Computing a Delaunay triangulation

Lecture 10, CS 4235
18 march 2004

Antoine Vigneron

antoine@comp.nus.edu.sg

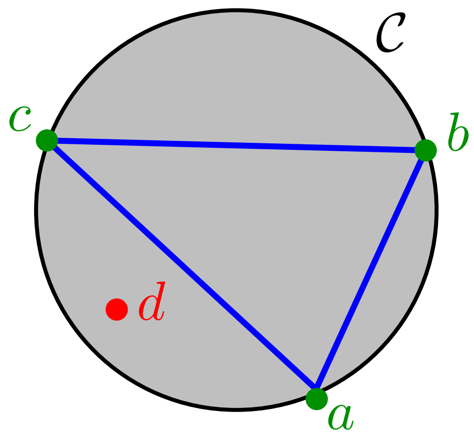
National University of Singapore

Outline

- RIC of the Delaunay triangulation
- optimal $O(n \log n)$ time randomized algorithm
- application: computing a Voronoi diagram
- references
  - D. Mount Lecture 18
  - textbook chapter 9
  - H. Edelsbrunner's book: Geometry and topology of mesh generation, chapter 1
  - demo (J. Snoeyink) at:
    http://www.cs.ubc.ca/spider/snoeyink/demos/crust/home.html

Incircle test

Definition



$\mathrm{inCircle}(a, b, c, d) < 0$

- assume triangle abc is counterclockwise
- let $\mathcal{C}$ be the circumcircle of abc
- we want to design a test $\mathrm{inCircle}(\cdot)$ such that
  - $\mathrm{inCircle}(a, b, c, d) = 0$ if $d \in \mathcal{C}$
  - $\mathrm{inCircle}(a, b, c, d) > 0$ if d is outside $\mathcal{C}$
  - $\mathrm{inCircle}(a, b, c, d) < 0$ if d is inside $\mathcal{C}$

Expression

- we use the following expression:

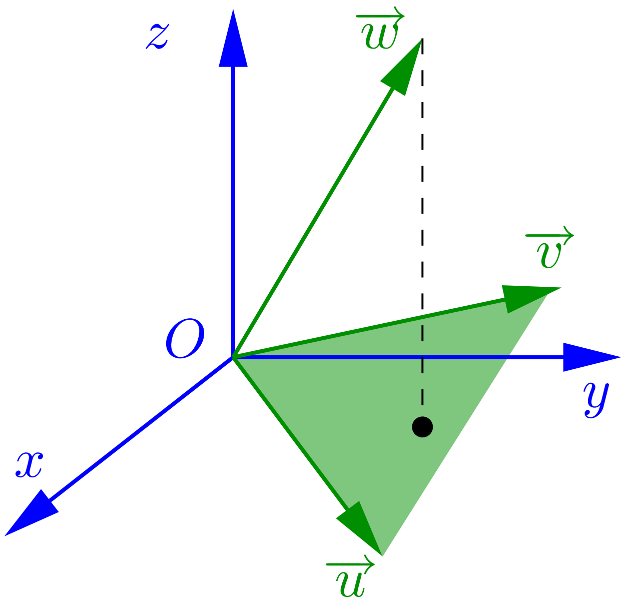
$$\mathrm{inCircle}(a,b,c,d) = \det \begin{pmatrix} 1 & a_x & a_y & {a_x}^2 + {a_y}^2 \\ 1 & b_x & b_y & {b_x}^2 + {b_y}^2 \\ 1 & c_x & c_y & {c_x}^2 + {c_y}^2 \\ 1 & d_x & d_y & {d_x}^2 + {d_y}^2 \end{pmatrix}$$

- why does it work?
  - next ten slides: proof with geometric interpretation
  - D. Mount's notes 18: different proof, through algebra
  - be careful: we reversed the sign of $\mathrm{inCircle}(\cdot)$ with respect to D. Mount's notes, in order to simplify the following proof.

Orientation of vectors in $\mathbb{R}^3$

- the orientation of $(\overrightarrow{u}, \overrightarrow{v}, \overrightarrow{w})$ is given by the sign of

$$\det \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}$$



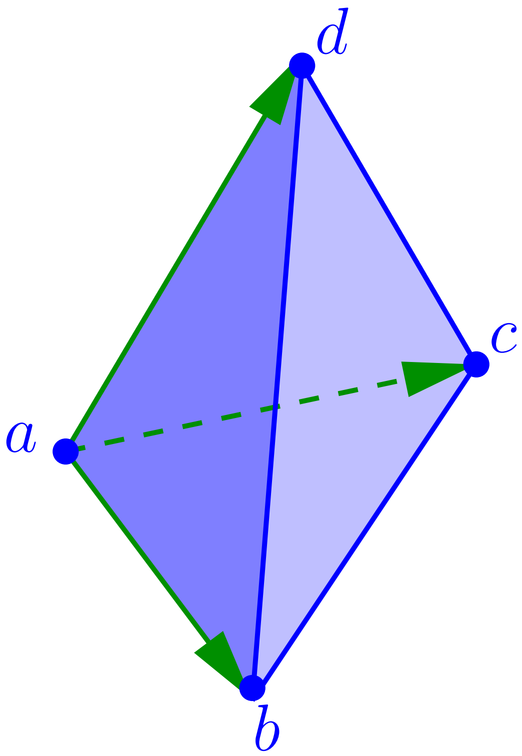
Orientation $(\overrightarrow{u}, \overrightarrow{v}, \overrightarrow{w}) > 0$

Orientation $(\overrightarrow{v}, \overrightarrow{u}, \overrightarrow{w}) < 0$

$\Leftarrow$ right thumb rule

Orientation of a tetrahedron

- orientation of tetrahedron $abcd =$ orientation of $(\overrightarrow{ab}, \overrightarrow{ac}, \overrightarrow{ad})$



Orientation$(abcd) =$

Orientation $(\overrightarrow{ab}, \overrightarrow{ac}, \overrightarrow{ad}) > 0$

$\Leftarrow$ right thumb rule

Orientation of a tetrahedron

- Orientation $(abcd)$

$$= \det \begin{pmatrix} b_x - a_x & b_y - a_y & b_z - a_z \\ c_x - a_x & c_y - a_y & c_z - a_z \\ d_x - a_x & d_y - a_y & d_z - a_z \end{pmatrix}$$

$$= \det \begin{pmatrix} 1 & a_x & a_y & a_z \\ 0 & b_x - a_x & b_y - a_y & b_z - a_z \\ 0 & c_x - a_x & c_y - a_y & c_z - a_z \\ 0 & d_x - a_x & d_y - a_y & d_z - a_z \end{pmatrix}$$

- why?
- Develop with respect to first column

Orientation of a tetrahedron

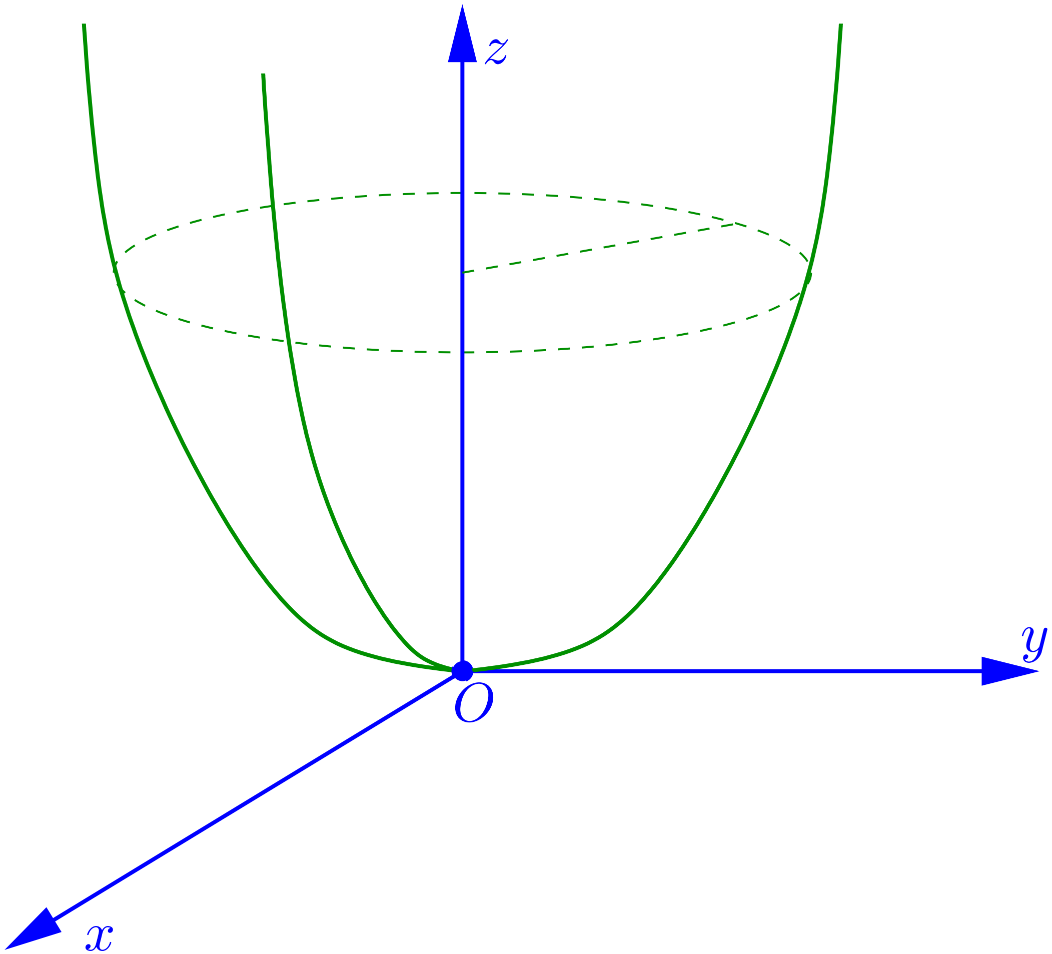
- add first row to the other rows
- Orientation $(abcd)$

$$= \det \begin{pmatrix} 1 & a_x & a_y & a_z \\ 1 & b_x & b_y & b_z \\ 1 & c_x & c_y & c_z \\ 1 & d_x & d_y & d_z \end{pmatrix}$$

- note that it generalizes the counterclockwise (CCW) predicate in $\mathbb{R}^2$ (see lecture 1)

