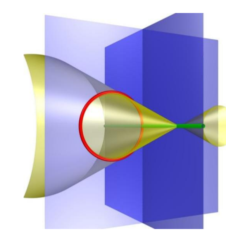


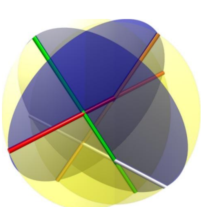
Many degenerate cases...

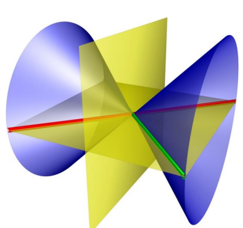
A published algorithm missed several.

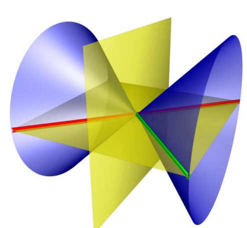
Classification is non-trivial

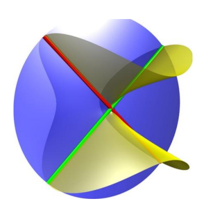


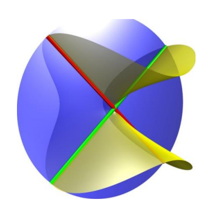


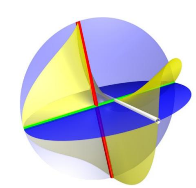


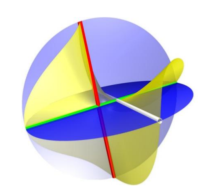




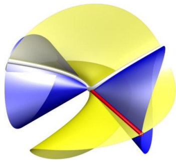






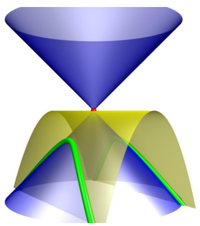


The hard way: zillions of cases

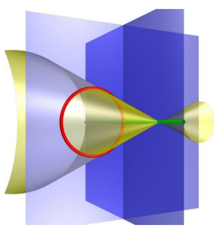


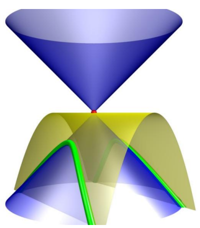
The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

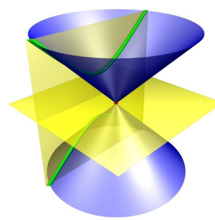
Example: intersection of two quadric surfaces in $\mathbb{R}^3$

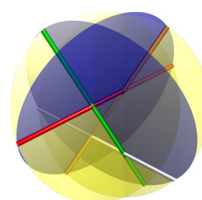


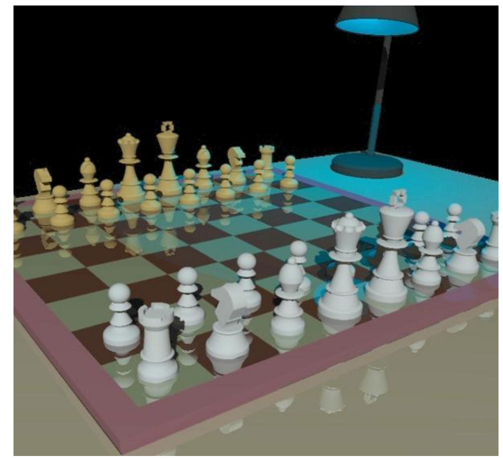
Fundamental problem in solid modelling.

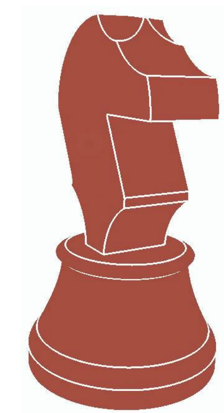


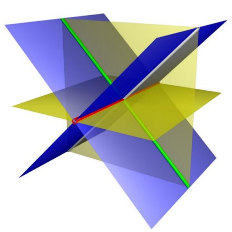








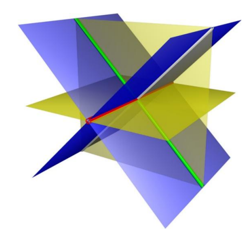


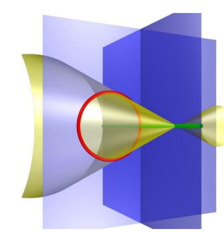


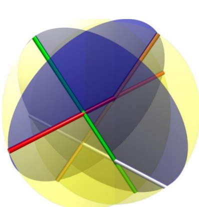
Many degenerate cases...

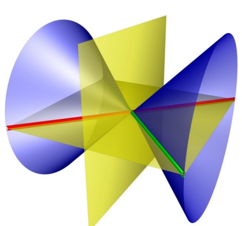
A published algorithm missed several.

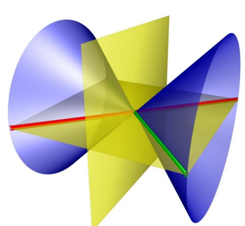
Classification is non-trivial

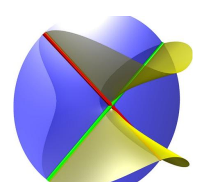


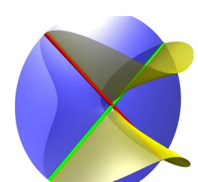


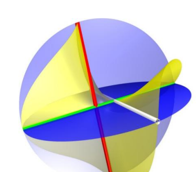


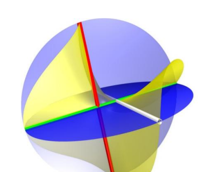




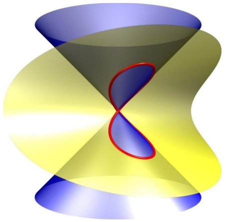


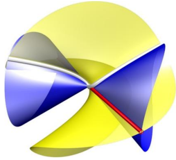






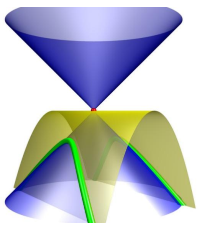
The hard way: zillions of cases



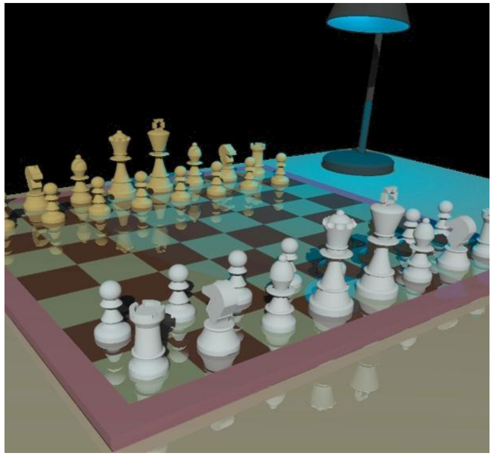


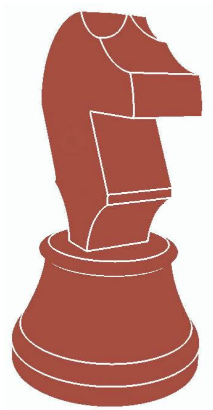
The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

