Delaunay Triangulations

image thanks to wikipedia and youtube

Computational Geometry [csci 3250]
Laura Toma
Bowdoin College

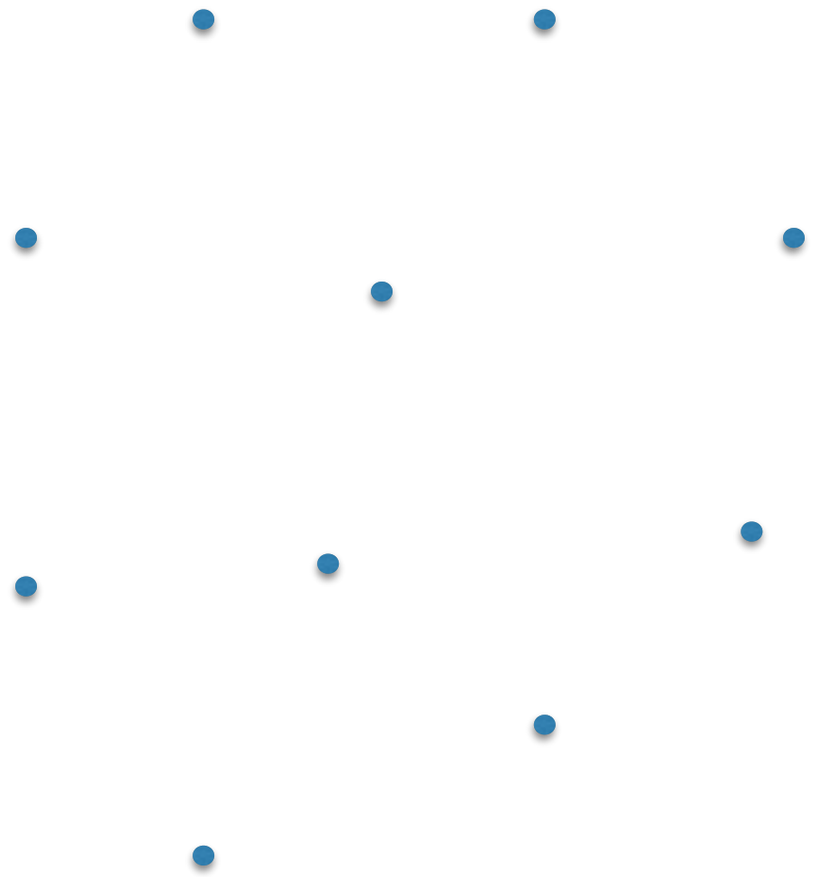
Outline

- Triangulating a set of points
- The Delaunay triangulation
  - Definition and properties
  - Construction via edge flipping
  - RIC
  - Delaunay triangulation and 3d hull
  - Applications

Triangulating a set of points

Let P = {p_1, p_2,…,p_n} a set of points in the plane.

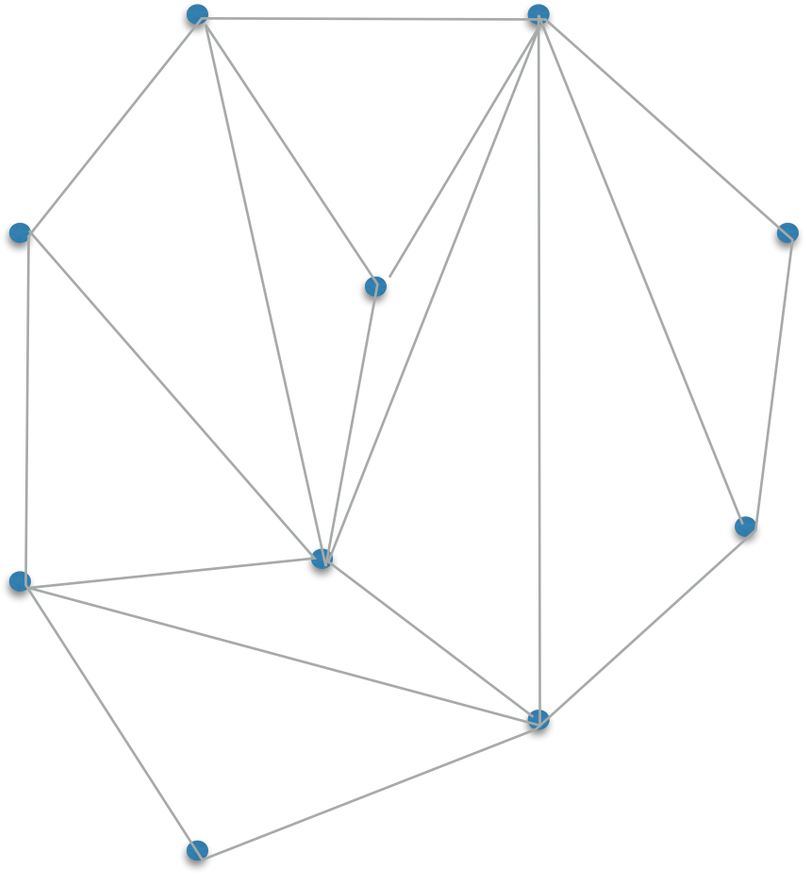
A triangulation T(P) is a planar subdivision whose vertices are P and all faces are triangles (except the unbounded face, which is bounded by the hull)



Triangulating a set of points

Let $P = \{p_1, p_2, \ldots, p_n\}$ a set of points in the plane.

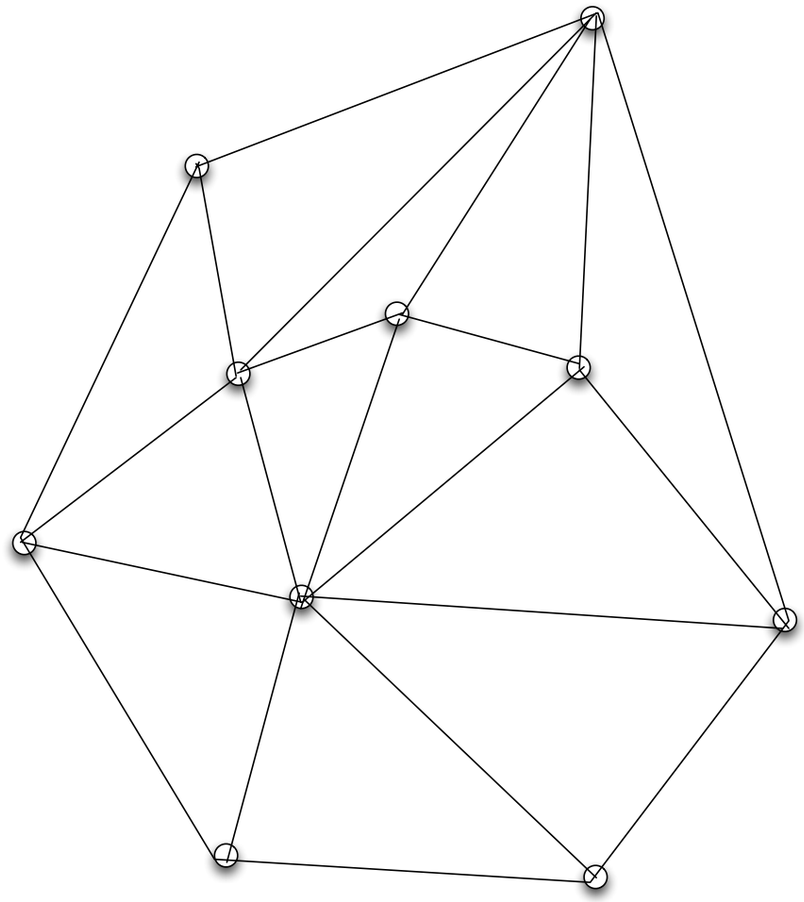
A triangulation T(P) is a planar subdivision whose vertices are P and all faces are triangles (except the unbounded face, which is bounded by the hull)