Paraboloid $\mathcal{P}$

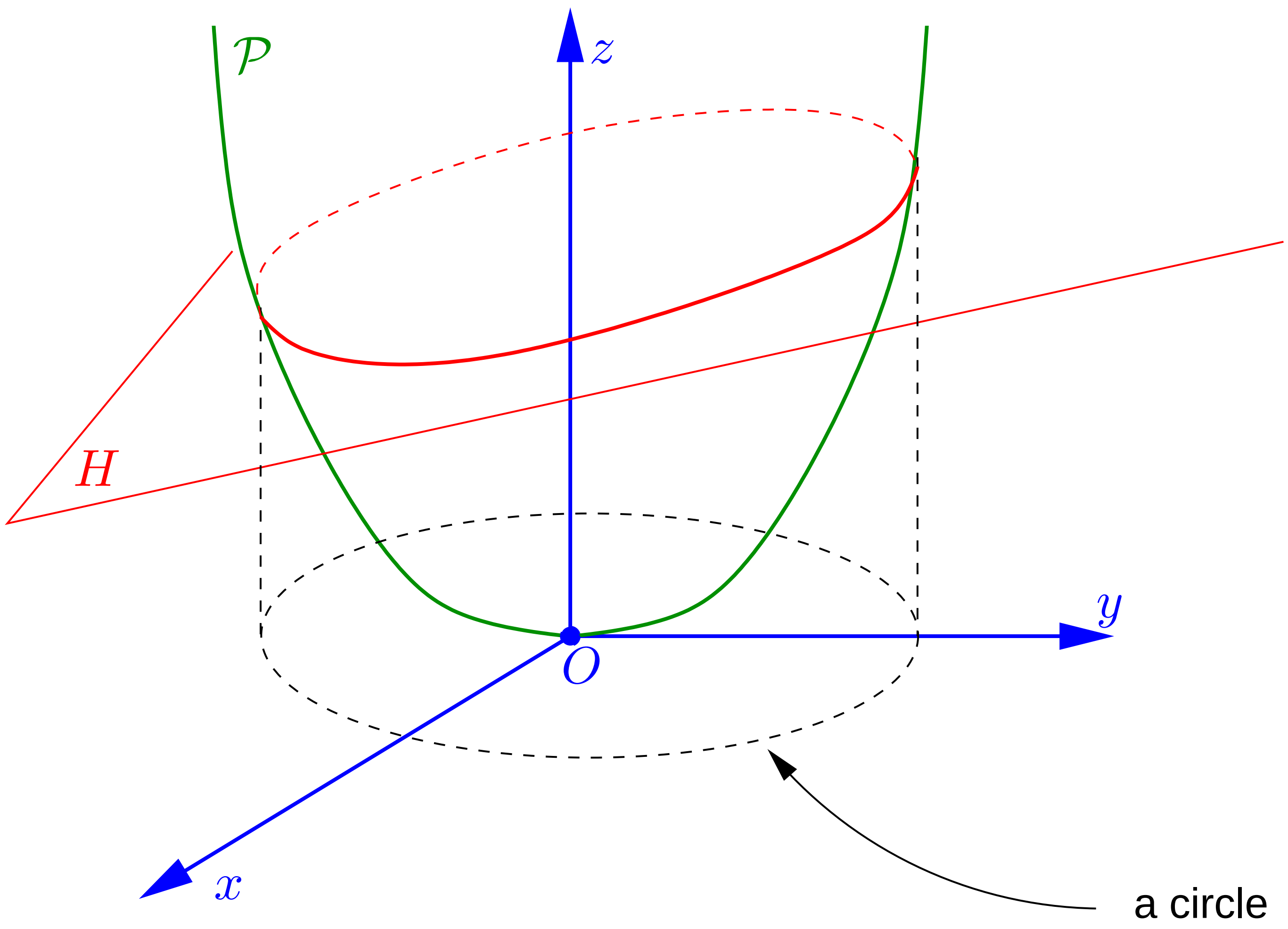
- in $\mathbb{R}^3$, let $\mathcal{P}$ be the paraboloid with equation $z = x^2 + y^2$



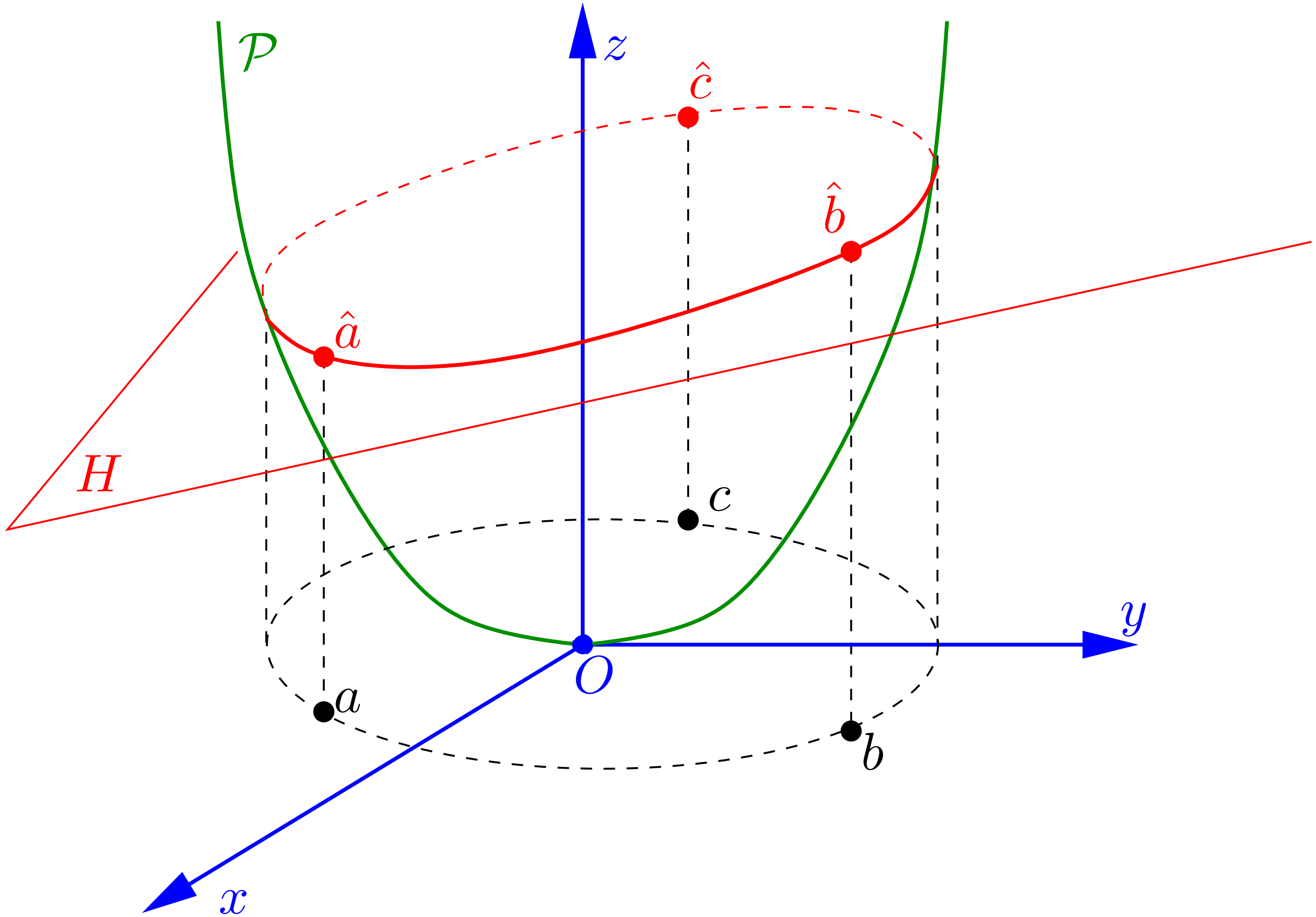
Property

- let H be a non–vertical plane
- H has equation $z = \alpha x + \beta y + \gamma$
- the projection of $H \cap \mathcal{P}$ onto plane Oxy has equation $x^2 + y^2 = \alpha x + \beta y + \gamma$
  - this is a circle
- property: the projection of $H \cap \mathcal{P}$ onto plane Oxy is a circle

Property



Proof



Proof

- let $p = (p_x, p_y)$
- we lift p onto $\mathcal{P}$ and obtain $\hat{p} = (p_x, p_y, p_x^2 + p_y^2)$
- the transformation $p \rightarrow \hat{p}$ is called the *lifting map*

Proof

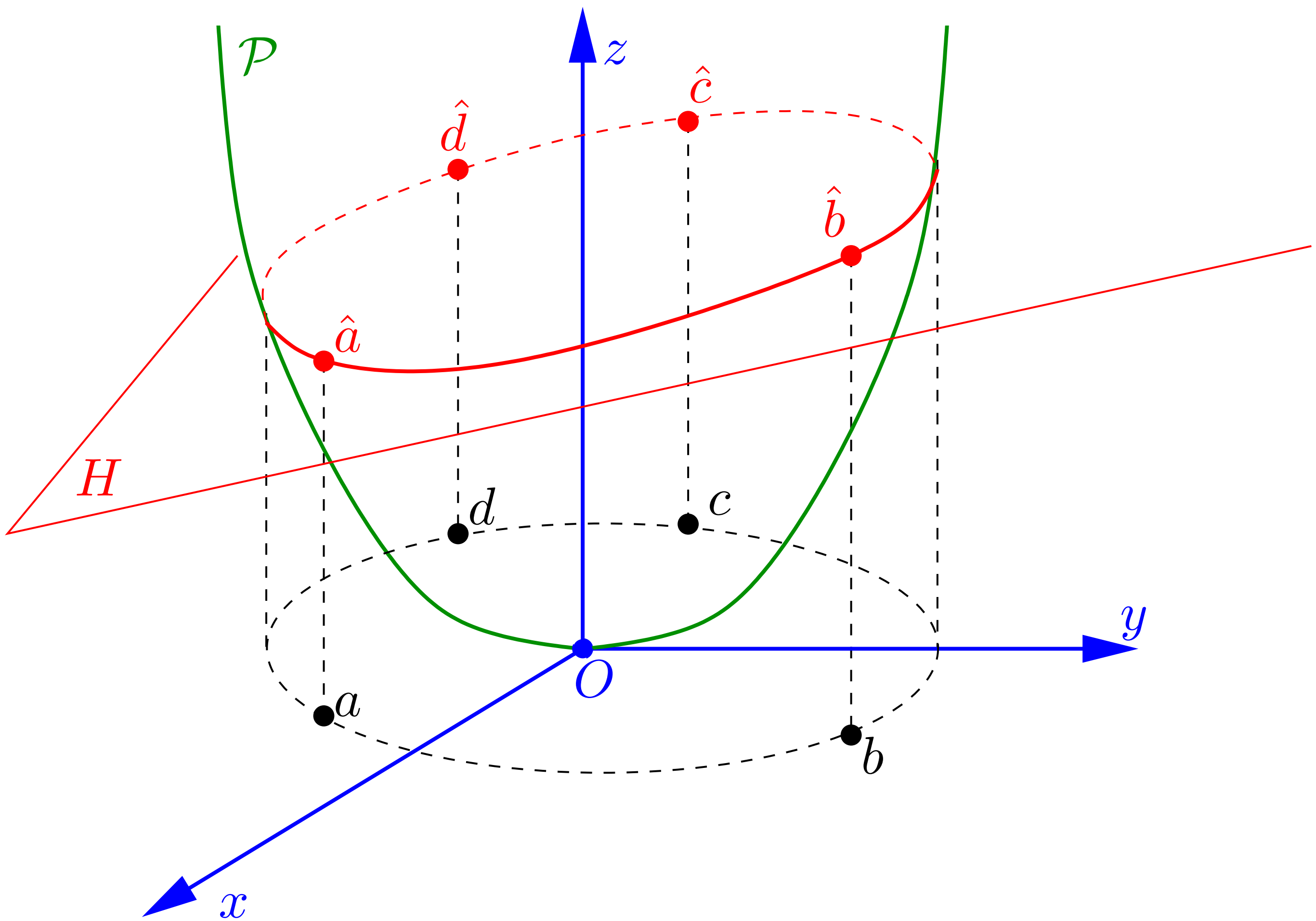
- we lift a, b, c and d:

$$\begin{aligned}\hat{a} &= (a_x, a_y, a_x^2 + a_y^2)\\ \hat{b} &= (b_x, b_y, b_x^2 + b_y^2)\\ \hat{c} &= (c_x, c_y, c_x^2 + c_y^2)\\ \hat{d} &= (d_x, d_y, d_x^2 + d_y^2)\end{aligned}$$