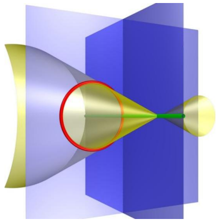
Example: intersection of two quadric surfaces in $\mathbb{R}^3$

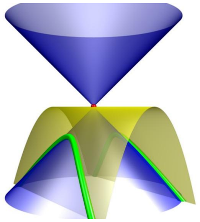


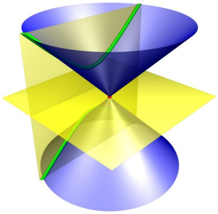
Fundamental problem in solid modelling.

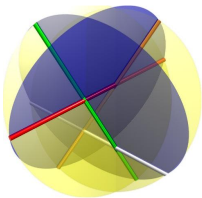


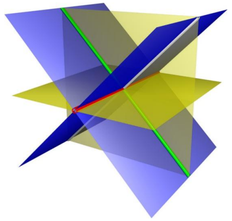








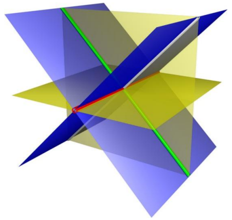


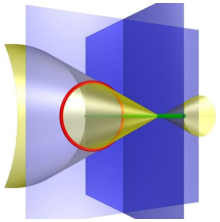


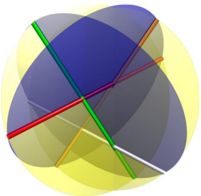
Many degenerate cases...

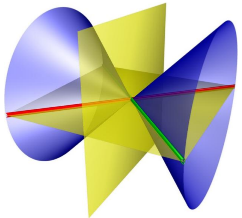
A published algorithm missed several.

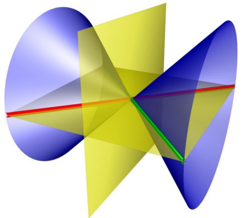
Classification is non-trivial

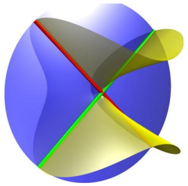


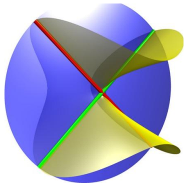


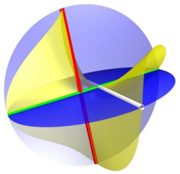


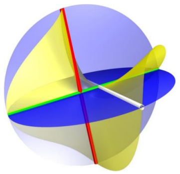




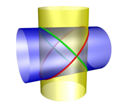
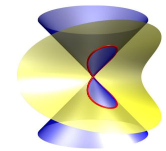
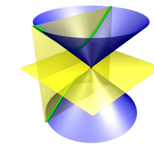
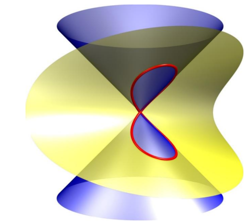


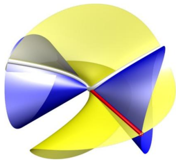






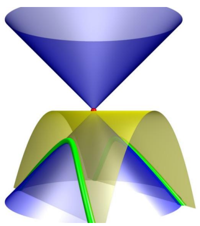
The hard way: zillions of cases

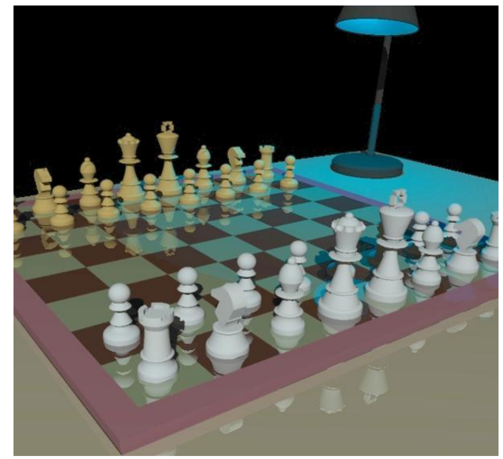
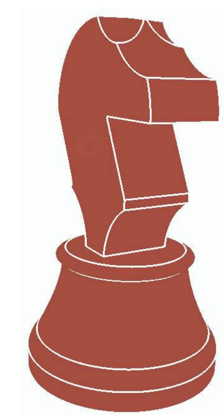


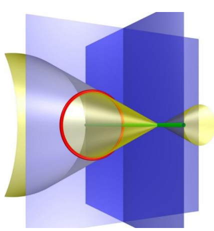
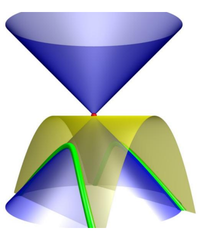
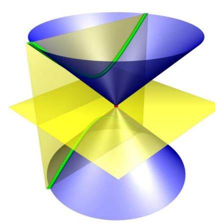
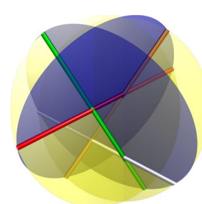
The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

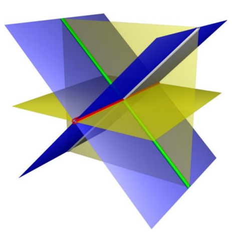
Example: intersection of two quadric surfaces in $\mathbb{R}^3$



Fundamental problem in solid modelling.

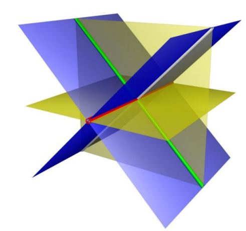
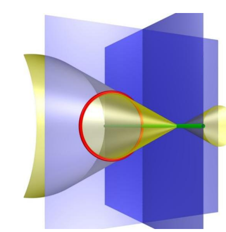
   

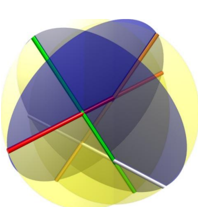
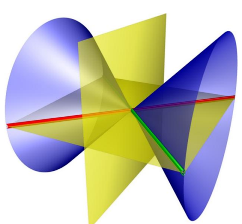
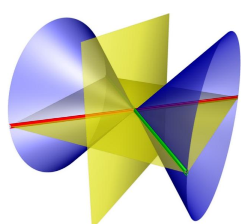
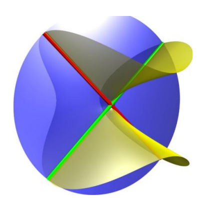
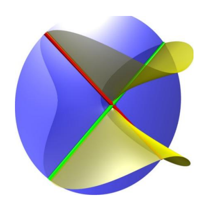
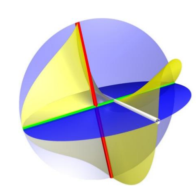
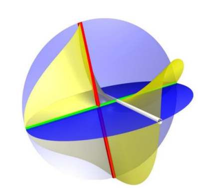


Many degenerate cases...