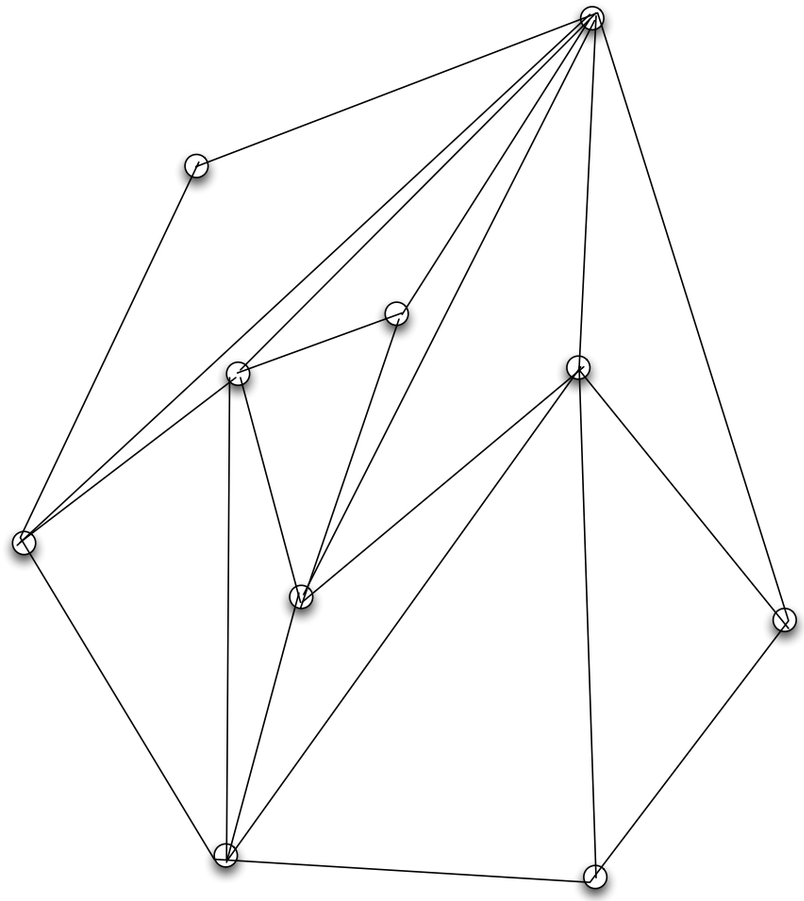


A possible triangulation

Triangulating a set of points

- A point set has many possible triangulations





Claim: All triangulations of P have the same number of edges and triangles.

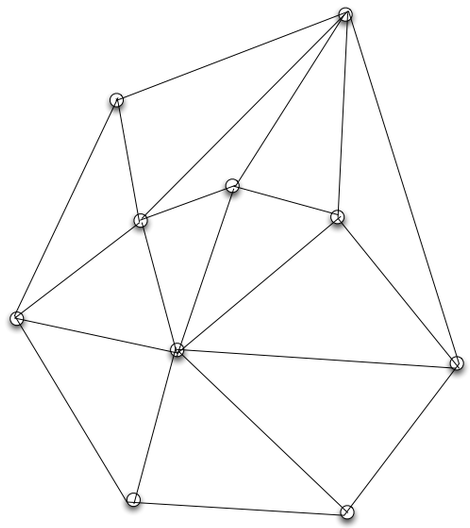
Size of a triangulation

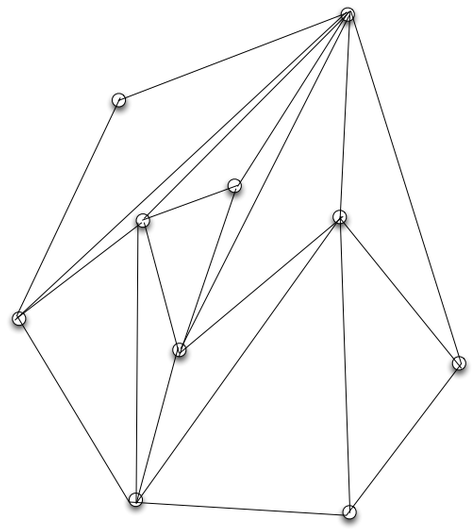
- Let P = {p_1, p_2,…,p_n} be a set of points in the plane
- Let k = number of points in P that lie on CH(P).

Theorem: Any triangulation of P has 2n - 2 - k triangles and 3n - 3 - k edges.

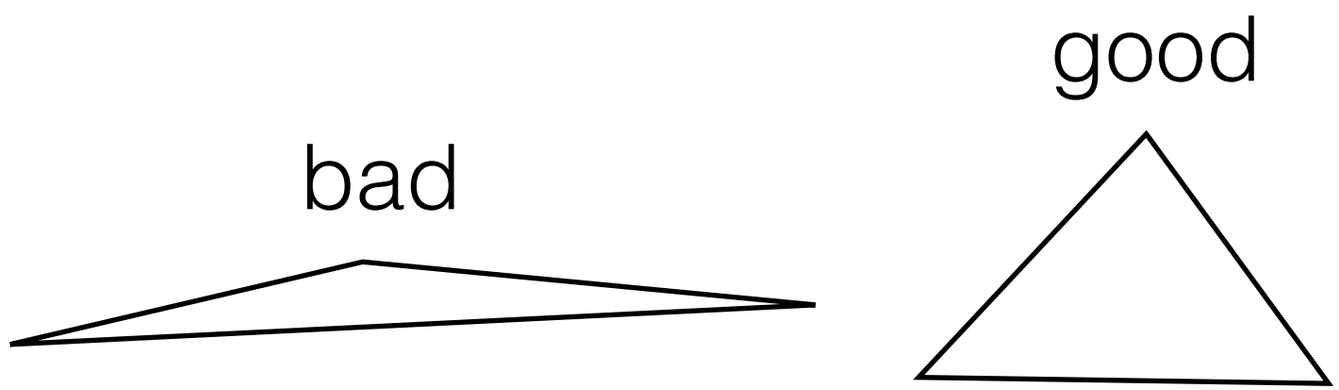
Proof: Use Euler's formula: v - e + f = 2

Which one ?

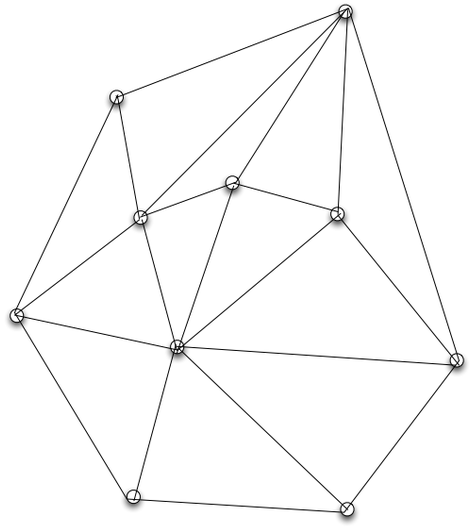


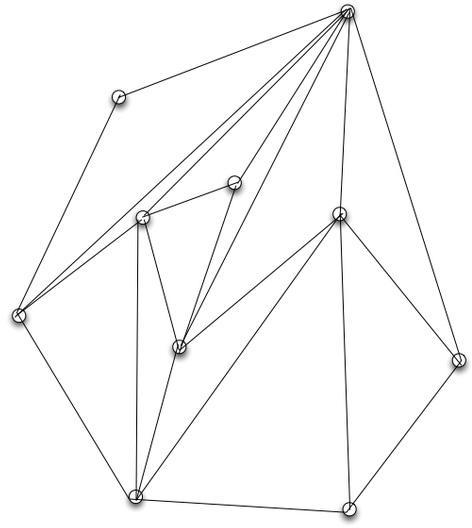


- In practice it is desirable to use triangulations that do not have small angles
  - Small angles introduce numerical issues with geometric predicates such as LeftOf(), also with computing derivatives, curvature, etc; this causes issues in modeling



Which one ?

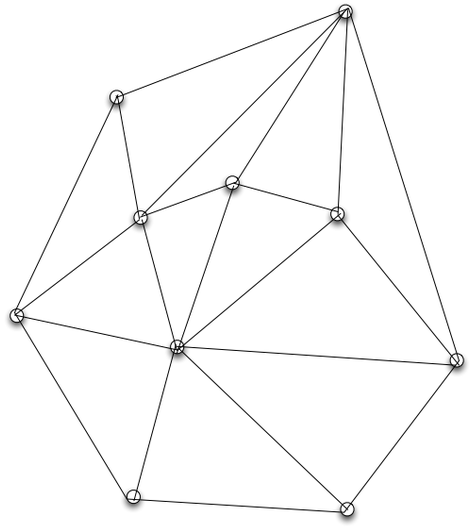


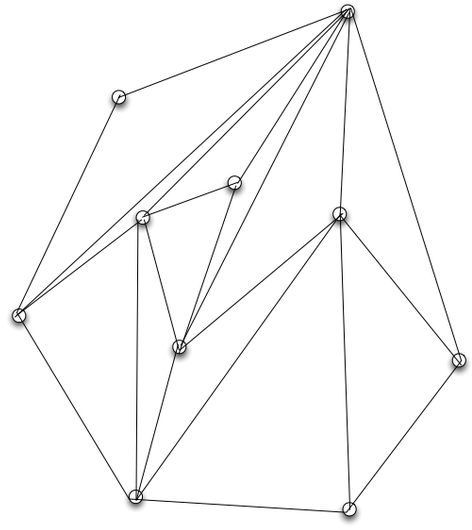


clearly this is better, but..

- ..is there an even better triangulation?
- ..how to define the “angle goodness” of a triangulation?
  - its minimum angle? how many small angles? …
  - which one is better: <5 80 95> or <10 10 160>
- ..how do we compute one?

Which one ?