- we denote by H the plane through $\{\hat{a}, \hat{b}, \hat{c}\}$
- $\mathrm{inCircle}(a, b, c, d) = 0$ means that Orientation$(\hat{a}, \hat{b}, \hat{c}, \hat{d}) = 0$
  - so $\hat{d} \in H$
  - we project to horizontal
  - we obtain that d is in the circumcircle of abc

Proof: first case

- a, b, c, d cocircular if $\hat{a}, \hat{b}, \hat{c}, \hat{d}$ coplanar

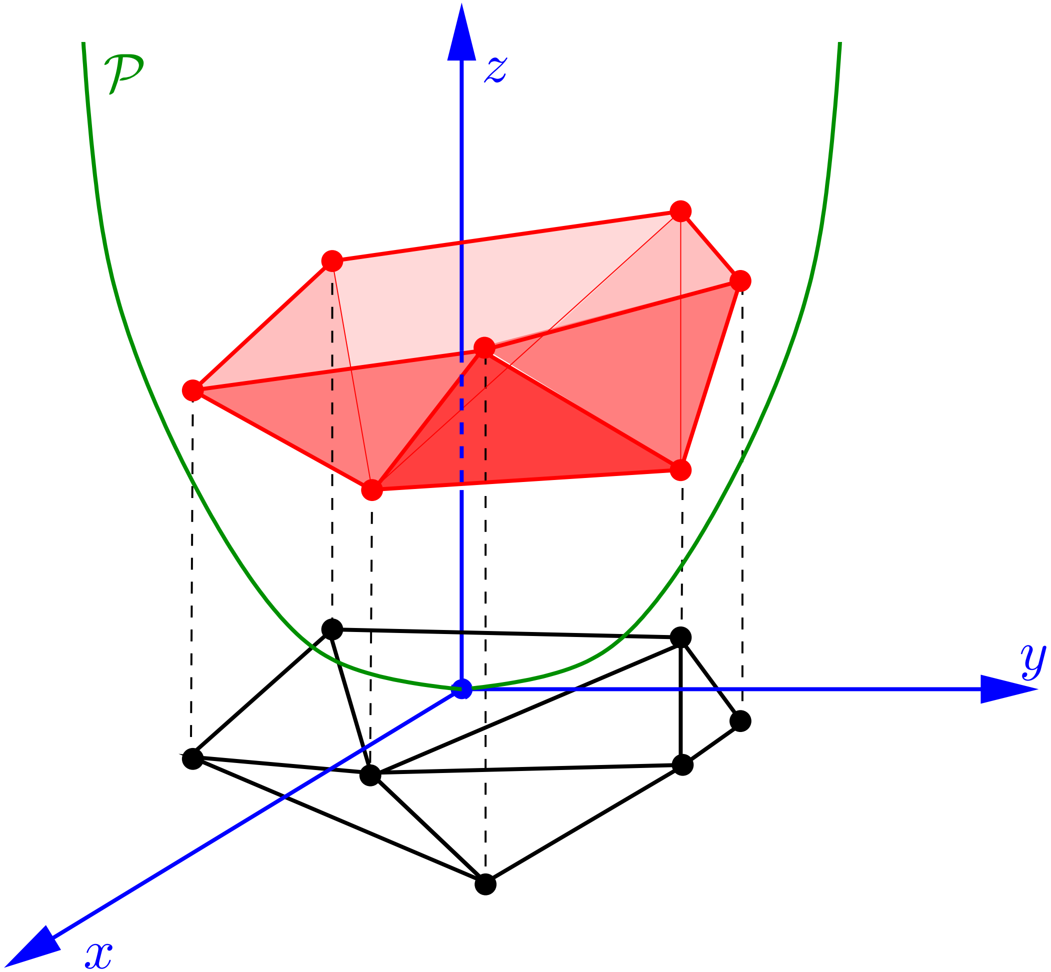


Proof: other cases

- $\mathrm{inCircle}(a, b, c, d) > 0$ means that Orientation$(\hat{a}, \hat{b}, \hat{c}, \hat{d}) > 0$
  - then $\hat{d}$ is above H
  - so d is outside the circumcircle of abc
- $\mathrm{inCircle}(a, b, c, d) < 0$ means that Orientation$(\hat{a}, \hat{b}, \hat{c}, \hat{d}) < 0$
  - then $\hat{d}$ is below H
  - so d is inside the circumcircle of abc

New interpretation of the Delaunay triangulation

Lifting $\mathcal{DT}(P)$

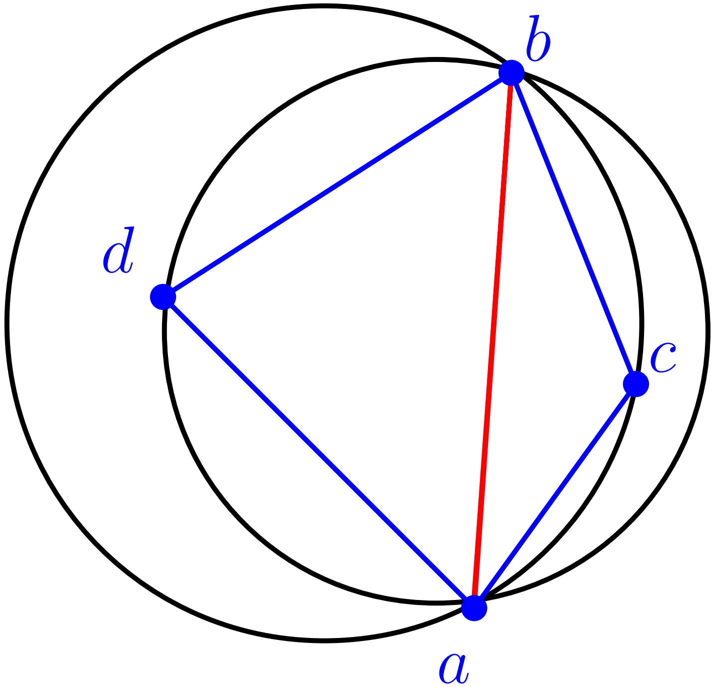


Circumcircle property

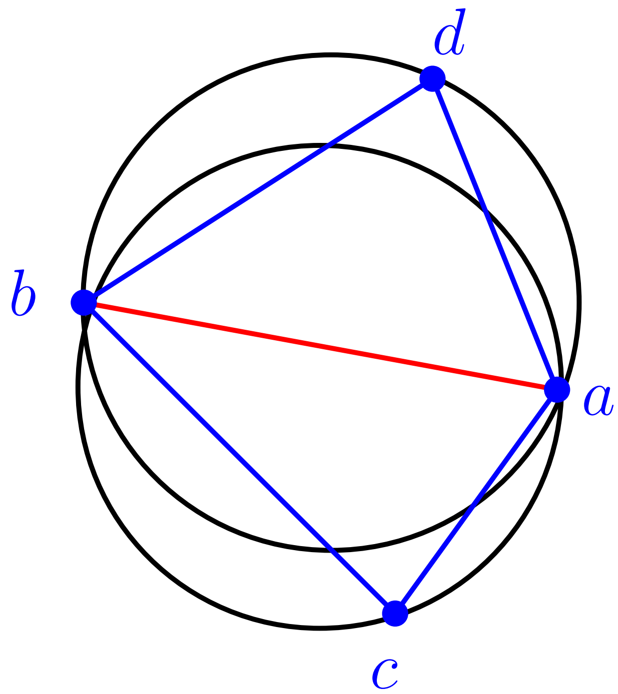
- $P = \{p_1, p_2, \ldots p_n\}$ is a set of points in the plane in general position
- we denote $\hat{P} = \{\hat{p}_1, \hat{p}_2, \ldots \hat{p}_n\}$
- last lecture: triangle $p_i p_j p_k$ is a face of $\mathcal{DT}(P)$ iff its circumcircle is empty
  - it means that $\forall p \in P \setminus \{p_i, p_j, p_k\}$, $\hat{p}$ is above the plane through $\hat{p}_i \hat{p}_j \hat{p}_k$
  - in other words, $\hat{p}_i \hat{p}_j \hat{p}_k$ is a facet of the lower hull of $\hat{P}$
- Theorem: $\mathcal{DT}(P)$ is the projection of the edges of the lower hull of $\hat{P}$ onto the plane $z = 0$