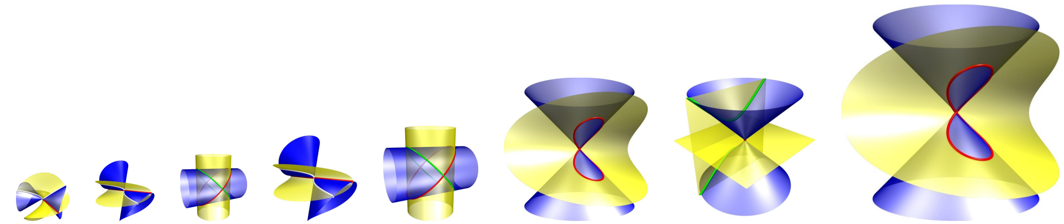
A published algorithm missed several.

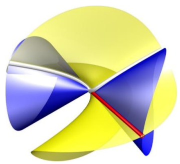
Classification is non-trivial

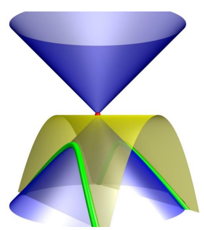
The hard way: zillions of cases



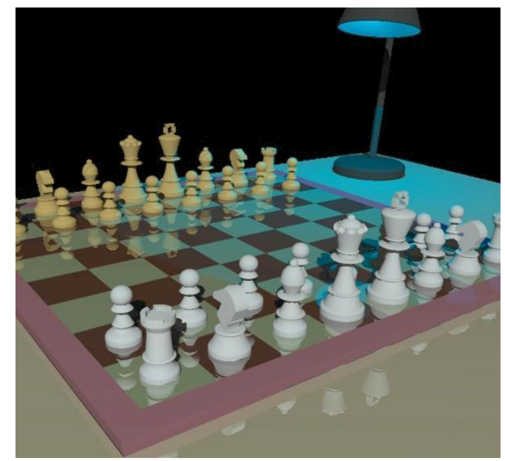


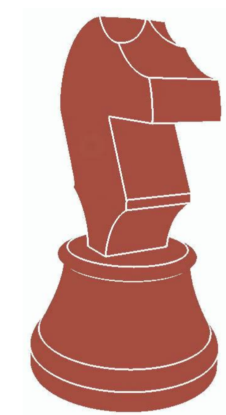
The problem is to find a way to tabulate all cases in a way where exhaustivity can be proven.

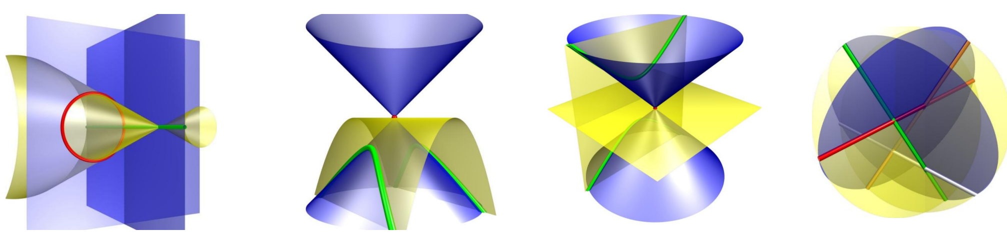
Example: intersection of two quadric surfaces in $\mathbb{R}^3$

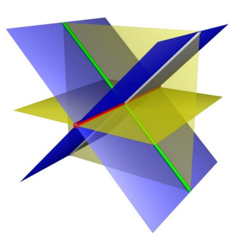


Fundamental problem in solid modelling.





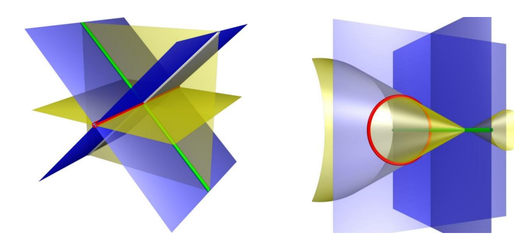


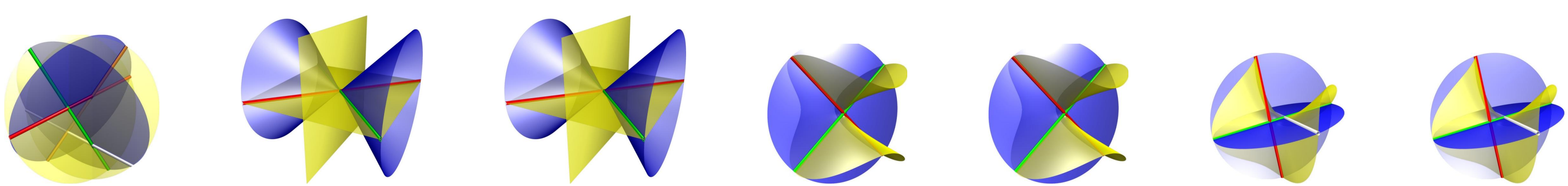


Many degenerate cases...

A published algorithm missed several.

Classification is non-trivial





Case analysis of quadric intersection

What are the possibly intersection curves of two quadric surfaces in $\mathbb{RP}^3$, up to projective transforms?

Case analysis of quadric intersection

What are the possibly intersection curves of two quadric surfaces in $\mathbb{RP}^3$, up to projective transforms?

Transformations that preserve the intersection type

- Replacing a quadric by a linear combinations of the equations of the two quadrics.
- Projective transform of the whole space.

Case analysis of quadric intersection

What are the possibly intersection curves of two quadric surfaces in $\mathbb{RP}^3$, up to projective transforms?

Transformations that preserve the intersection type

Replacing a quadric by a linear combinations of the equations of the two quadrics.

Projective transform of the whole space.

Encoding of the orbits under these transformations

Work on pencils of quadrics.

First encoding with Segre's characteristic (discriminates between intersection types in $\mathbb{CP}^3$).

Characteristic polynomial of a pencil.

One quadric Q in $\mathbb{R}^3 \to$ symmetric 4×4 matrix M_Q.

Pencil of Q and $R \to P(\lambda, \mu) = \det(\lambda M_Q + \mu M_R)$.