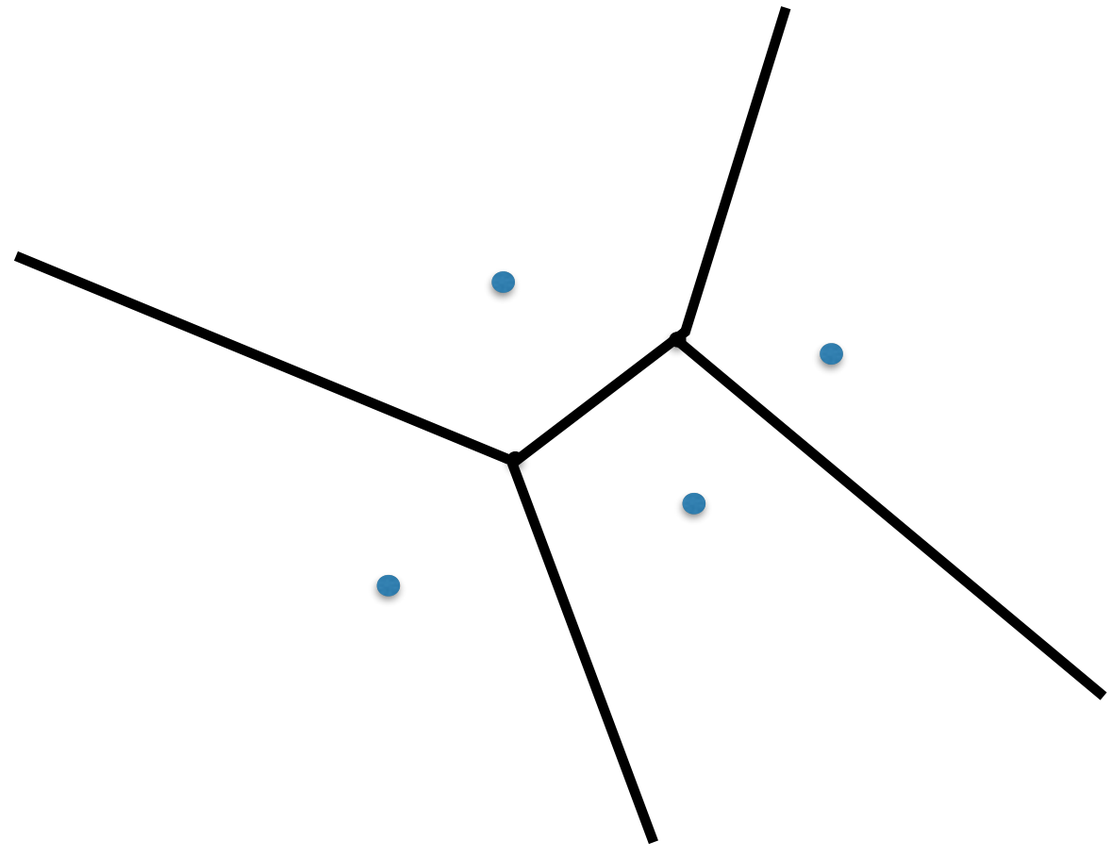


Delaunay triangulation

- Of all possible triangulations of a point set P, the triangulation that maximizes the minimum angle is the Delaunay triangulation
- Delaunay triangulation is the default triangulation used in practice, and has many applications and elegant properties
- It is defined via Voronoi diagram

Voronoi Diagram

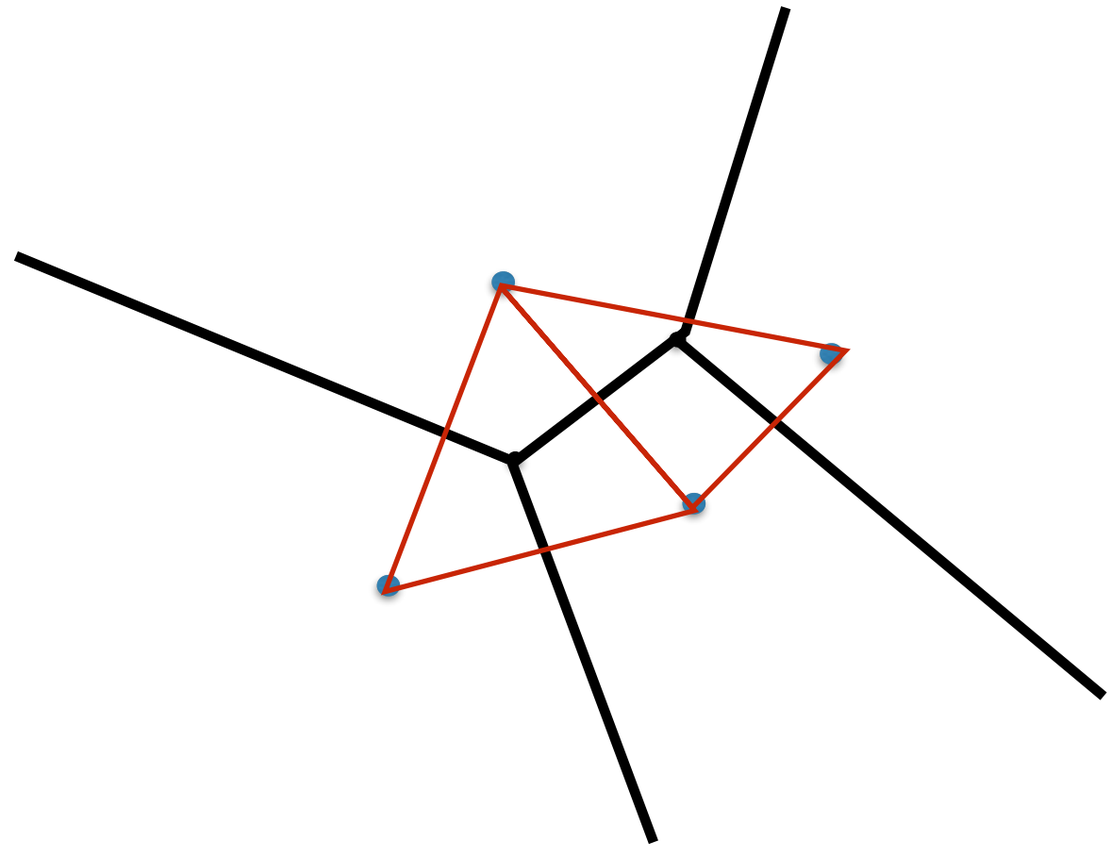
Let P = {p_1, p_2,…,p_n} set of points in the plane such that no 4 are co-circular.



The straight-line dual of Vor(P) : For every pair of sites such that Vor(u) and Vor(v) that share an edge, draw an edge between u and v in the dual

Voronoi Diagram

Let P = {p_1, p_2,…,p_n} set of points in the plane such that no 4 are co-circular.

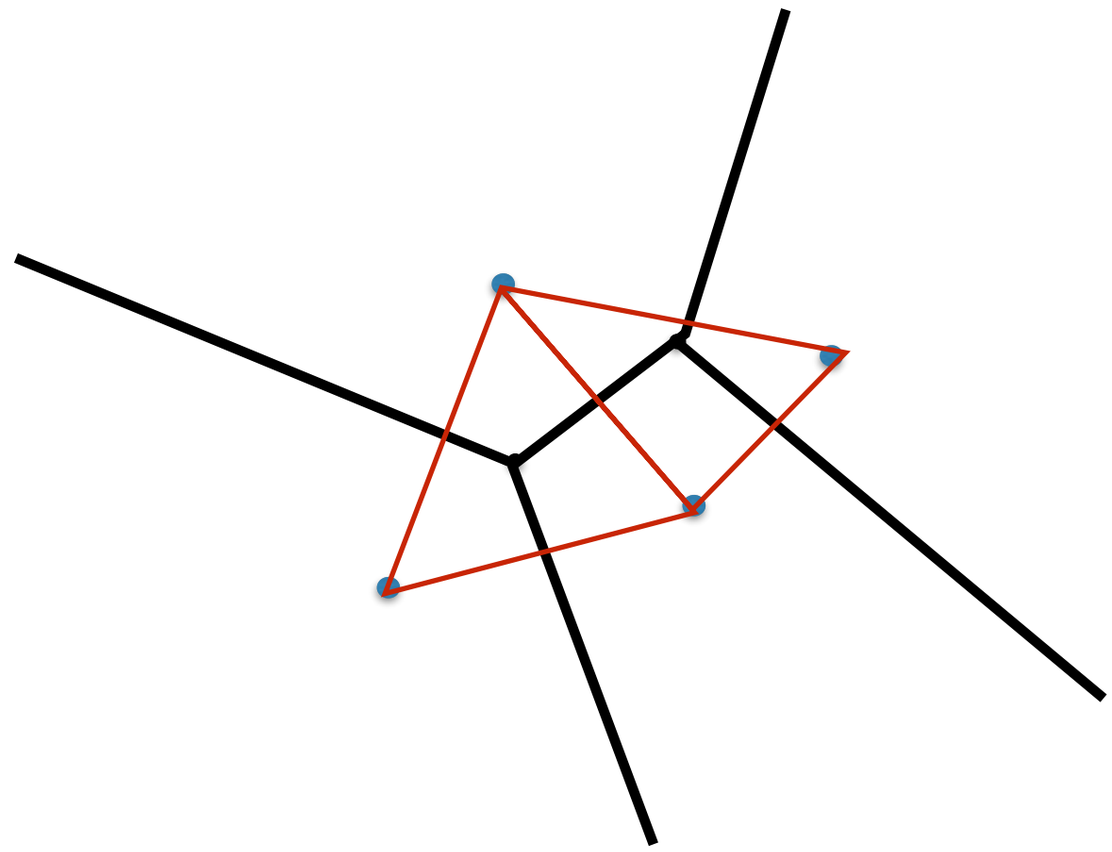


The straight-line dual of Vor(P) : For every pair of sites such that Vor(u) and Vor(v) that share an edge, draw an edge between u and v in the dual

Voronoi Diagram

Let P = {p_1, p_2,…,p_n} set of points in the plane such that no 4 are co-circular.