Edge flip

Property



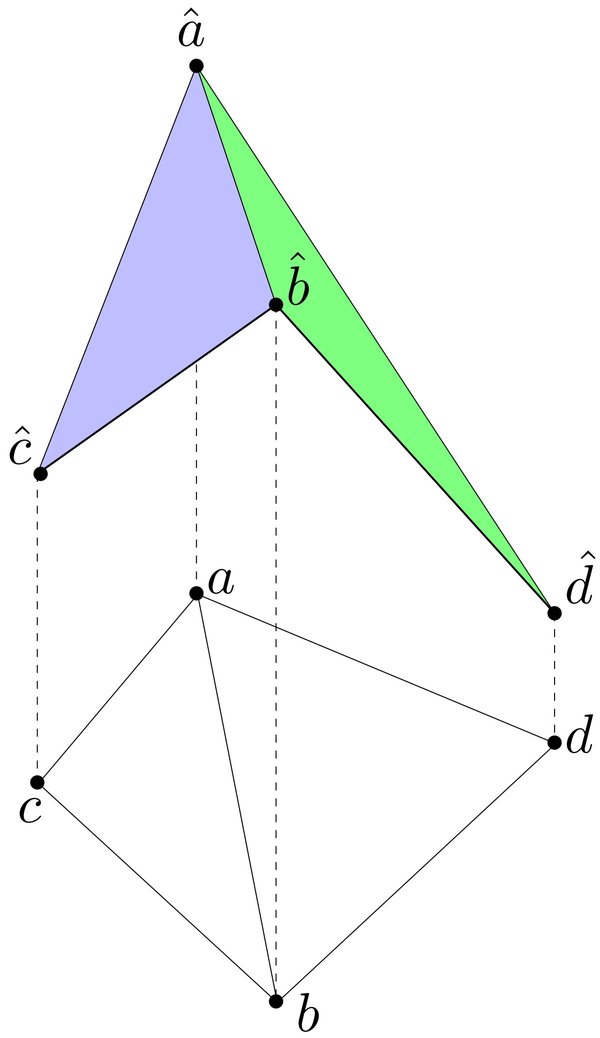
or



Property

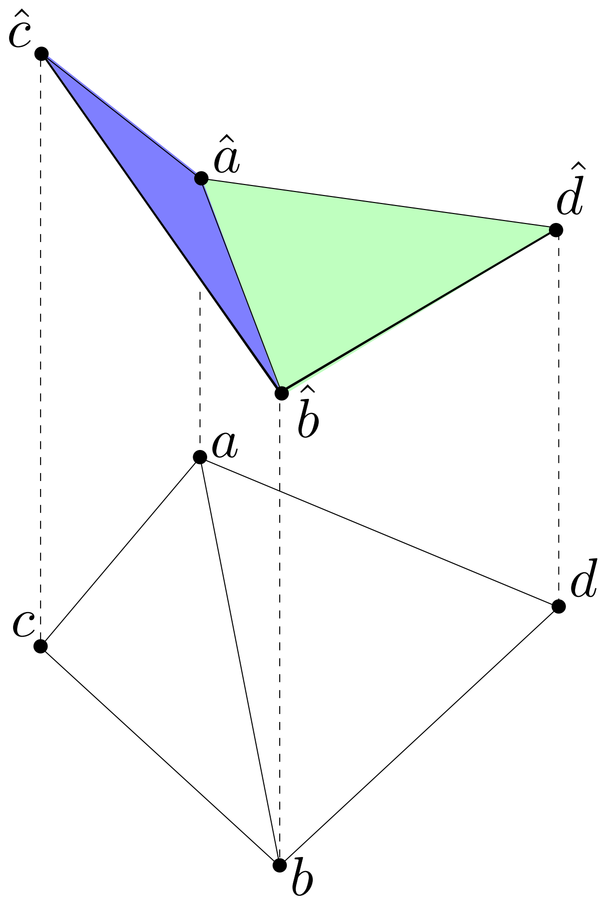
- let $acbd$ be a quadrilateral with diagonal $\overline{ab}$
- then either
  - c is inside the circumcircle of abd and d is inside the circumcircle of abc
  - or c is outside circumcircle of abd and d is outside the circumcircle of abc

Proof (by picture)



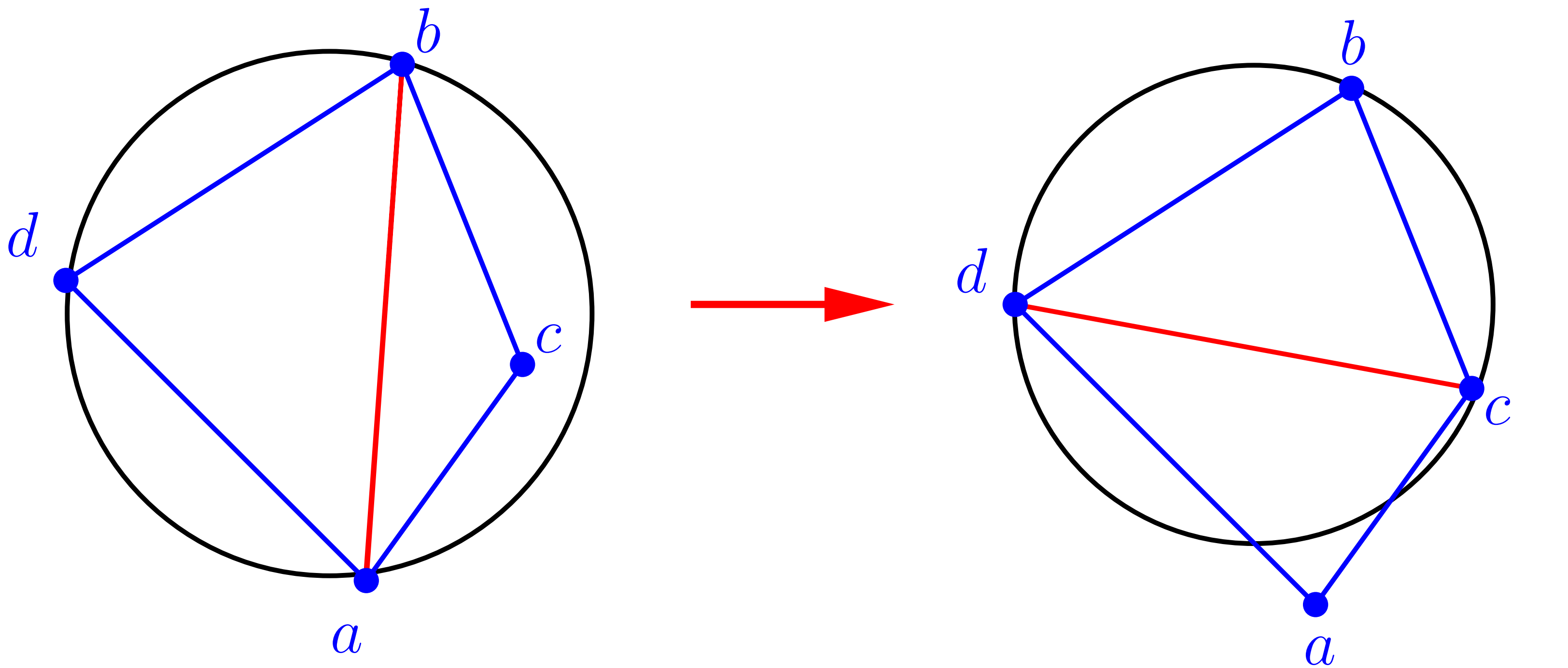
Concave

OR



Convex

Edge flip: definition



ab is illegal

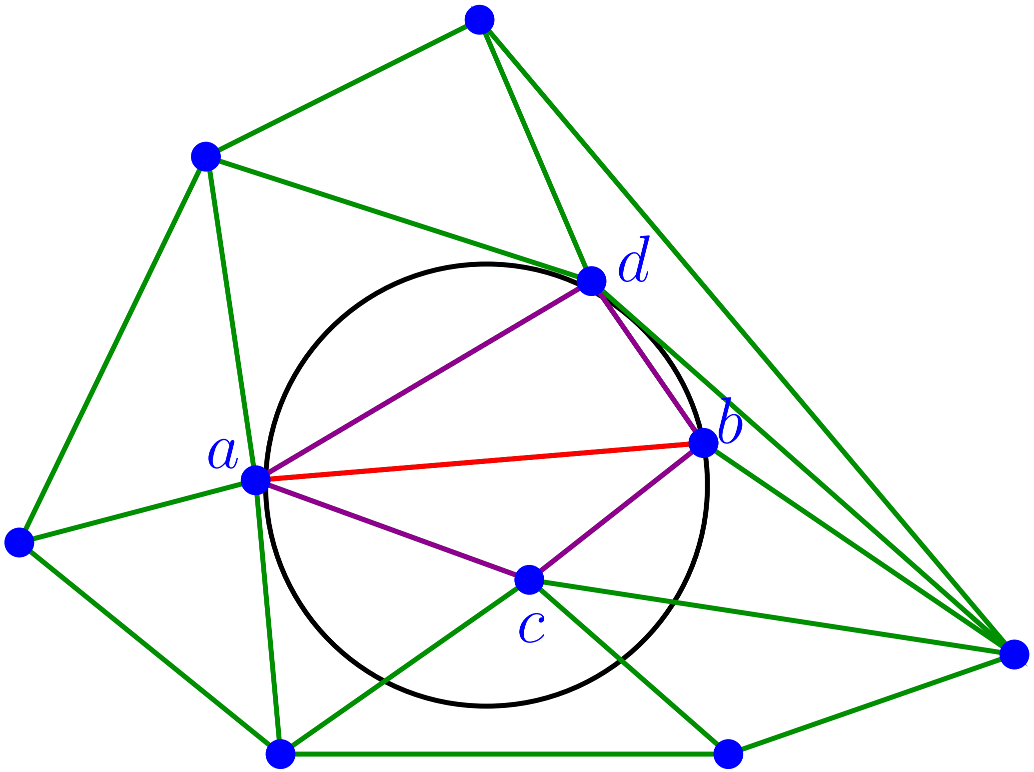
cd is locally Delaunay

Definitions

- let P be a set of n points in $\mathbb{R}^2$
- P is in general position: no 4 points are cocircular
- let $\mathcal{T}$ be a triangulation of P
- let ab be an edge of $\mathcal{T}$
- let $(c, d) \in P^2$ such that abc and abd are triangles of $\mathcal{T}$
- ab is *locally Delaunay* iff d is outside the circumcircle of abc
- ab is *illegal* iff d is inside the circumcircle of abc
- note that we can decide whether ab is locally Delaunay or illegal by computing the sign of $CCW(abc)$ and the sign of $\mathrm{inCircle}(a, b, c, d)$

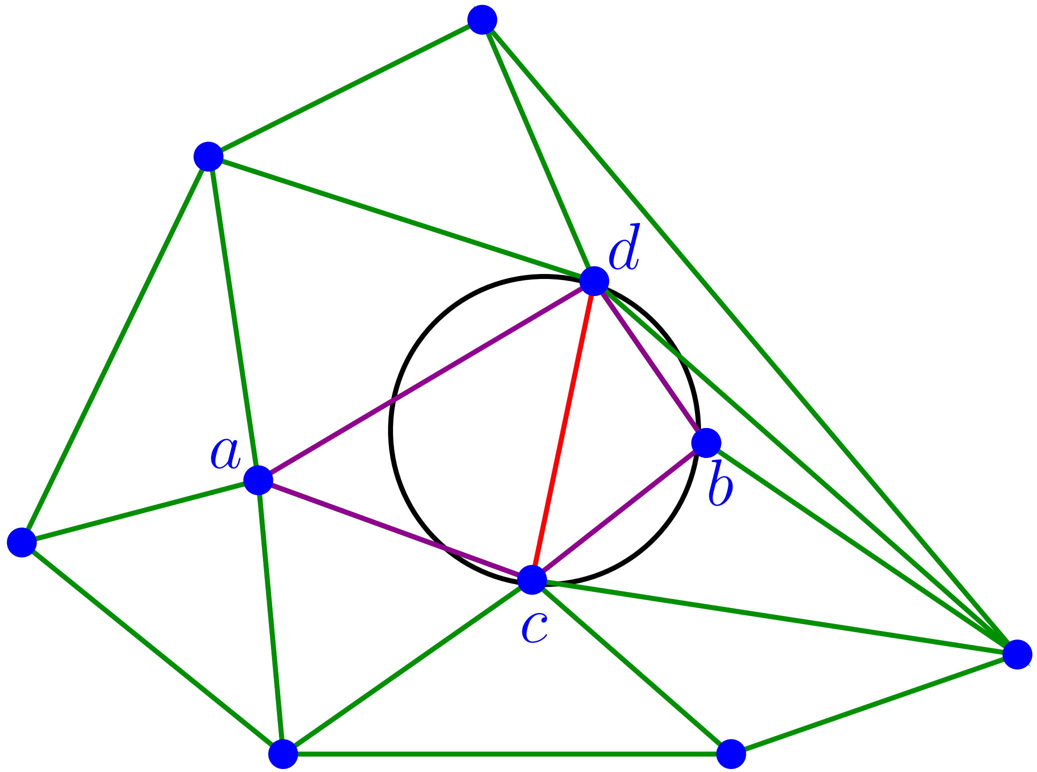
Definition

- if ab is illegal, we can perform an *edge flip*: remove ab from $\mathcal{T}$ and insert cd
- now cd is locally Delaunay