Number of roots of P, their multiplicity, inertia of their associated matrix...

Case analysis of quadric intersection

What are the possibly intersection curves of two quadric surfaces in $\mathbb{RP}^3$, up to projective transforms?

Transformations that preserve the intersection type

Replacing a quadric by a linear combinations of the equations of the two quadrics.

Projective transform of the whole space.

Encoding of the orbits under these transformations

Work on pencils of quadrics.

First encoding with Segre's characteristic (discriminates between intersection types in $\mathbb{CP}^3$).

Characteristic polynomial of a pencil.

One quadric Q in $\mathbb{R}^3 \to$ symmetric 4×4 matrix M_Q.

Pencil of Q and $R \to P(\lambda, \mu) = \det(\lambda M_Q + \mu M_R)$.

Number of roots of P, their multiplicity, inertia of their associated matrix...

Result: Tabulation with over 40 cases, 26 intersection types in total, proof of exhaustivity.

Second issue: numerical rounding

The arithmetic on a computer uses bounded precision (32 bits, 64 bits, IEEE float norms, etc...).

Small errors will be made in computations.

Not to mention that processors make errors from time to time...

Second issue: numerical rounding

The arithmetic on a computer uses bounded precision (32 bits, 64 bits, IEEE float norms, etc...).

Small errors will be made in computations.

Not to mention that processors make errors from time to time...

The question is: **can these error have a significant impact?**

Second issue: numerical rounding

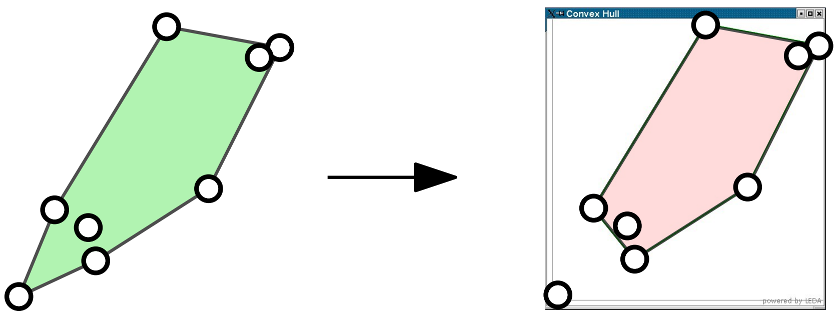
The arithmetic on a computer uses bounded precision (32 bits, 64 bits, IEEE float norms, etc...).

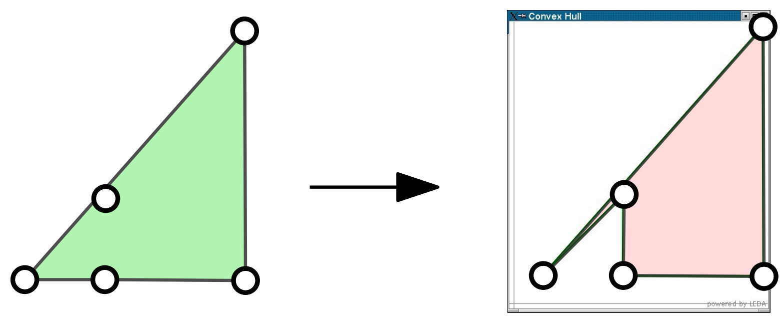
Small errors will be made in computations.

Not to mention that processors make errors from time to time...

The question is: **can these error have a significant impact?**

"Judge for yourself": the example of 2D convex hull computation.





Second issue: numerical rounding

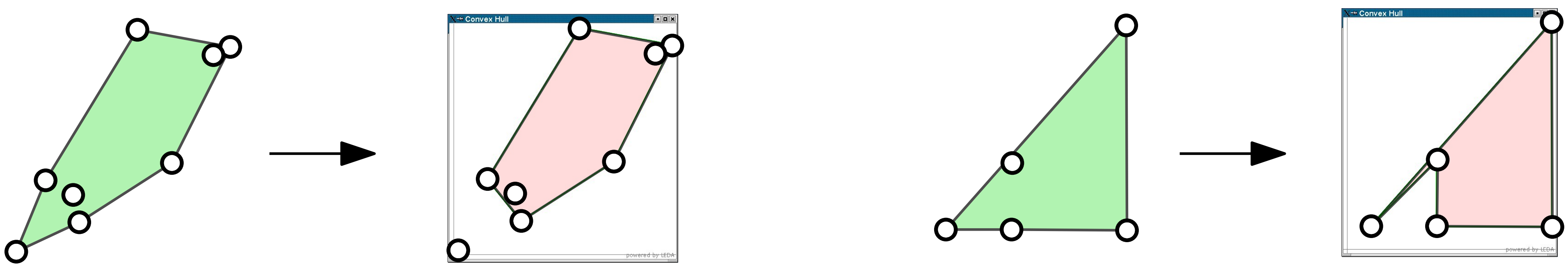
The arithmetic on a computer uses bounded precision (32 bits, 64 bits, IEEE float norms, etc...).

Small errors will be made in computations.

Not to mention that processors make errors from time to time...

The question is: **can these error have a significant impact?**

"Judge for yourself": the example of 2D convex hull computation.

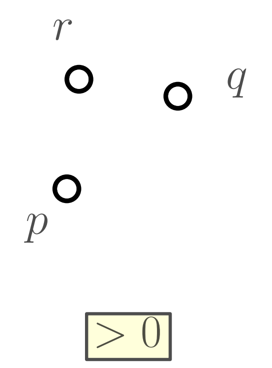


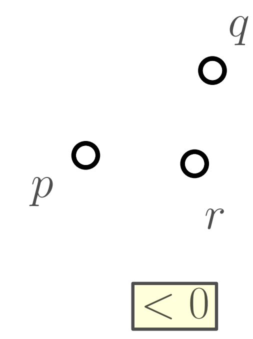
The problem: three points are nearly aligned, and the orientation predicates make inconsistent errors.

"Sometimes left, sometimes right".

A close look at that example

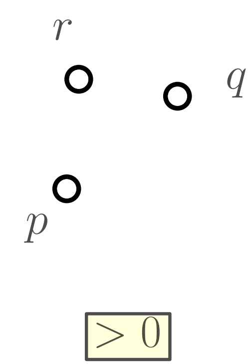
Orientation of (p, q, r) given by the sign of $\begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix}$.

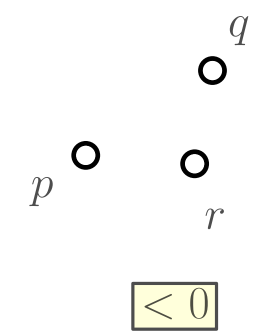




A close look at that example

Orientation of (p, q, r) given by the sign of $\begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix}$.