Each Voronoi vertex has degree 3 ==> it defines a triangle



The straight-line dual of Vor(P) : For every pair of sites such that Vor(u) and Vor(v) that share an edge, draw an edge between u and v in the dual

Voronoi Diagram

Let P = {p

Each Voronoi vertex has degree 3 ==> it defines a triangle

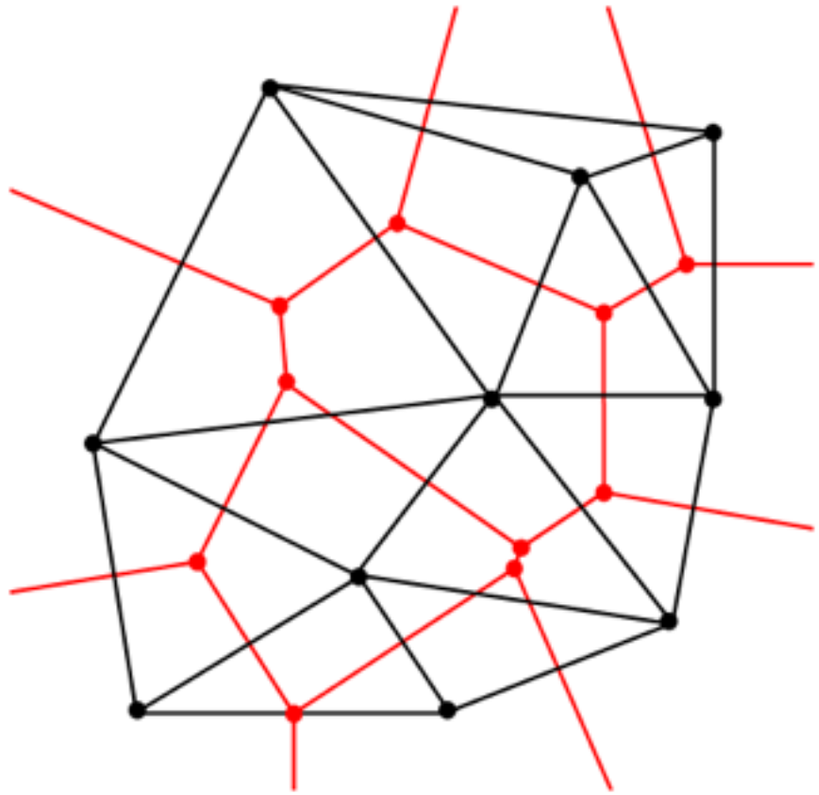
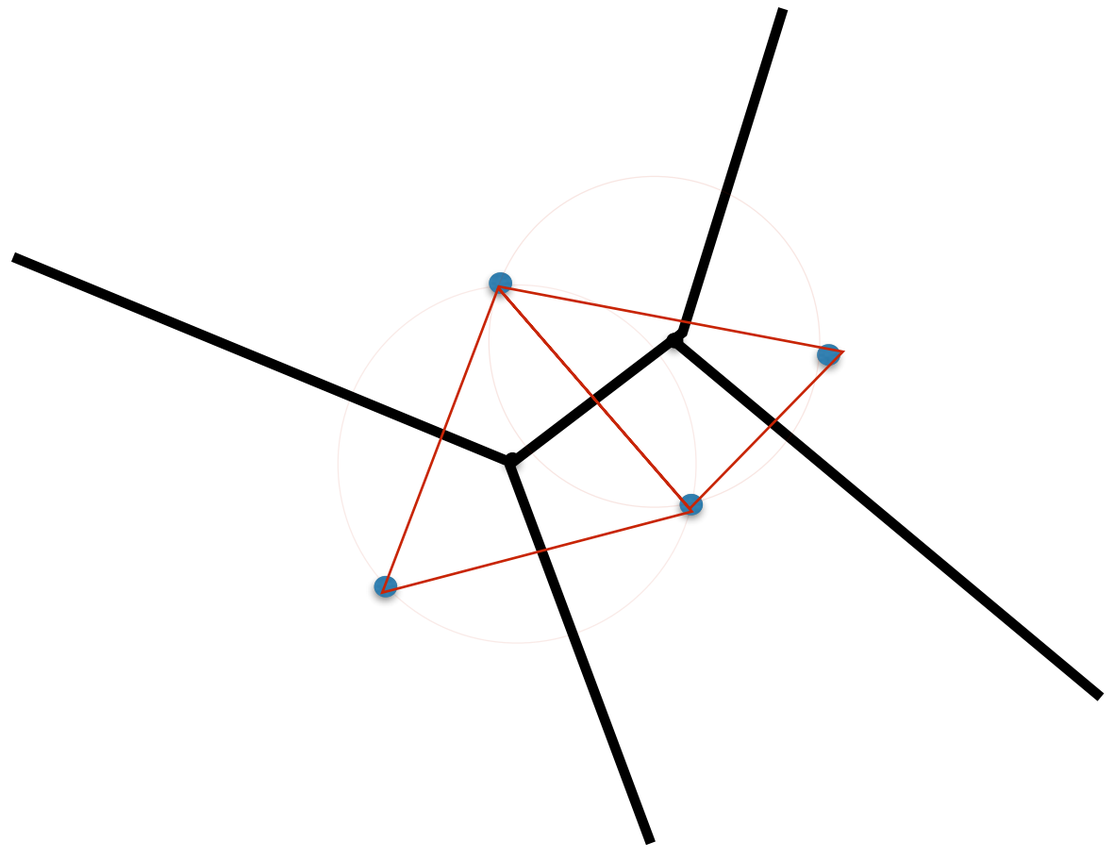


image thanks to wikipedia

Note: the Voronoi vertex isnot necessarily inside its corresponding triangle

Theorem [Delaunay 1934]:

The straight-line dual of Vor(P) is planar, and is a triangulation.



Clearly every Voronoi vertex => triangle. Planarity not trivial to show.

The Delaunay Triangulation

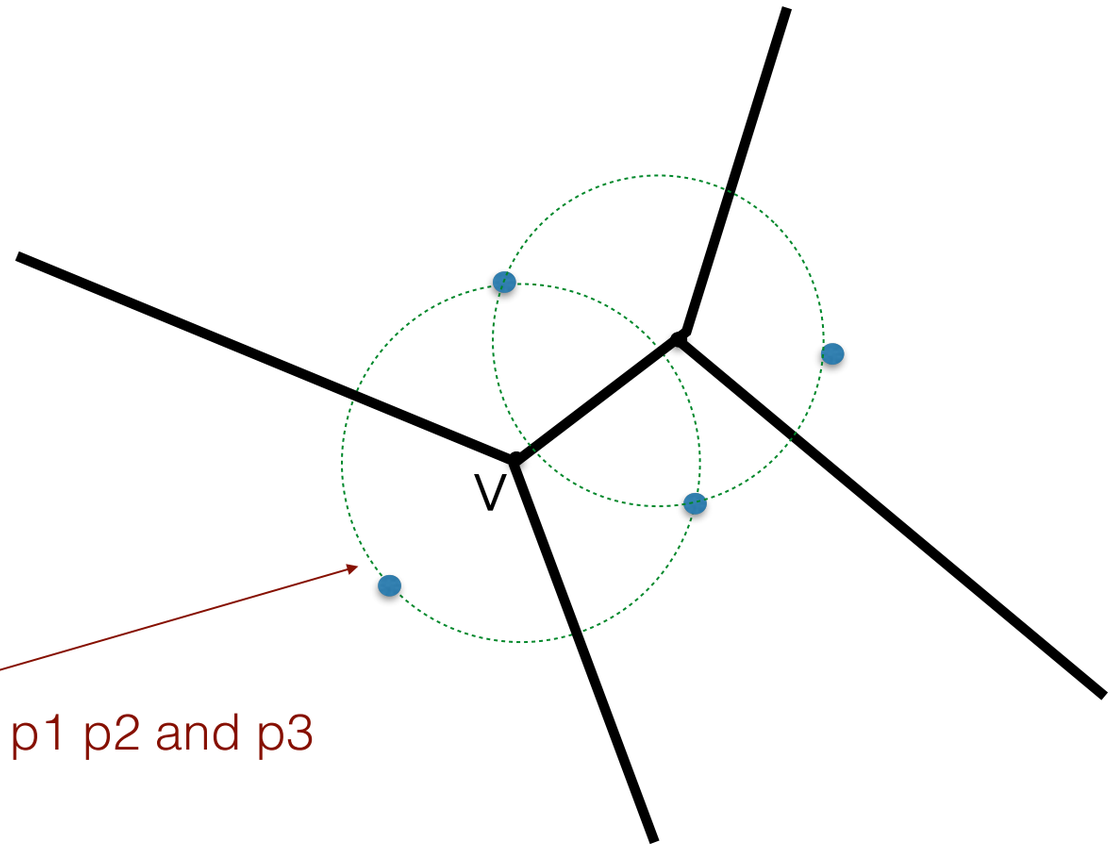
- By definition, DT(P) is the dual of Vor(P)