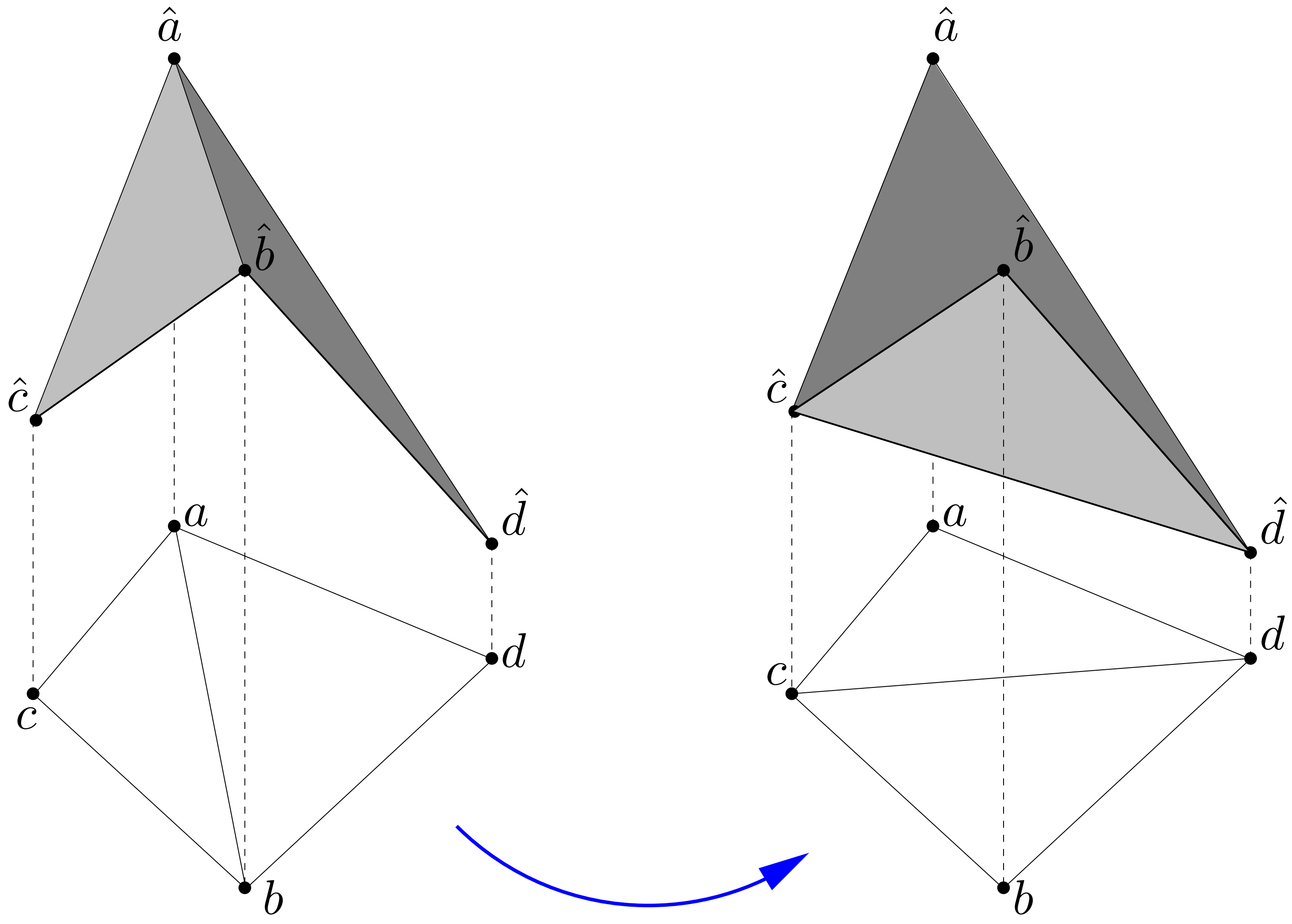


Definition

- if ab is illegal, we can perform an *edge flip*: remove ab from $\mathcal{T}$ and insert cd
- now cd is locally Delaunay



Edge flip: interpretation



the lifted triangulation gets lower
the upper envelope becomes convex

A first algorithm

Theorem

- let $\mathcal{T}$ be a triangulation of P
- $\mathcal{T} = \mathcal{DT}(P)$ iff all the edges of $\mathcal{T}$ are locally Delaunay
- proof:
  - if $\mathcal{T}$ is Delaunay, then clearly all edges are locally Delaunay (by definition)
  - other direction: non trivial
    - see textbook Theorem 9.8
    - or use the lifting map: locally Delaunay $\Leftrightarrow$ locally convex $\Leftrightarrow$ globally convex $\Leftrightarrow$ globally Delaunay

Idea

- draw a triangulation $\mathcal{T}$ of P
- if all the edges of $\mathcal{T}$ are locally Delaunay, we are done
- otherwise, pick an illegal edge and flip it
- repeat this process until all edges are locally Delaunay

Pseudocode

Algorithm *SlowDelaunay*(P)
Input: a set P of n points in $\mathbb{R}^2$
Output: $\mathcal{DT}(P)$

1. compute a triangulation $\mathcal{T}$ of P
2. initialize a stack containing all the edges of $\mathcal{T}$
3. **while** stack is non–empty
4. **do** pop ab from stack and unmark it
5. **if** ab is illegal then
6. **do** flip ab to cd
7. **for** $xy \in \{ac, cb, bd, da\}$
8. **do if** xy is not marked
9. **then** mark xy and push it on stack
10. **return** $\mathcal{T}$

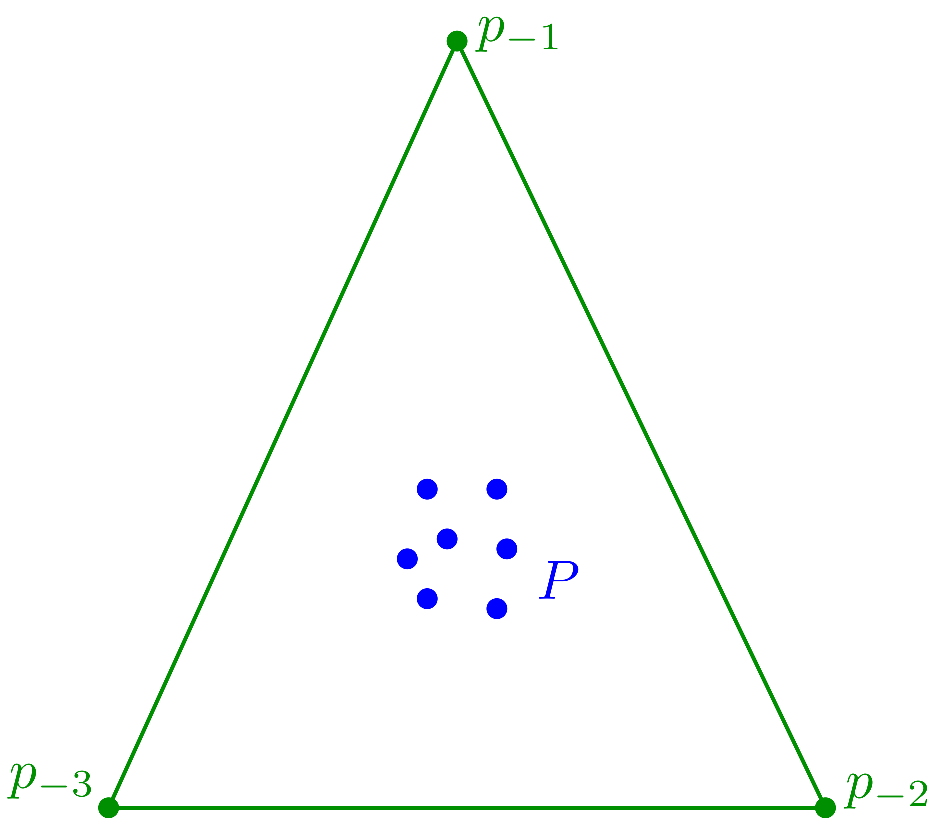
Analysis

- it is not obvious that this program halts!
- in fact runs in $\Theta(n^2)$ time
- proof using lifting map
  - each time we flip an edge, the lifted triangulation gets lower
  - so an edge can be flipped only once: afterward it remains above the lifted triangulation
  - there are $O(n^2)$ edges
  - so the algorithm runs in $O(n^2)$ time
  - lower bound left as an exercise

Randomized incremental algorithm

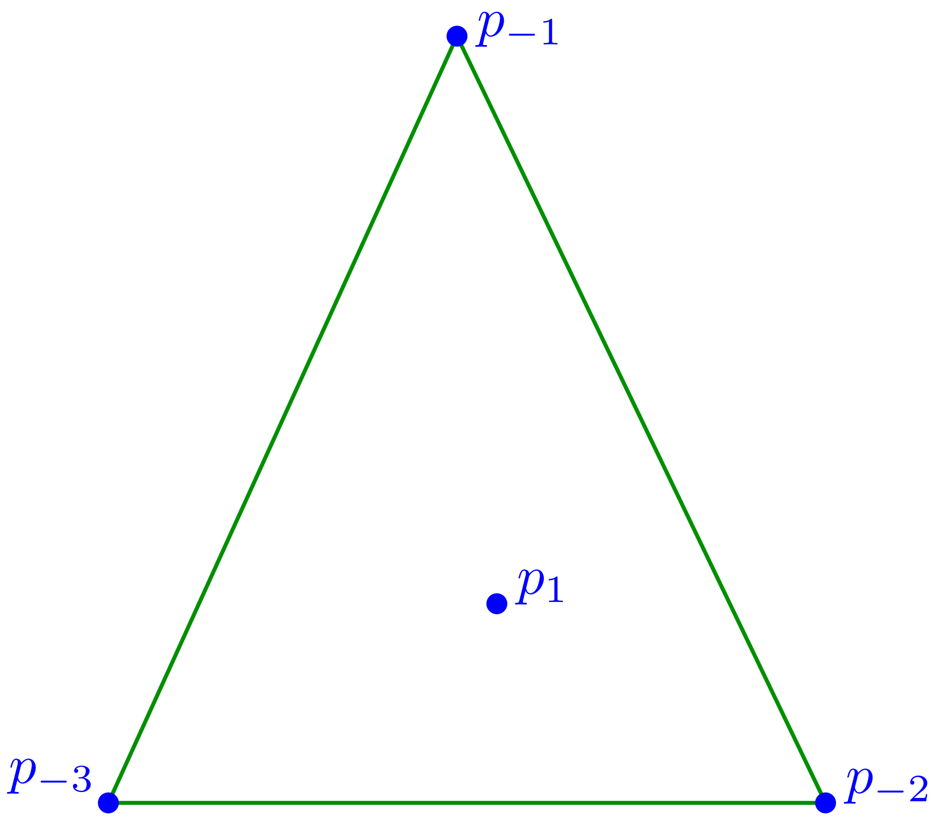
Preliminary

- let $(p_1, p_2, p_3 \dots p_n)$ be a random permutation of P
- let $p_{-3}p_{-2}p_{-1}$ be a large triangle containing P