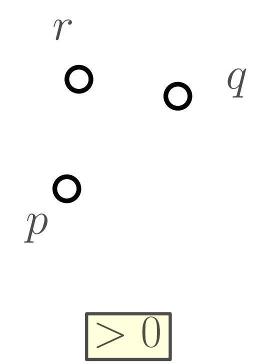


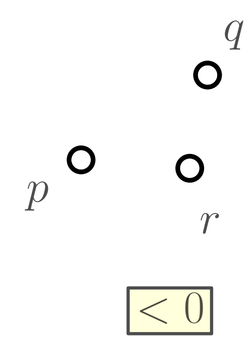

```
Float xp,yp,xq,yq,xr,yr;

Orientation = sign((xq-xp)*(yr-yp)-(xr-xp)*(yq-yp));
```

A close look at that example

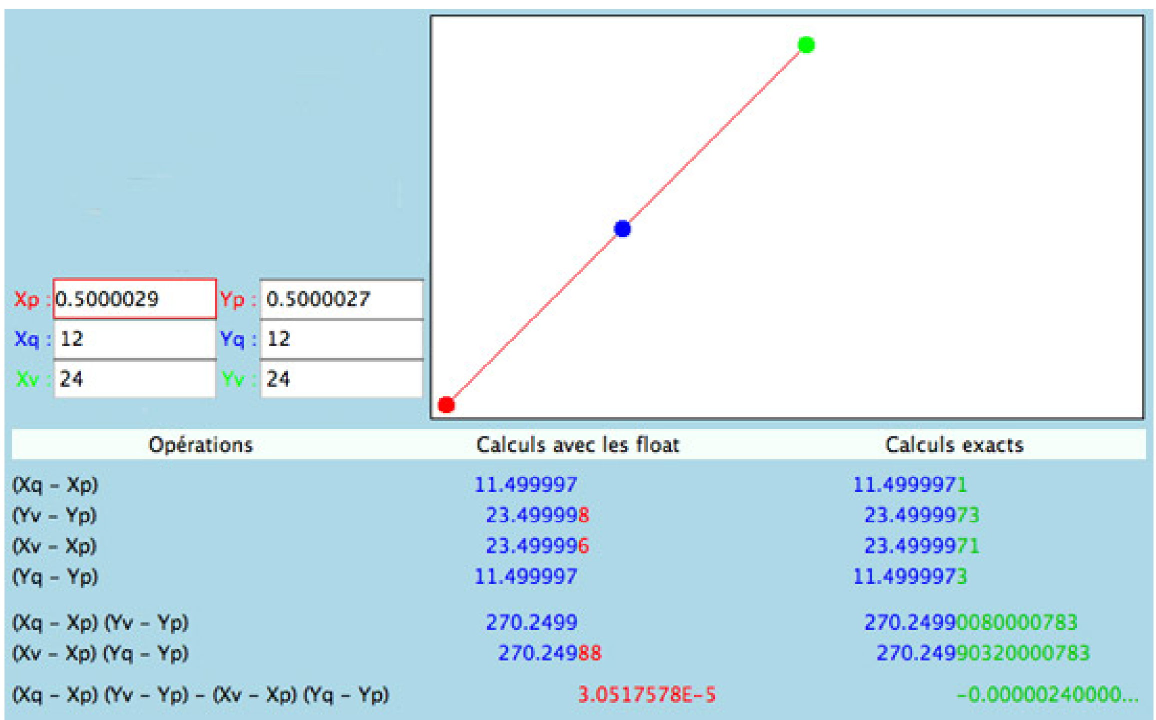
Orientation of (p, q, r) given by the sign of $\begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix}$.





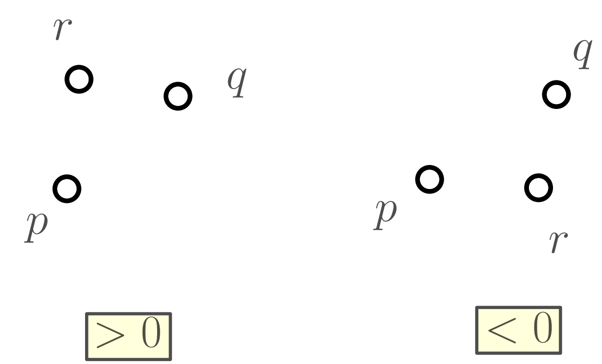

```
Float xp,yp,xq,yq,xr,yr;

Orientation = sign((xq-xp)*(yr-yp)-(xr-xp)*(yq-yp));
```