- We can construct DT(P) from Vor(P)
  - Use for e.g. Fortune's algorithm which runs in O(n lg n)
  - Size of Vor(P) is O(n); its can be obtained in O(n) time

- Would be nice to have a direct way to characterize DT(P)

Let P = {p_1, p_2,…,p_n} set of points in the plane such that no 4 are co-circular.

Empty circle property: Every Voronoi vertex is the center of a circle that has 3 sites on its boundary and no other site inside

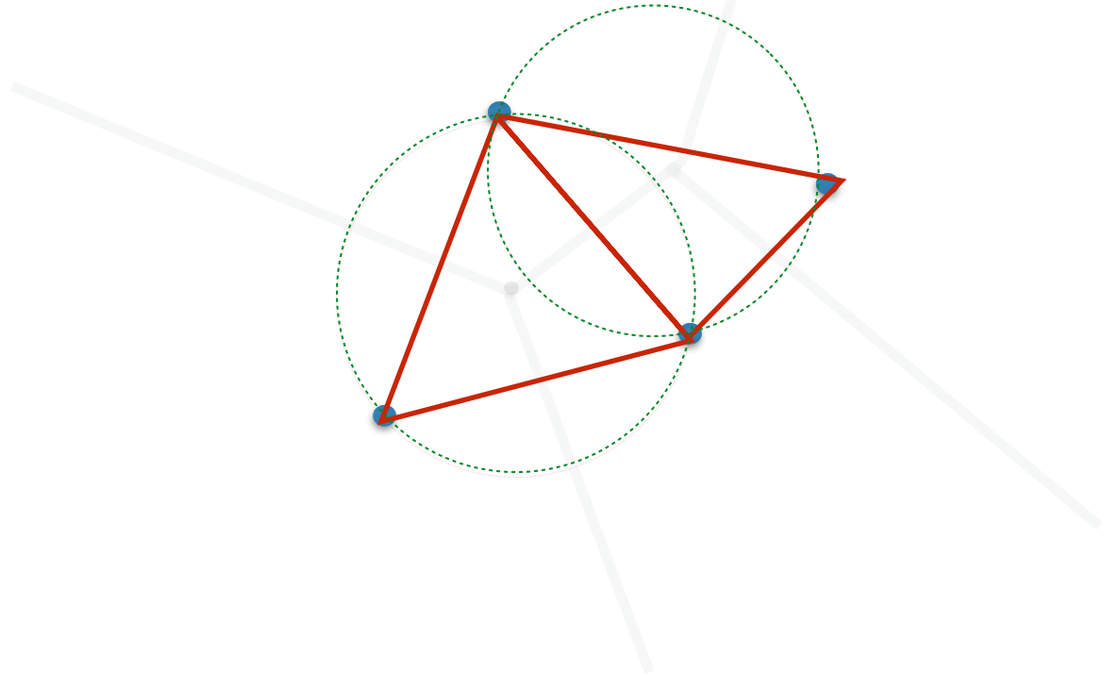


C(v): circle through v, p1 p2 and p3

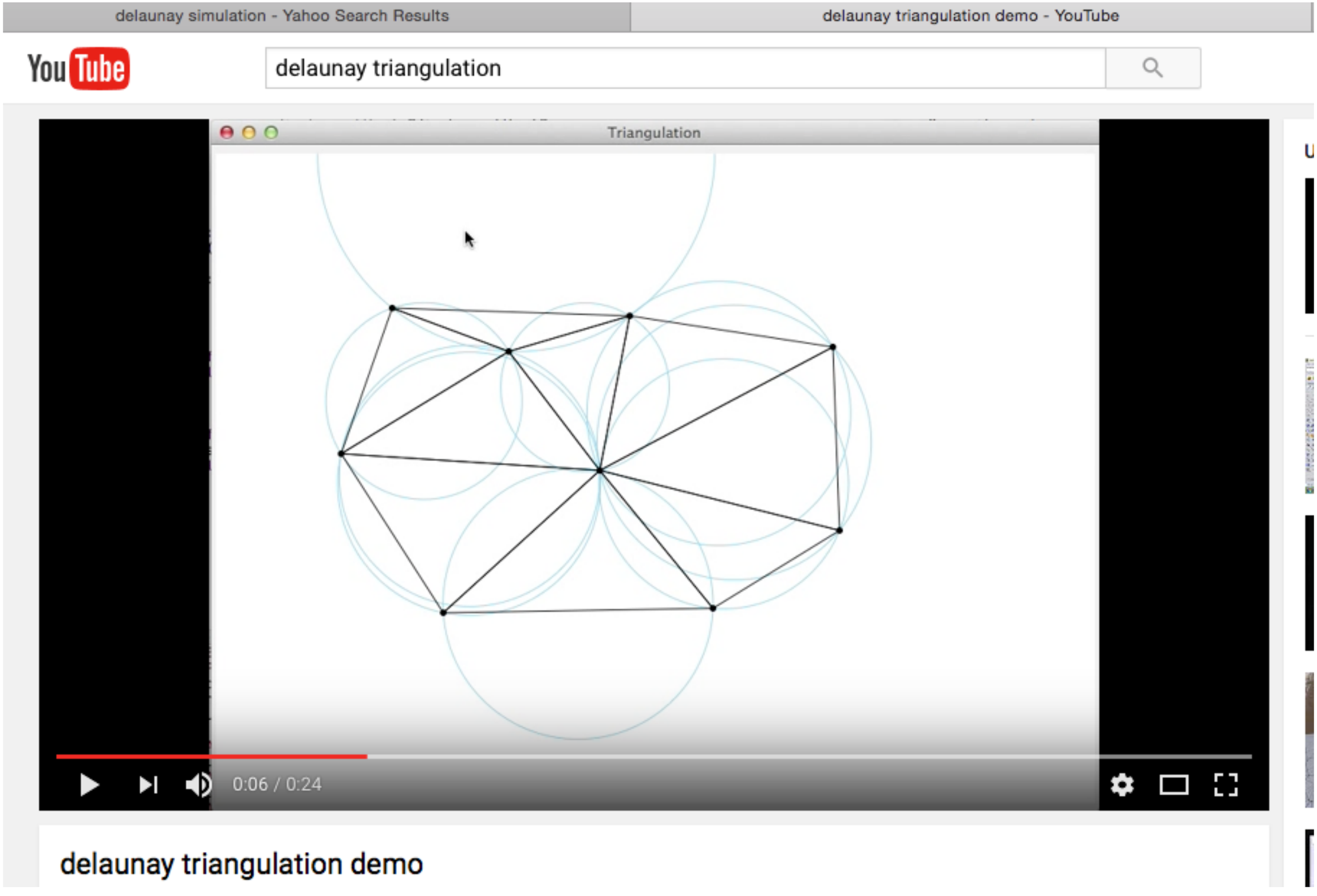
Let P = {p_1, p_2,…,p_n} set of points in the plane such that no 4 are co-circular.