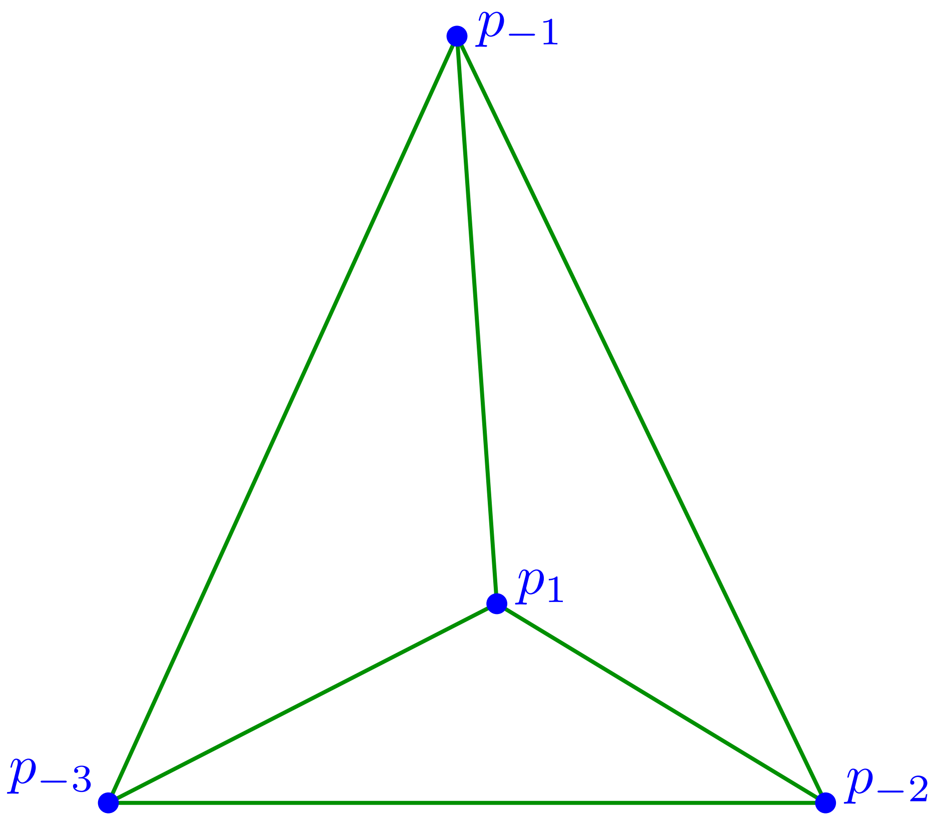


- for all i we denote $P_i = \{p_{-3}, p_{-2}, p_{-1}, p_1, p_2, \dots p_i\}$

First step



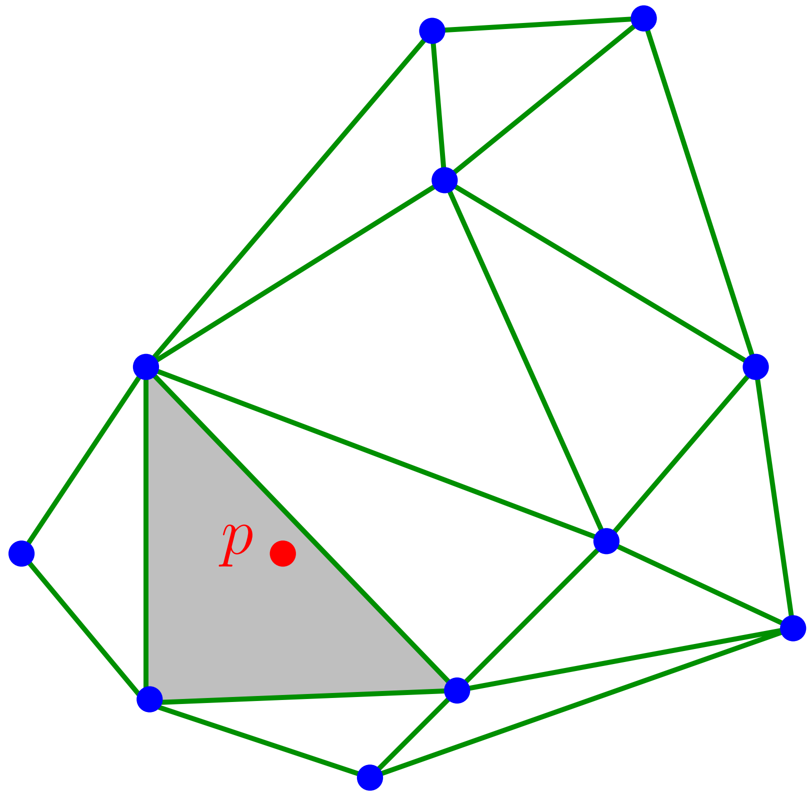
First step



Idea

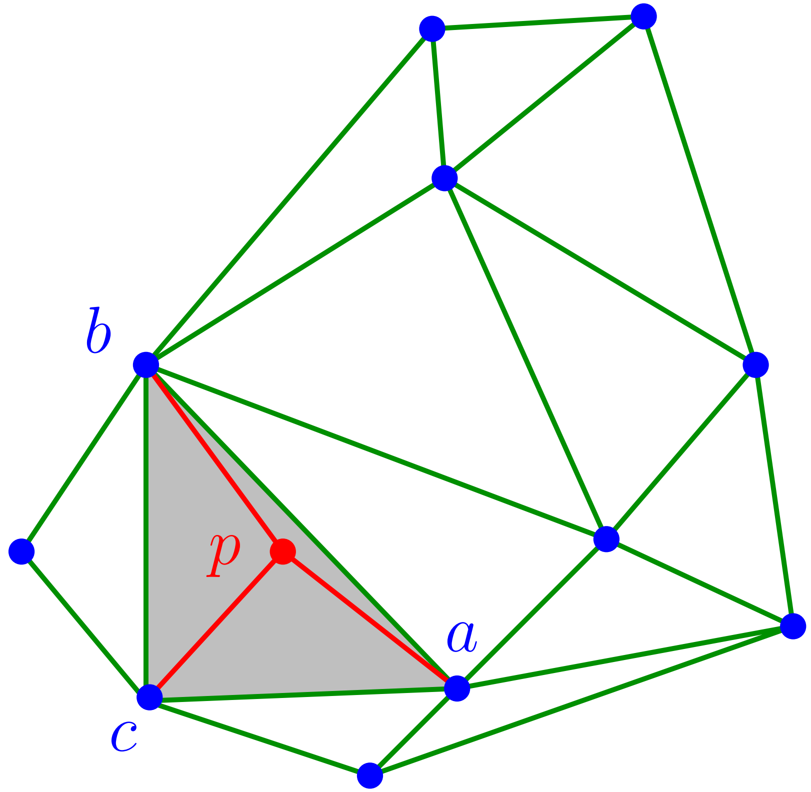
- insert p_1, then p_2 … and finally p_n
- suppose we have computed $\mathcal{DT}(P_{i-1})$
- insert $p_i \Rightarrow$ splits a triangle into three
  - find this triangle using conflict lists
    - each non inserted point has a pointer to the triangle in $\mathcal{DT}(P_{i-1})$ that contains it
    - each triangle in $\mathcal{DT}(P_{i-1})$ is associated with the list of all the non–inserted points that it contains
- perform edge flips until no illegal edge remains
  - we only need to perform flips around p_i
  - on average, this step takes constant time
- we have just computed $\mathcal{DT}(P_i)$
- repeat the process until $i = n$

Example



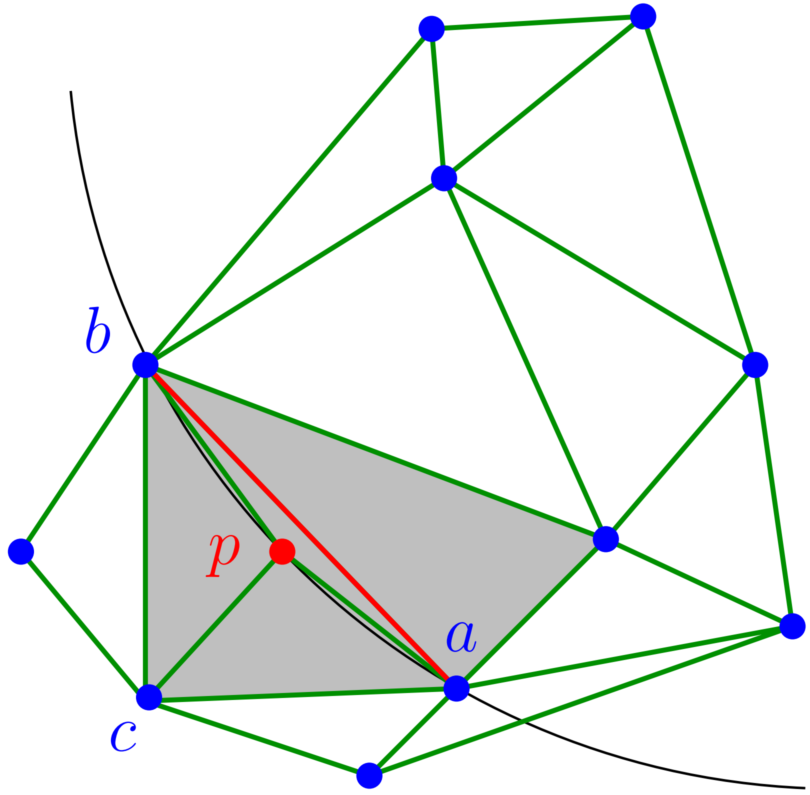
- inserting p_i
- to simplify the notations, we denote $p = p_i$
- we do not draw $p_{-1}p_{-2}p_{-3}$