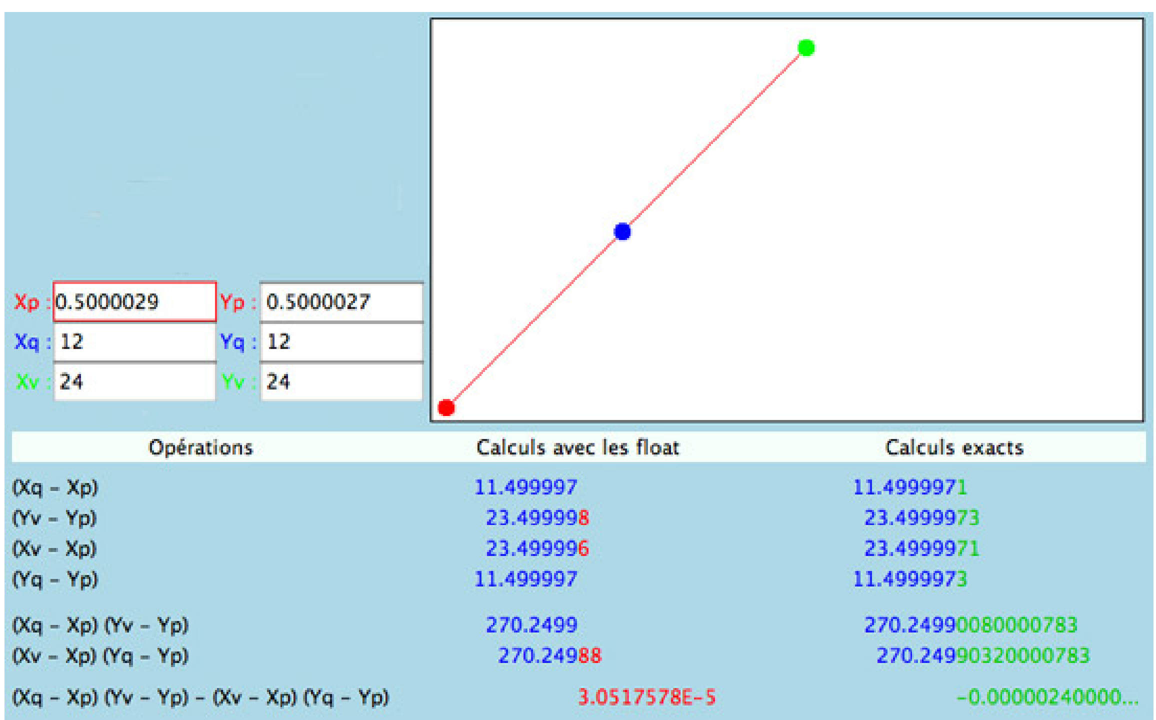
A close look at that example

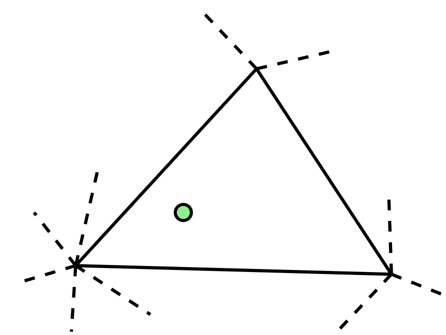
Orientation of (p,q,r) given by the sign of $\begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix}$.

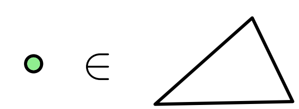



```
Float xp,yp,xq,yq,xr,yr;

Orientation = sign((xq-xp)*(yr-yp)-(xr-xp)*(yq-yp));
```

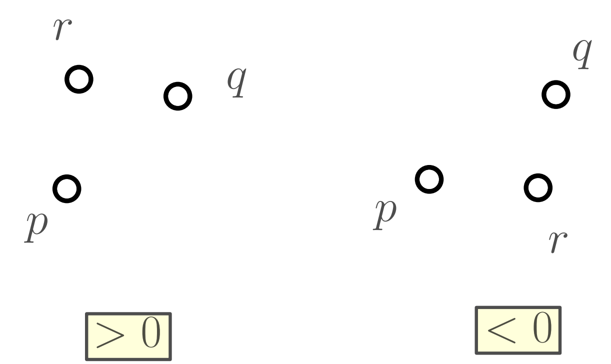




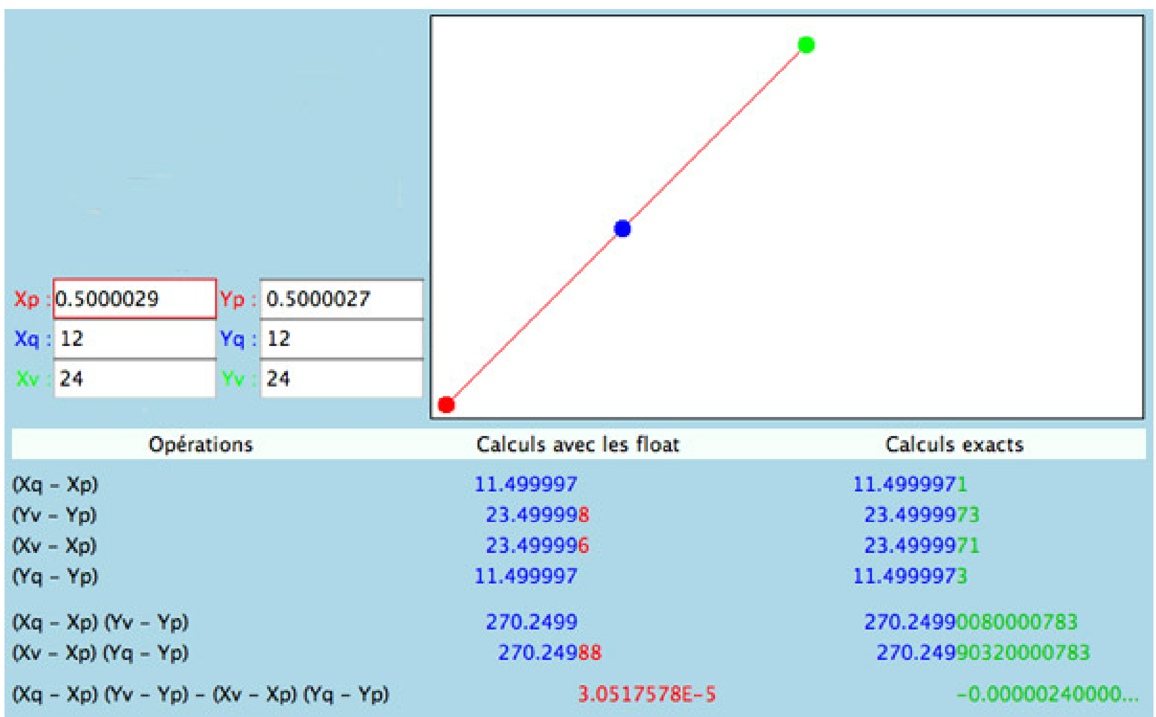


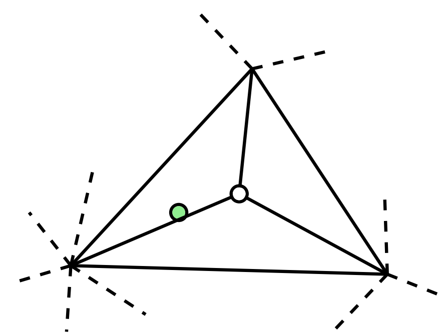
A close look at that example

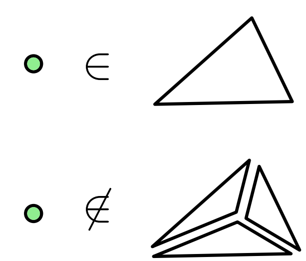
Orientation of (p,q,r) given by the sign of $\begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix}$.




```
Float xp,yp,xq,yq,xr,yr;
Orientation = sign((xq-xp)*(yr-yp)-(xr-xp)*(yq-yp));
```







Consequences of numerical rounding

A "correct" code can make incorrect decisions. These errors are inconsistent.

Crash, infinite loops, smooth execution but wrong answer... which is the worse?

Can be hard to detect...

Interval arithmetic

Keep the precision bounded but keep track of the error.

A number is represented by an interval (reduced to a single element if precision is sufficient).

Define all operations on intervals.

$24 - 0.5000027 = 23.499998$ becomes $[24, 24] - [0.5000027, 0.5000027] = [23.49999, 23.50000]$.

Interval arithmetic

Keep the precision bounded but keep track of the error.

A number is represented by an interval (reduced to a single element if precision is sufficient).

Define all operations on intervals.

$24 - 0.5000027 = 23.499998$ becomes $[24, 24] - [0.5000027, 0.5000027] = [23.49999, 23.50000]$.