Empty circle property: Every Voronoi vertex is the center of a circle that has 3 sites on its boundary and no other site inside

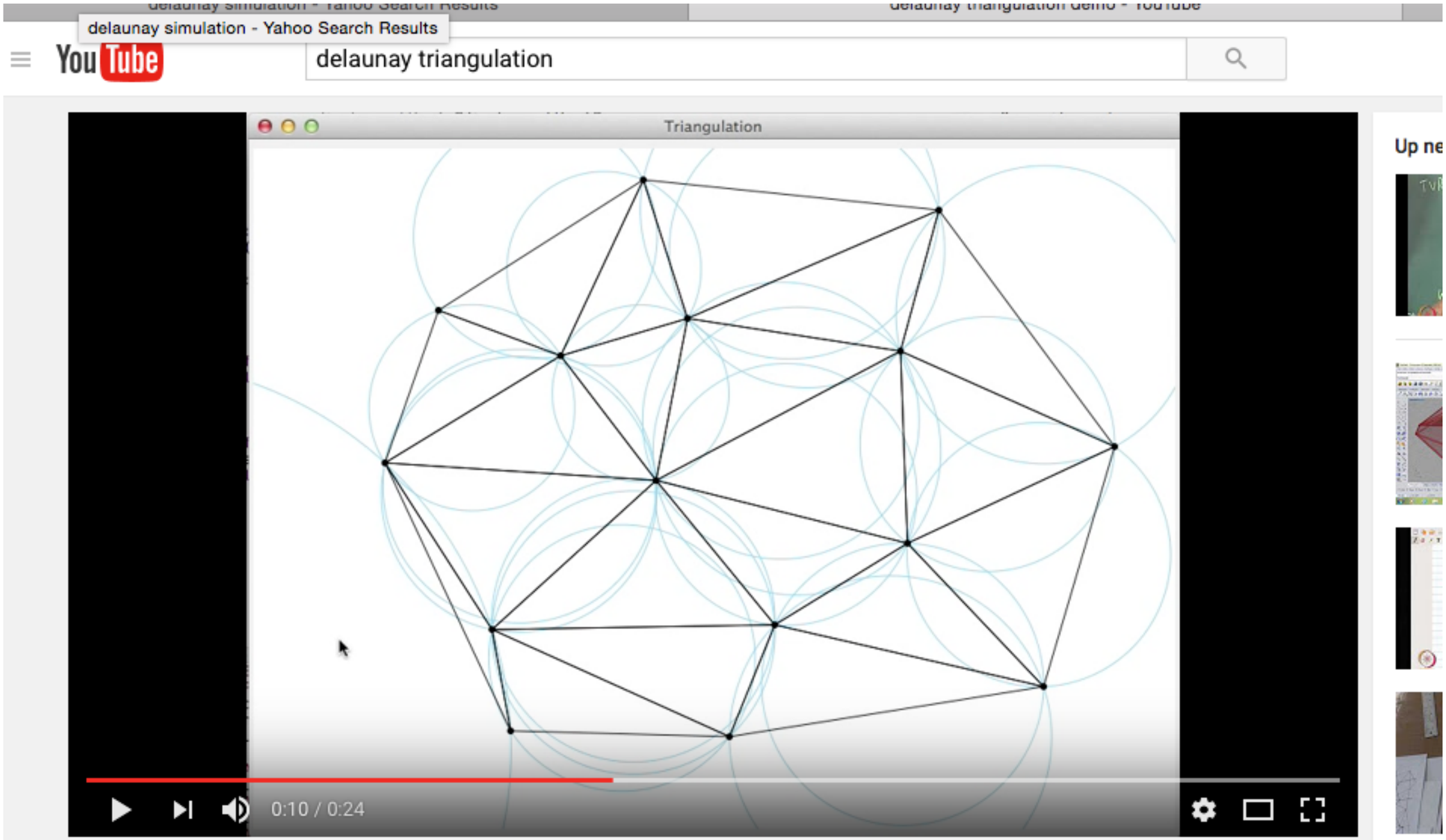
=> For every triangle in DT(P), its circumcircle is empty



For every triangle in DT(P), its circumcircle is empty



For every triangle in DT(P), its circumcircle is empty



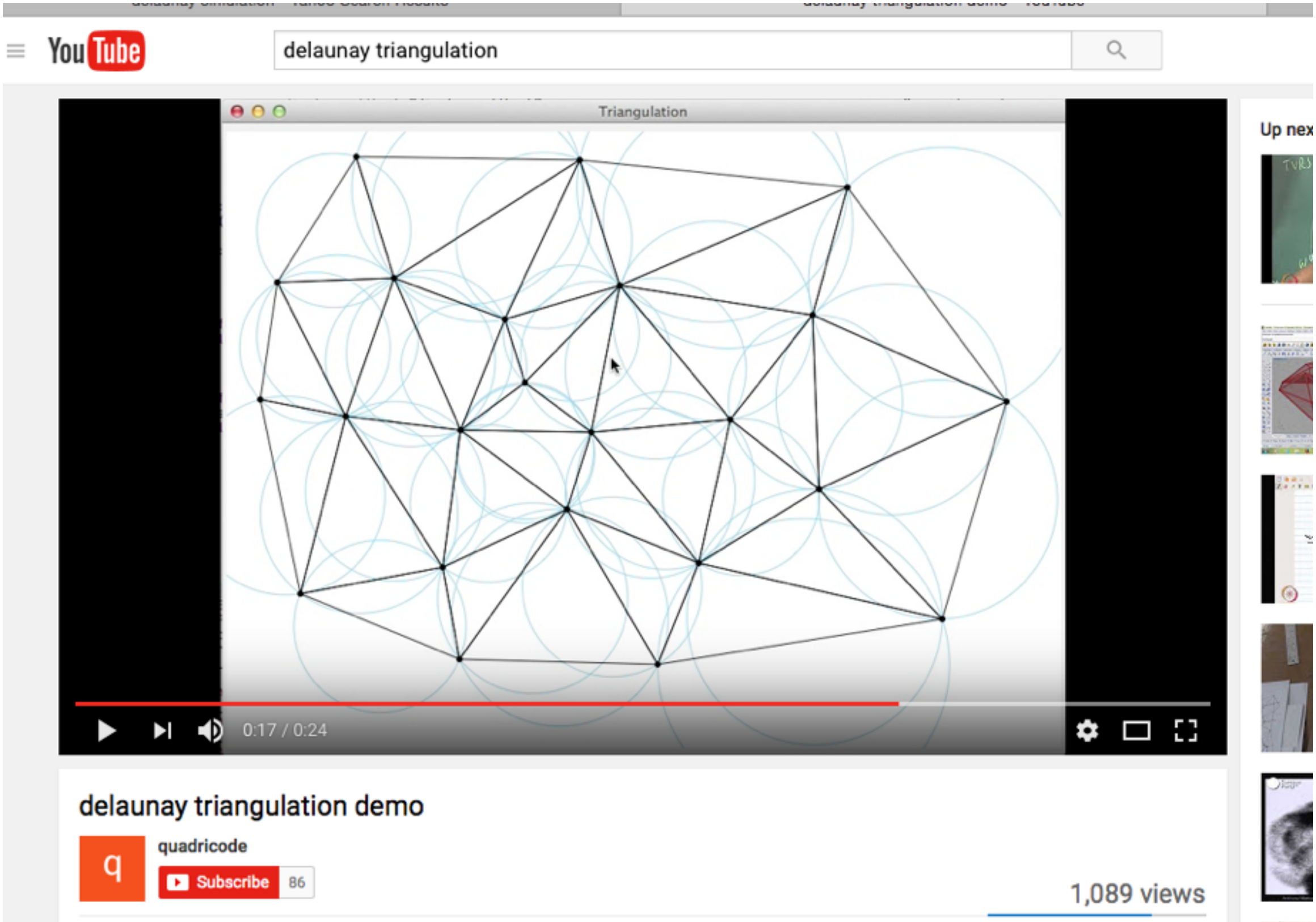
delaunay triangulation demo

quadricode

Subscribe 86

1,089 views

For every triangle in DT(P), its circumcircle is empty



Delaunay Triangulation

Lemma: For every triangle in DT(P), its circumcircle is empty

- It was shown that this is true the other way around:

Lemma: If all triangles in a triangulation have their circumcircles empty, then the triangulation is the DT