Example



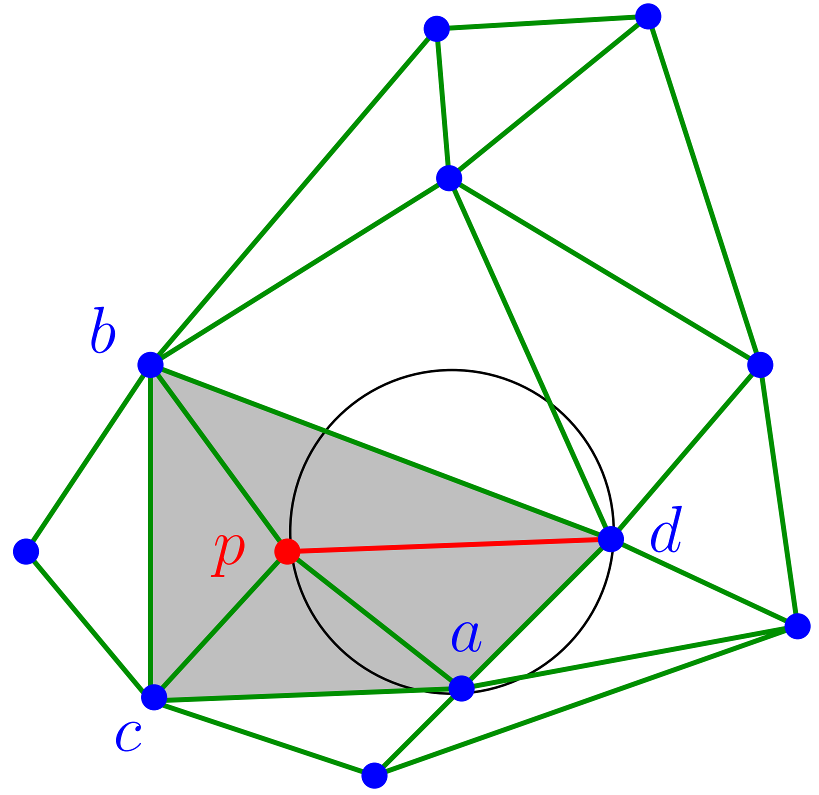
- use the pointer from p to the triangle abc that contains it
- split abc into abp, bcp and cap
- split the conflict list of abc accordingly

Example



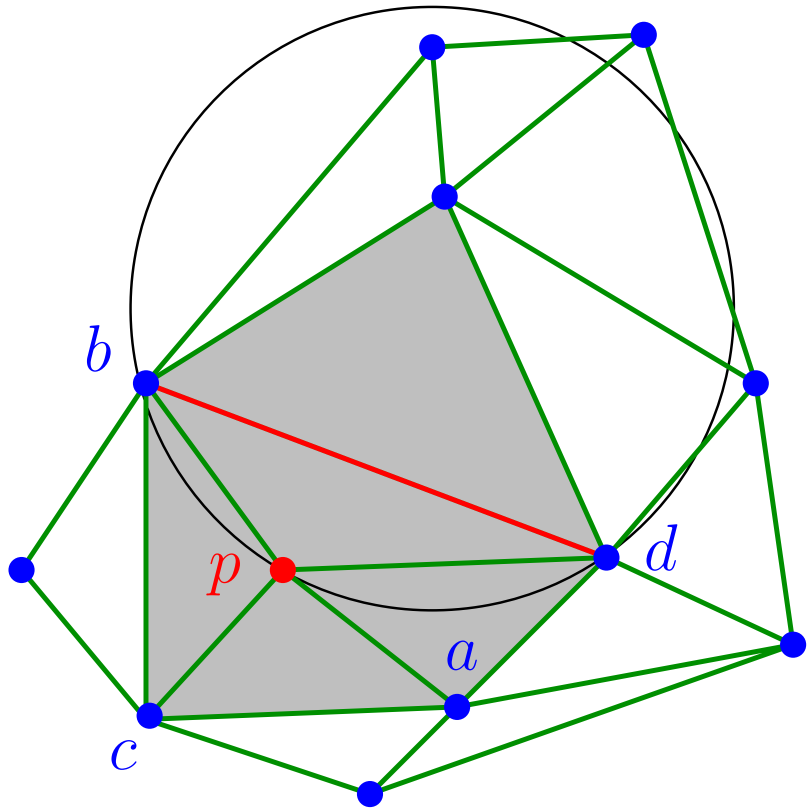
- edge ab is illegal
- flip it

Example



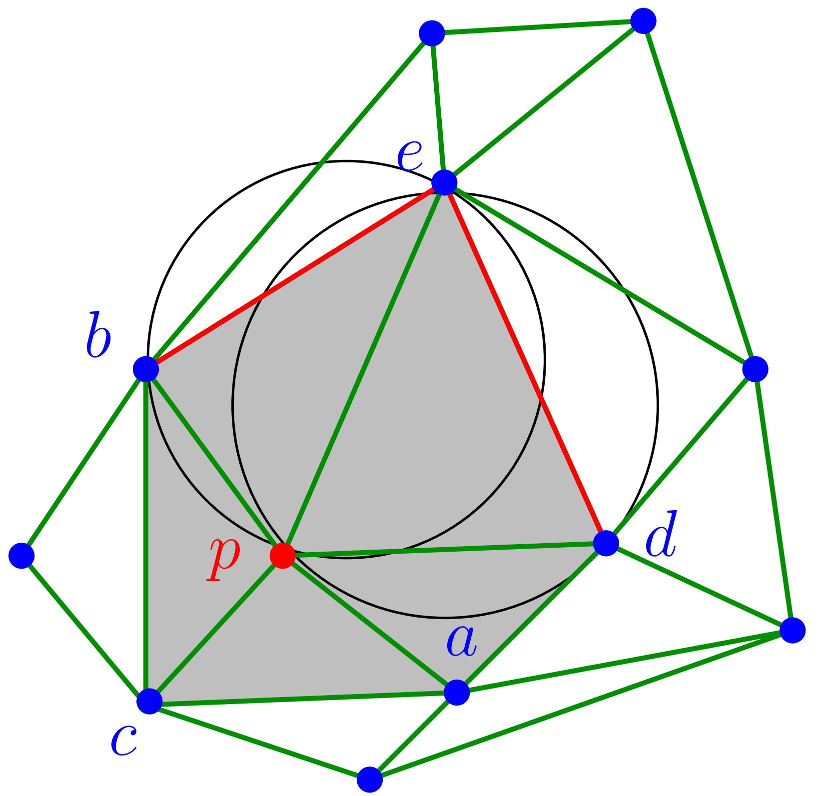
- edge ab has been flipped into pd
- ad is locally Delaunay, we keep it

Example



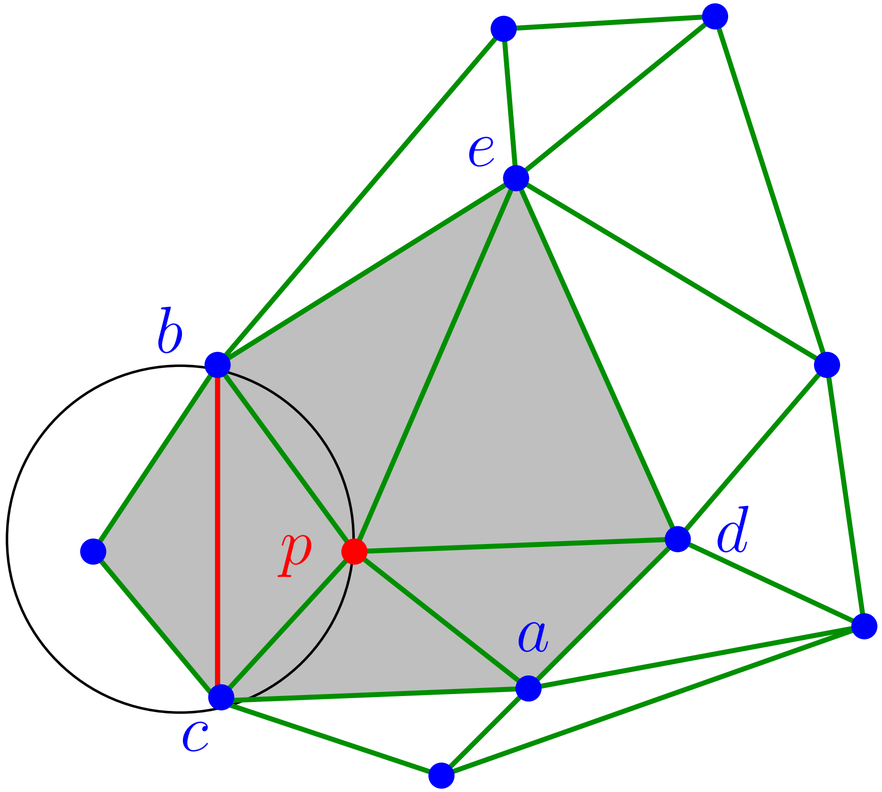
- edge bd is illegal

Example



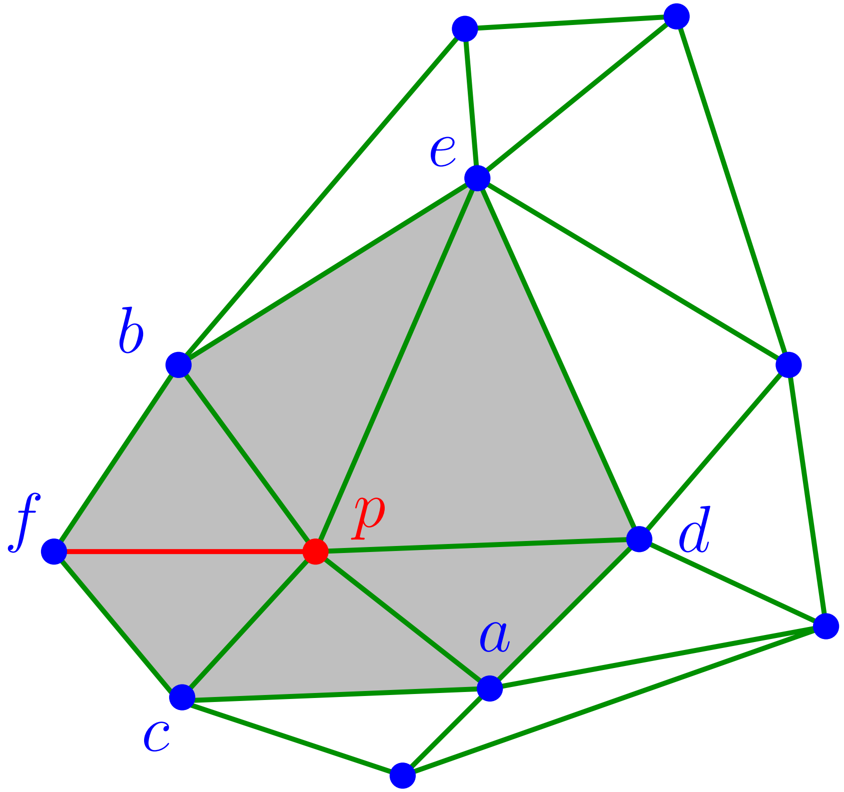
- edge bd has been flipped into pe
- edges de and be are locally Delaunay, we keep them