If the interval does not contain 0 then we can decide the sign with certainty.

This suffices "most of the time".
Otherwise, we need more precision... Restart the computation with twice as many digits.

Interval arithmetic

Keep the precision bounded but keep track of the error.

A number is represented by an interval (reduced to a single element if precision is sufficient).

Define all operations on intervals.

$24 - 0.5000027 = 23.499998$ becomes $[24, 24] - [0.5000027, 0.5000027] = [23.49999, 23.50000]$.

If the interval does not contain 0 then we can decide the sign with certainty.

This suffices "most of the time".
Otherwise, we need more precision... Restart the computation with twice as many digits.

If the result of the computation is exactly 0 we will never have enough precision...
For those few cases, we need to be able to do the computations exactly.

Exact number types for integers, rational numbers, algebraic numbers.

Algebraic numbers

$\sqrt[5]{23}$ and $\sqrt[7]{25}$ are not integers, but we can still compare them exactly using integer arithmetic.

Algebraic numbers

$\sqrt[5]{23}$ and $\sqrt[7]{25}$ are not integers, but we can still compare them exactly using integer arithmetic.

A real r is algebraic if there exists a polynomial $P(X)$ with integer coefficients such that $P(r) = 0$.

Algebraic numbers

$\sqrt[5]{23}$ and $\sqrt[7]{25}$ are not integers, but we can still compare them exactly using integer arithmetic.

A real r is algebraic if there exists a polynomial $P(X)$ with integer coefficients such that $P(r) = 0$.

What about $n \in \mathbb{N}$, $f \in \mathbb{Q}$, $\sqrt{2}$, $\sqrt[5]{17}$, e, π... ?

Algebraic numbers

$\sqrt[5]{23}$ and $\sqrt[7]{25}$ are not integers, but we can still compare them exactly using integer arithmetic.

A real r is algebraic if there exists a polynomial $P(X)$ with integer coefficients such that $P(r) = 0$.

What about $n \in \mathbb{N}$, $f \in \mathbb{Q}$, $\sqrt{2}$, $\sqrt[5]{17}$, e, π... ?

The set of algebraic numbers is closed under $+, -, \times, /, x \mapsto x^t$ for $t \in \mathbb{Q}$ (in particular $\sqrt{\ }$).

Algebraic numbers

$\sqrt[5]{23}$ and $\sqrt[7]{25}$ are not integers, but we can still compare them exactly using integer arithmetic.

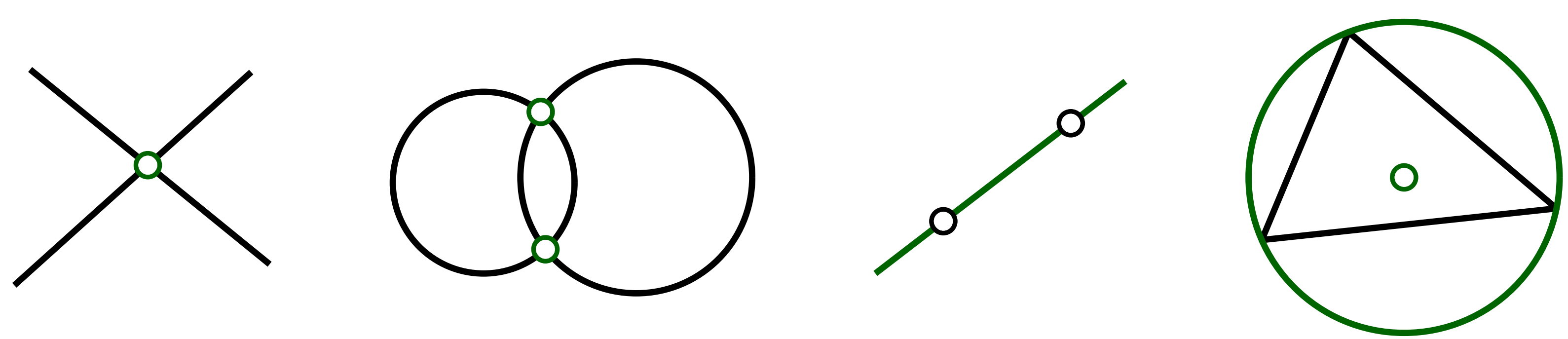
A real r is algebraic if there exists a polynomial $P(X)$ with integer coefficients such that $P(r) = 0$.

What about $n \in \mathbb{N}$, $f \in \mathbb{Q}$, $\sqrt{2}$, $\sqrt[5]{17}$, e, π... ?

The set of algebraic numbers is closed under $+, -, \times, /, x \mapsto x^t$ for $t \in \mathbb{Q}$ (in particular $\sqrt{\ }$).

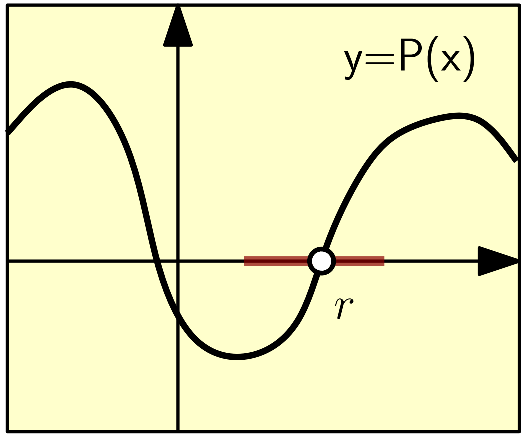
There are few algebraic numbers (ie countably many).

The result of most classical operations on geometric objects defined by integers
can be described using algebraic numbers.



Representing and manipulating algebraic numbers

An algebraic number can be represented by a polynomial (a family of integers) and an isolation interval.

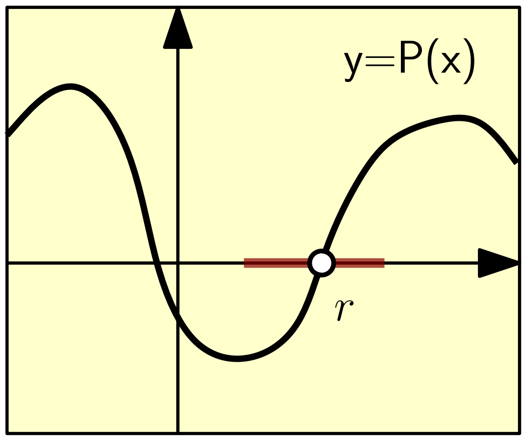


Interval containing a single root of P.

Ex: $\sqrt{2} \simeq (X^2 - 1, [1, 2])$.

Representing and manipulating algebraic numbers

An algebraic number can be represented by a polynomial (a family of integers) and an isolation interval.