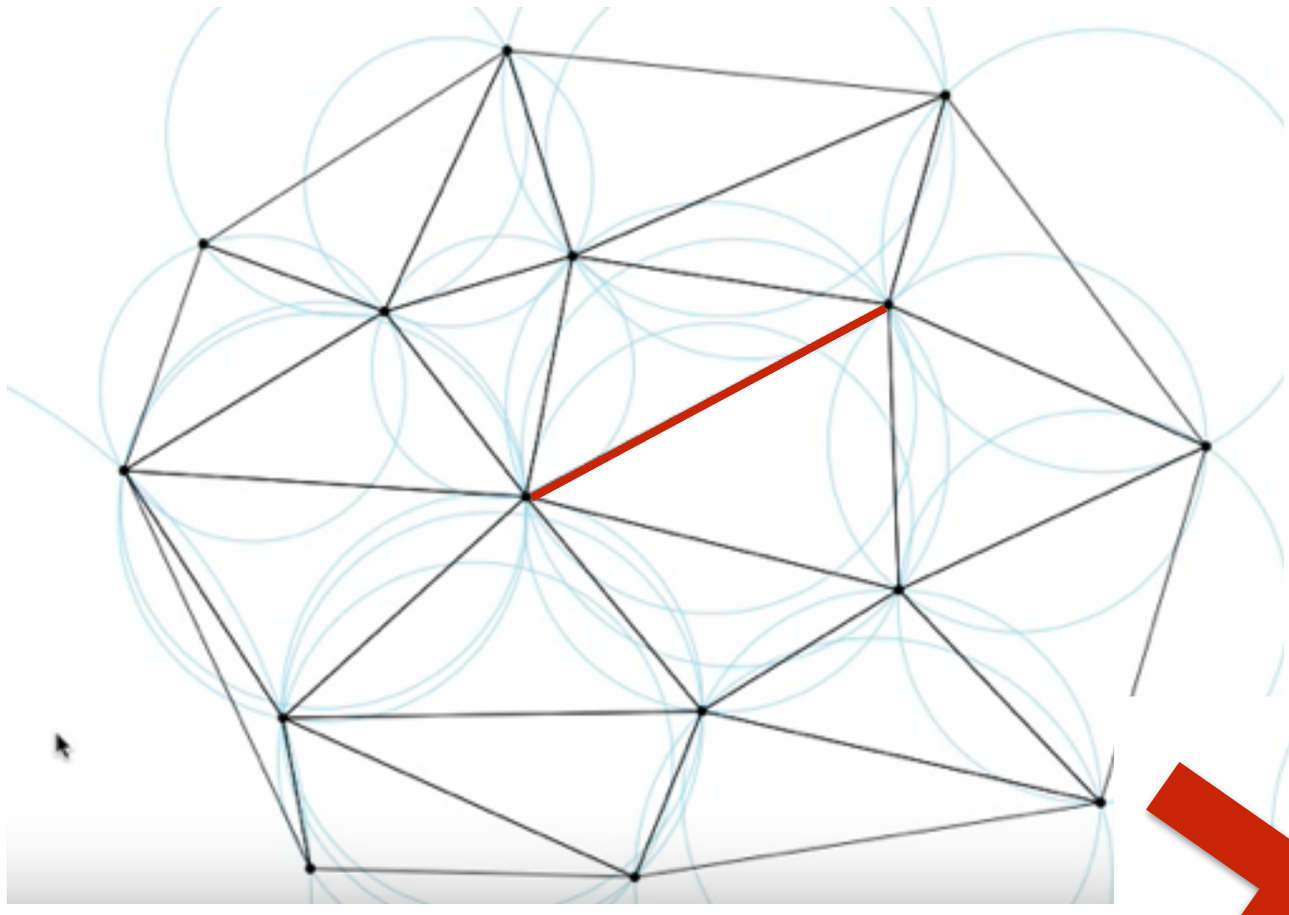
- Therefore, if we could build a triangulation such that all triangles have their circles empty, then this is the DT