Example



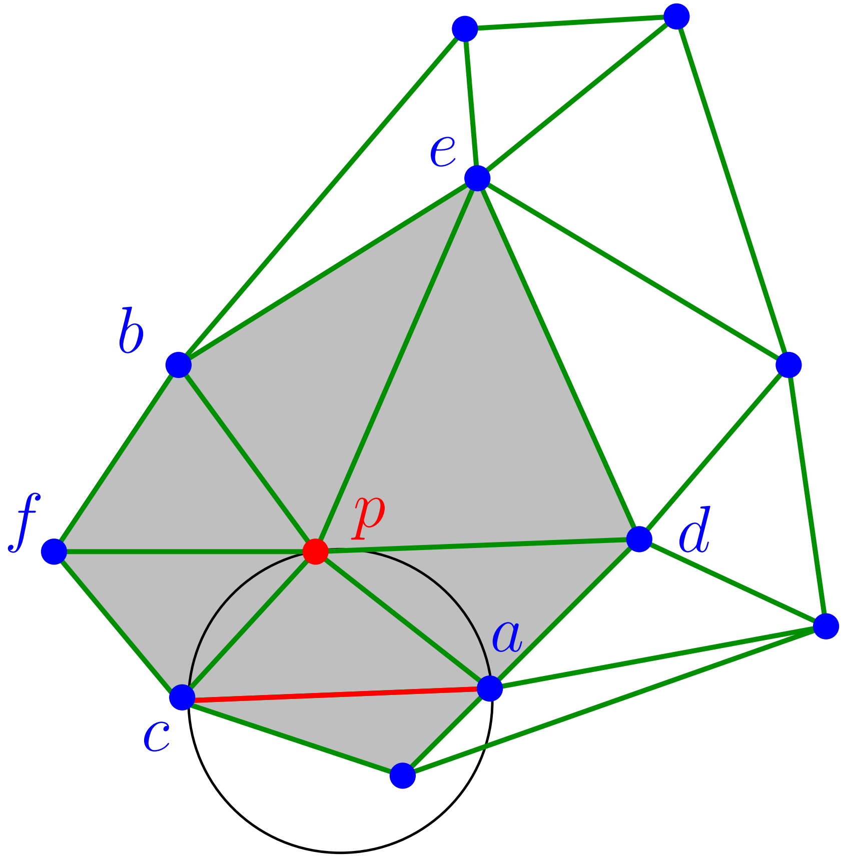
- edge bc is illegal

Example



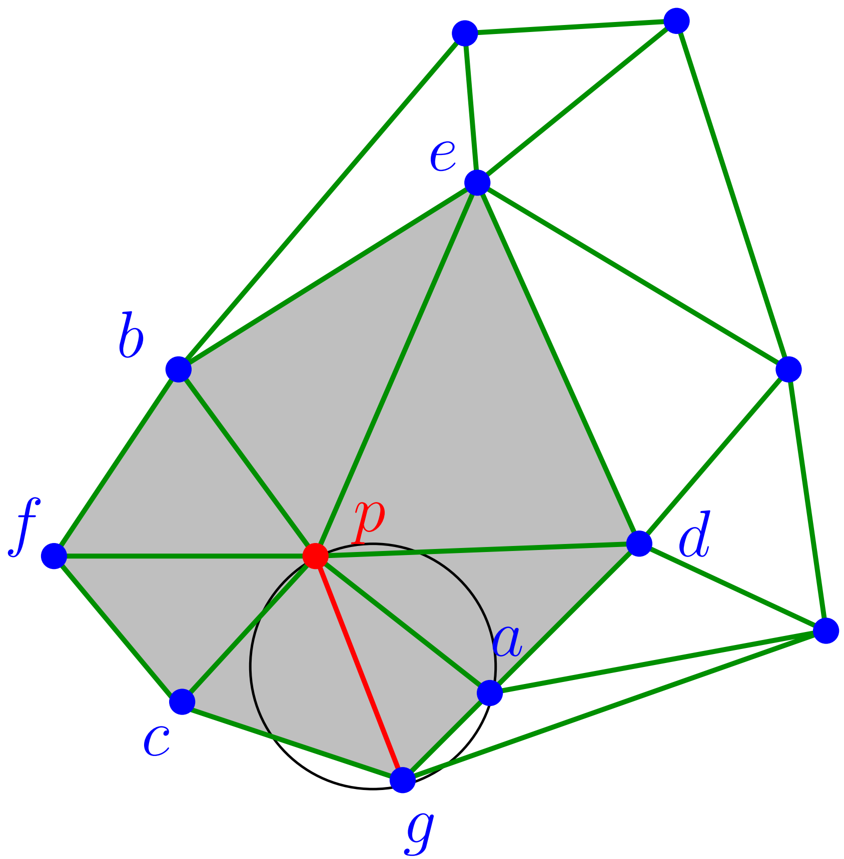
- edge bc has been flipped into pf

Example



- edge ac is illegal

Example



- edge ag is locally Delaunay
- no more edge to flip: we are done

Explanation

- we considered triangles in counterclockwise order around p and flipped illegal edges
- why is it enough to consider only triangles adjacent to p?
- see proof later
- the pseudocode for this algorithm is very simple
- see next slide

Pseudocode

Algorithm *Insert*(p)
Input: a point p, a set of point P and $\mathcal{T} = \mathcal{DT}(p)$
Output: $\mathcal{DT}(P \cup \{p\})$
1. Find the triangle abc of $\mathcal{DT}(P \cup \{p\})$ containing p
($*$ use reverse pointers from conflict lists $*$)
($*$ abc is chosen to be counterclockwise $*$)
2. Insert edges pa,pb and pc
($*$ it includes conflict lists updates $*$)
3. $SwapTest(ab)$
($*$ pseudocode of this procedure next slide $*$)
4. $SwapTest(bc)$
5. $SwapTest(ca)$

Pseudocode

Algorithm *SwapTest(ab)*

1. **if** ab is an edge of the exterior face
2. **do return**
3. $d \leftarrow$the vertex to the right of edge ab
4. **if** $\mathrm{inCircle}(p, a, b, d) < 0$
5. **do** Flip edge ab for pd

($*$ it includes conflict lists update $*$)

6. $SwapTest(ad)$
7. $SwapTest(db)$

Proof

- we only flipped edges of triangles that contain p
- why is it sufficient?
- remember Theorem slide 26: locally Delaunay implies Delaunay
- any edge between two triangles that do not contain p was locally Delaunay before insertion of p
- so it is still locally Delaunay
- thus the triangulation we obtain is the Delaunay triangulation

Analysis

- we look at t_i: time taken to update the current triangulation while inserting p_i
- it does not account for conflict lists updates
- each new edge (after splitting abc or after a flip) contains p_i
- so t_i is proportional to the degree of p_i in $\mathcal{DT}(p_i)$
  - degree of p_i: number of edges that contain p_i

Analysis

- we use backward analysis: P_i is fixed, p_i is random
- each edge has two vertices
- so each edge contains p_i with probability $\frac{2}{i}$
- there are $O(i)$ edges in the whole triangulation (by Euler formula)
- so by backward analysis

$$E[t_i] = \frac{O(i)}{i} = O(1)$$

- so the time for updating the triangulation is $O(n)$ over the course of the whole algorithm
- similar with trapezoidal map: what takes $\Theta(n \log n)$ time is the update of conflict lists (see next slide)

Analysis

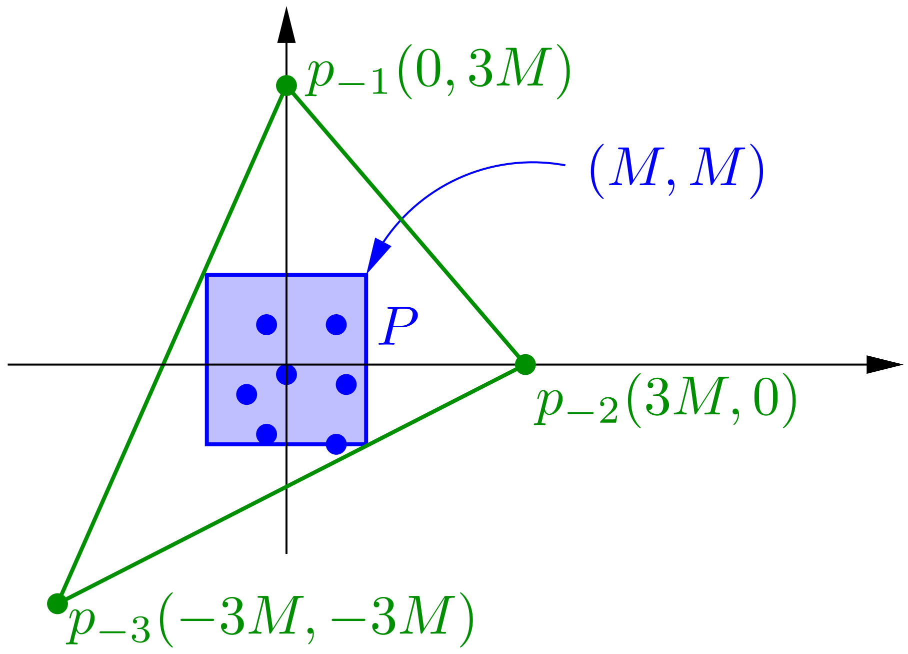
- while inserting p_i, what is the probability that $p \in P \setminus P_i$ is rebucketed?
- backward analysis
  - assume p is in triangle abc
  - this is the probability that $p_i \in \{a, b, c\}$
  - so it is $3/i$
- so while inserting p_i, we rebucket less than $3n/i$ sites on average

Analysis

- problem: a site may be rebucketed several times at step i
  - intuition: only a constant number of flips at each step so it only account for a constant factor
  - detailed proof in textbook
- so overall, rebucketing takes expected time

$$O\left(\sum_{i=1}^{n} \frac{n}{i}\right) = O(n \log n)$$

Choice of $p_{-1}p_{-2}p_{-3}$



- M: max of any coordinate of any point in P
- For incircle test, do as if these three points are outside any circle defined by three points in P

Conclusion

Conclusion

- the Delaunay triangulation of n points can be computed in expected time $O(n \log n)$
- it holds for worst case input, the expectation is over the random choices made by the algorithm
- it can also be done in $O(n \log n)$ deterministic time
- knowing the Delaunay triangulation of P, we can find the Voronoi diagram of P in $O(n)$ time
  - left as an exercise

Conclusion

- combining with the point location data structure of lecture 8, we can answer proximity queries in the plane (see lecture 9) with
  - $O(\log n)$ expected query time
  - $O(n \log n)$ expected preprocessing time
  - $O(n)$ expected space usage
- all these bounds can be made deterministic
  - harder, less practical