Interval containing a single root of P.

Ex: $\sqrt{2} \simeq (X^2 - 1, [1, 2])$.

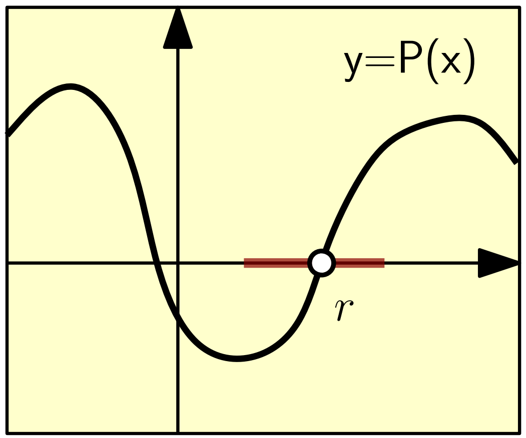
Given two algebraic numbers a and b represented by polynomials and isolation intervals,

we can compute a polynomial / isolation interval that represents:

$a + b, a - b, a \times b, \frac{a}{b}, a^2, \sqrt{a}$, etc...

Representing and manipulating algebraic numbers

An algebraic number can be represented by a polynomial (a family of integers) and an isolation interval.



Interval containing a single root of P.

Ex: $\sqrt{2} \simeq (X^2 - 1, [1,2])$.

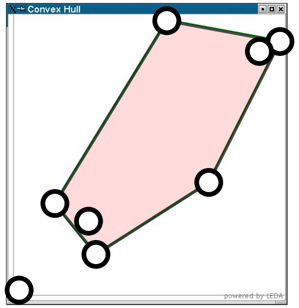
Given two algebraic numbers a and b represented by polynomials and isolation intervals,

we can compute a polynomial / isolation interval that represents:

$$a+b, a-b, a\times b, \frac{a}{b}, a^2, \sqrt{a}, \text{ etc...}$$

Implemented in the C/C++ CORE library.

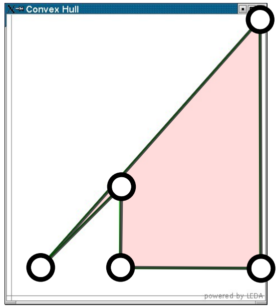
Using algebraic numbers




```
Float xp,yp,xq,yq,xr,yr;

Orientation = sign((xq-xp)*(yr-yp)-(xr-xp)*(yq-yp));
```

leads to



These problems can be avoided by using

```
Core::Expr xp,yp,xq,yq,xr,yr;
Orientation = sign((xq-xp)*(yr-yp)-(xr-xp)*(yq-yp));
```

Decision VS constructions

Distinguish between decision (for branching) and constructions.

Decisions are made by evaluating signs of polynomial in the input and can be filtered.

Constructions produce a geometric object from the input; representing exactly that object is costly.

Decision VS constructions

Distinguish between decision (for branching) and constructions.

Decisions are made by evaluating signs of polynomial in the input and can be filtered.

Constructions produce a geometric object from the input; representing exactly that object is costly.

If the decisions are correct, the "type" of the result is correct (constructions do not matter much).

Using repeated substitutions, we can avoid using constructions when branching.

Decision VS constructions

Distinguish between decision (for branching) and constructions.

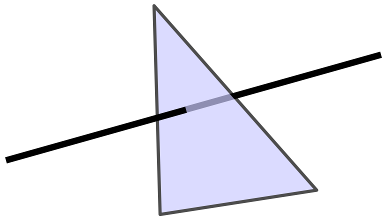
Decisions are made by evaluating signs of polynomial in the input and can be filtered.

Constructions produce a geometric object from the input; representing exactly that object is costly.

If the decisions are correct, the "type" of the result is correct (constructions do not matter much).

Using repeated substitutions, we can avoid using constructions when branching.

Ex: line/triangle intersection test



find intersection with plane, compute barycentric coordinates.

Decision VS constructions

Distinguish between decision (for branching) and constructions.

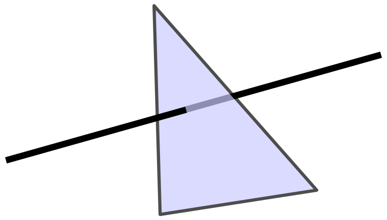
Decisions are made by evaluating signs of polynomial in the input and can be filtered.

Constructions produce a geometric object from the input; representing exactly that object is costly.

If the decisions are correct, the "type" of the result is correct (constructions do not matter much).

Using repeated substitutions, we can avoid using constructions when branching.

Ex: line/triangle intersection test



find intersection with plane, compute barycentric coordinates.
$\rightarrow$ evaluate the sign of polynomials of degree 6.

Decision VS constructions

Distinguish between decision (for branching) and constructions.

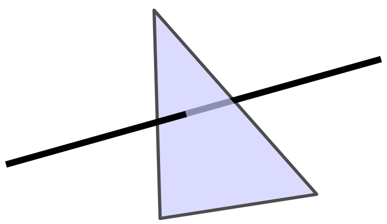
Decisions are made by evaluating signs of polynomial in the input and can be filtered.

Constructions produce a geometric object from the input; representing exactly that object is costly.

If the decisions are correct, the "type" of the result is correct (constructions do not matter much).

Using repeated substitutions, we can avoid using constructions when branching.

Ex: line/triangle intersection test



find intersection with plane, compute barycentric coordinates.
$\rightarrow$ evaluate the sign of polynomials of degree 6.

Evaluate 3D orientations of quadruples of points

Decision VS constructions

Distinguish between decision (for branching) and constructions.

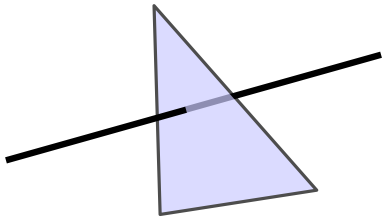
Decisions are made by evaluating signs of polynomial in the input and can be filtered.

Constructions produce a geometric object from the input; representing exactly that object is costly.

If the decisions are correct, the "type" of the result is correct (constructions do not matter much).

Using repeated substitutions, we can avoid using constructions when branching.

Ex: line/triangle intersection test



find intersection with plane, compute barycentric coordinates.
$\rightarrow$ evaluate the sign of polynomials of degree 6.

Evaluate 3D orientations of quadruples of points
$\rightarrow$ evaluate the sign of polynomials of degree 3.

Wrap-up: robustness

Treating degeneracies requires great care.

Numerical problems will arise.

If not treated properly, they produce crashes, infinite loops or wrong results.

Exact number types exist and are implemented. This is good enough for prototyping.

Reliability and efficiency are achieved by using good predicates

and filtering exact number type with interval arithmetic.

The End