Flipping an edge

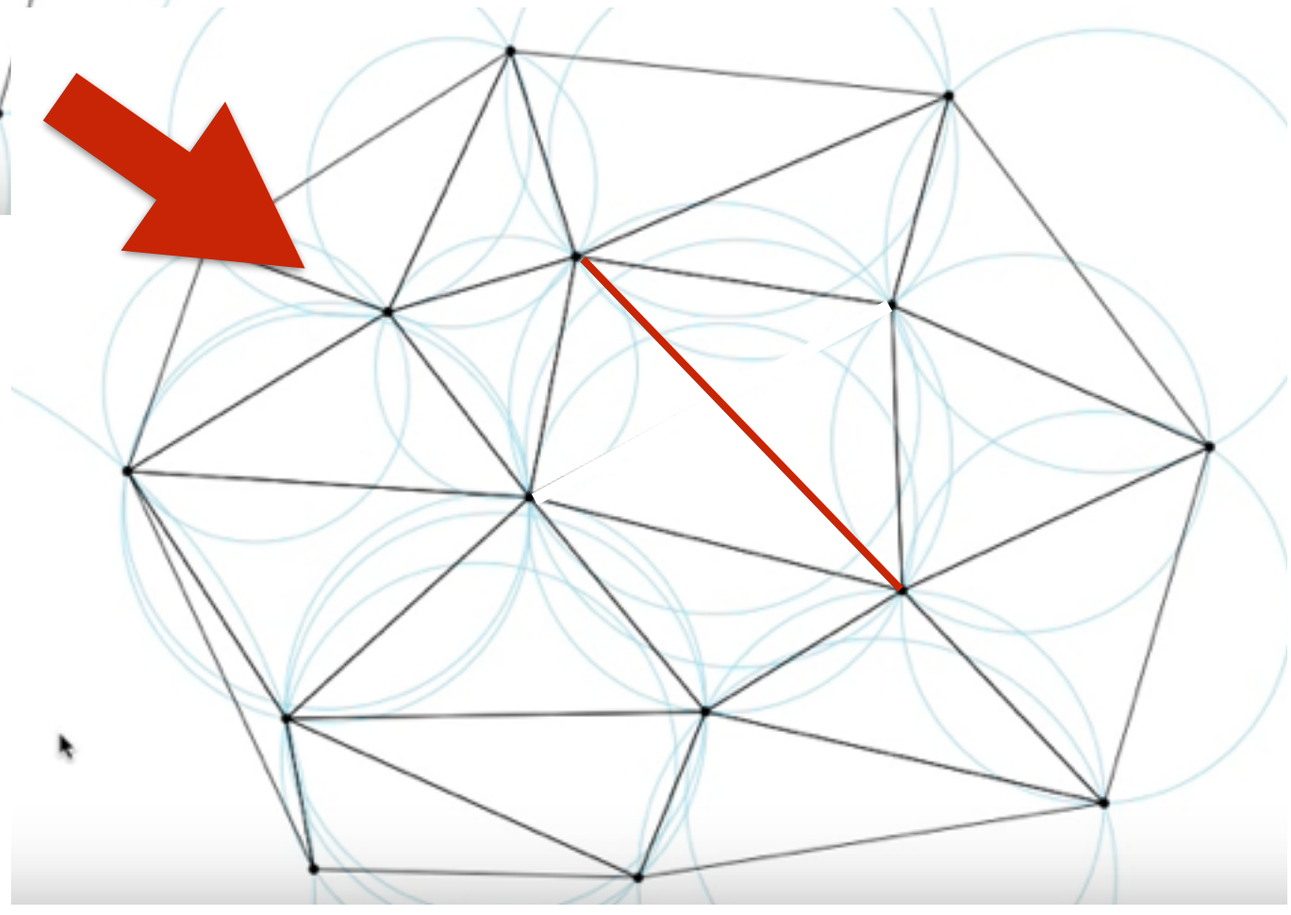
Delaunay

Let’s see what happens if we flip an edge in a Delaunay triangulation

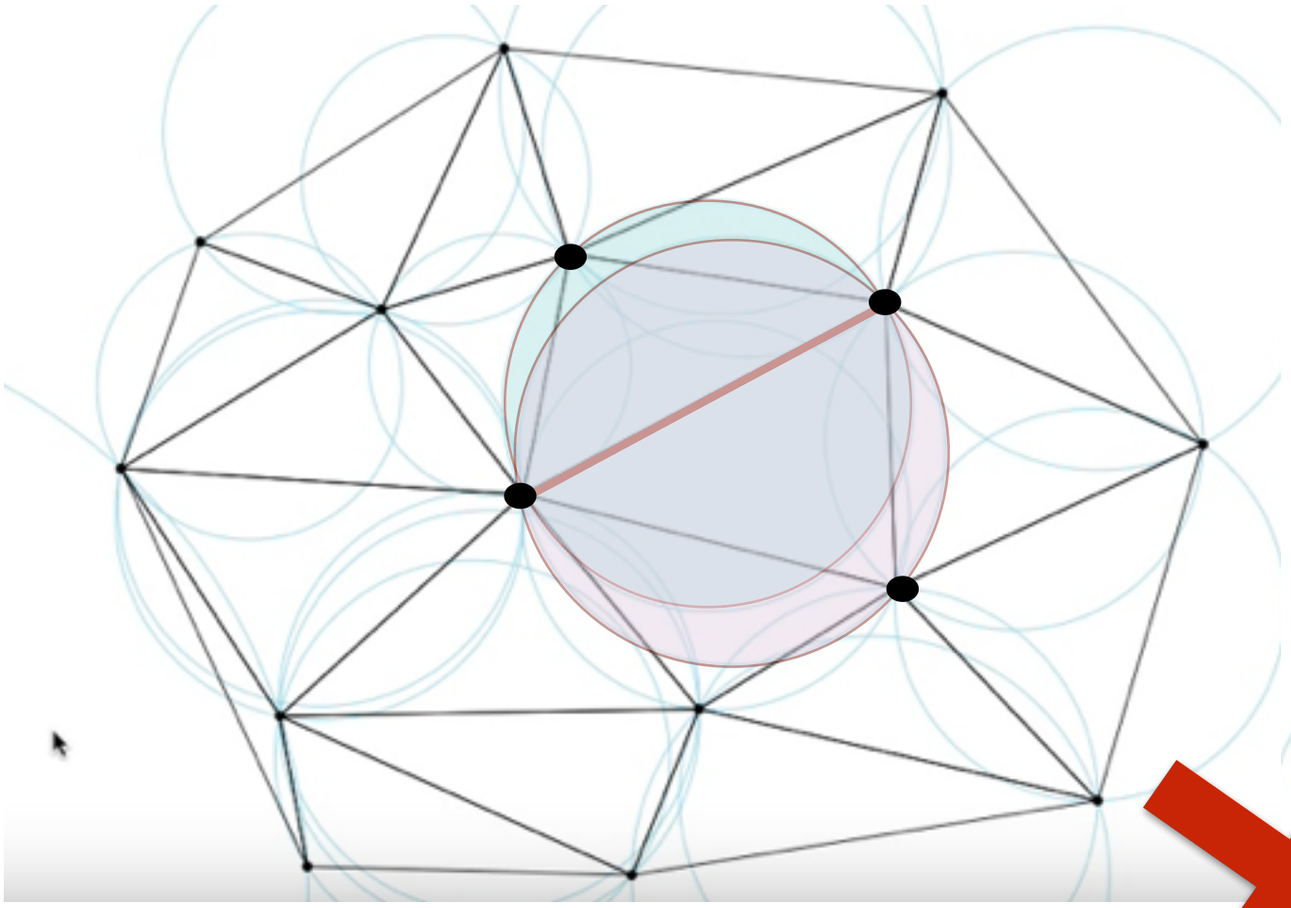
Flipping an edge



Delaunay

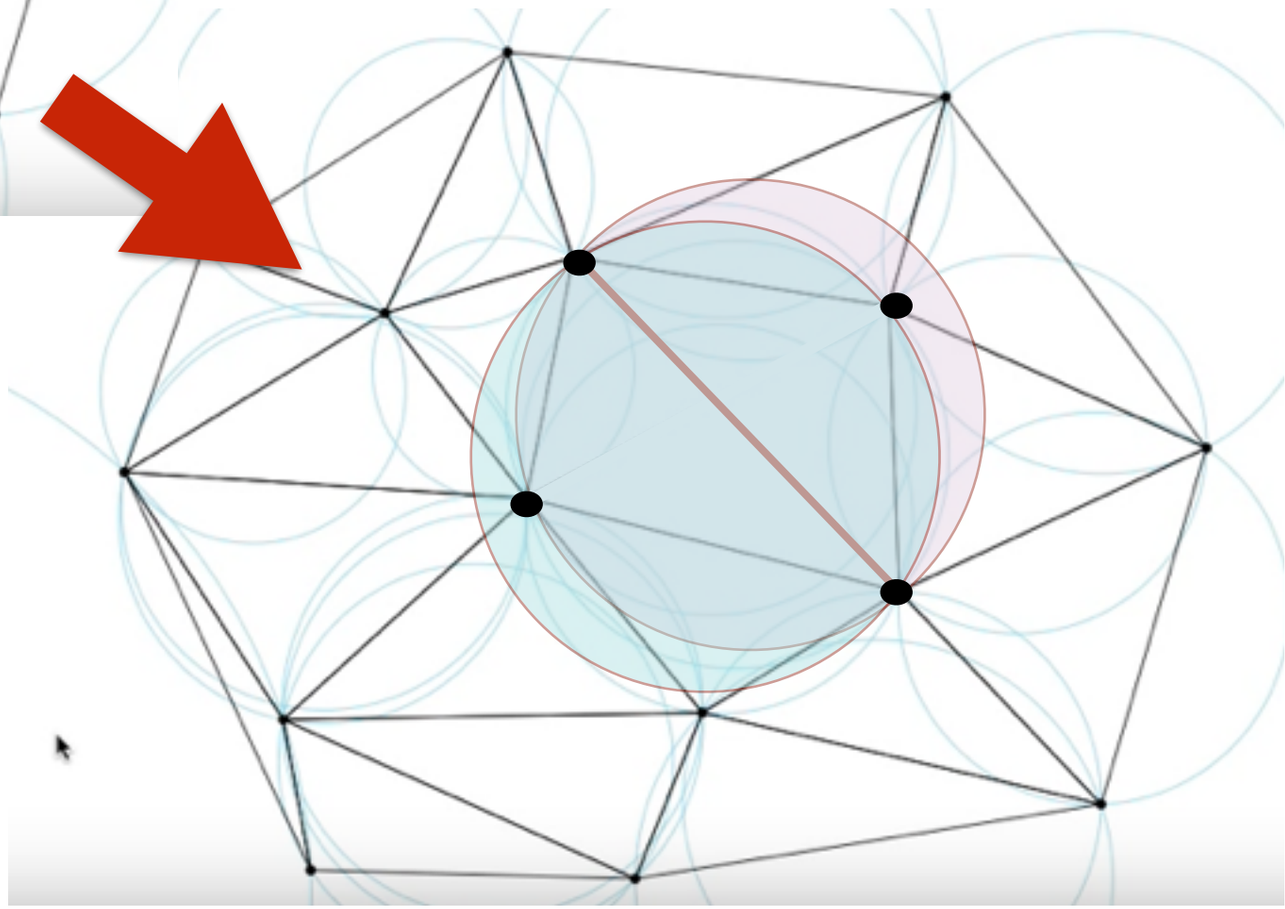


Flipping an edge



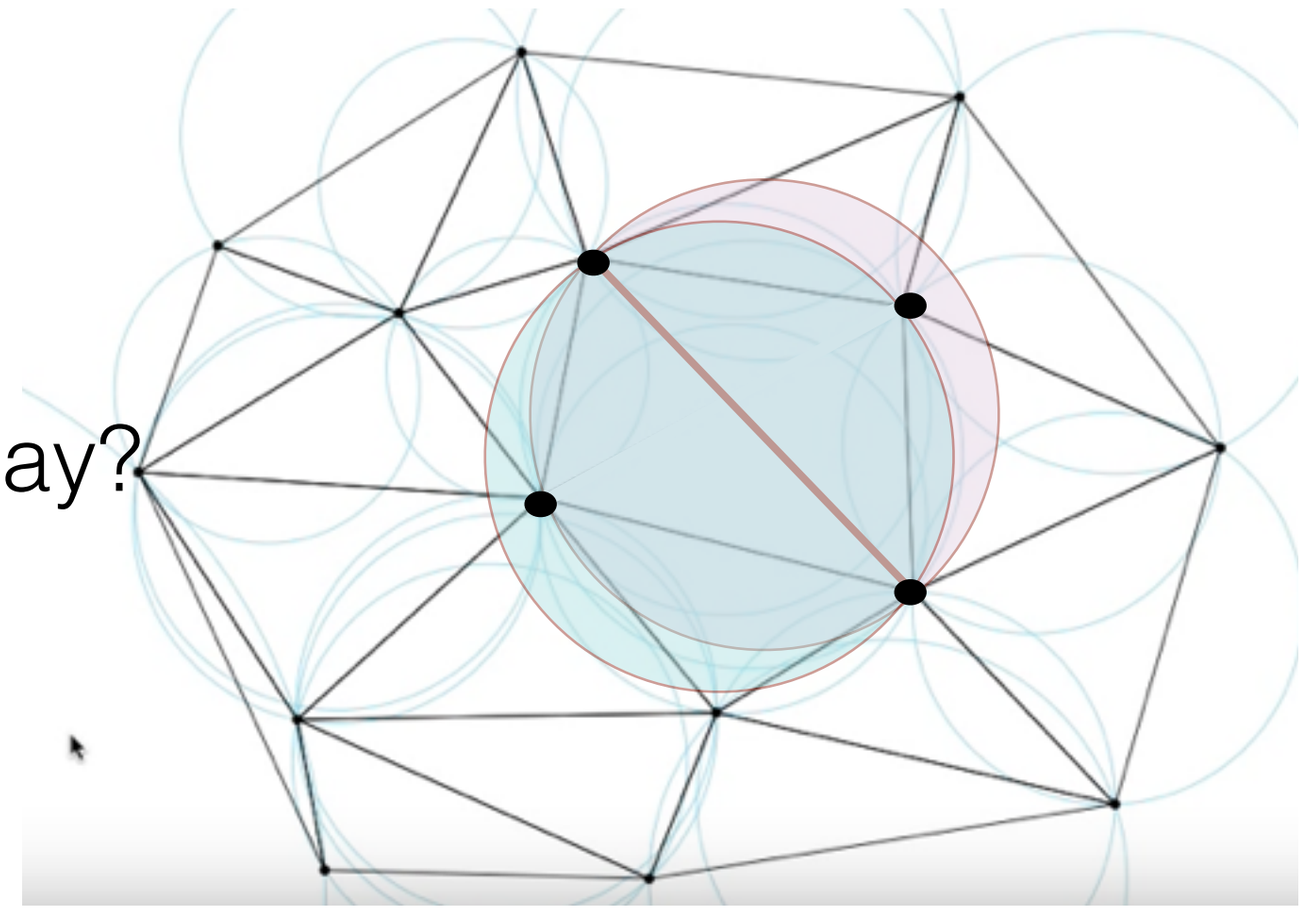
Delaunay

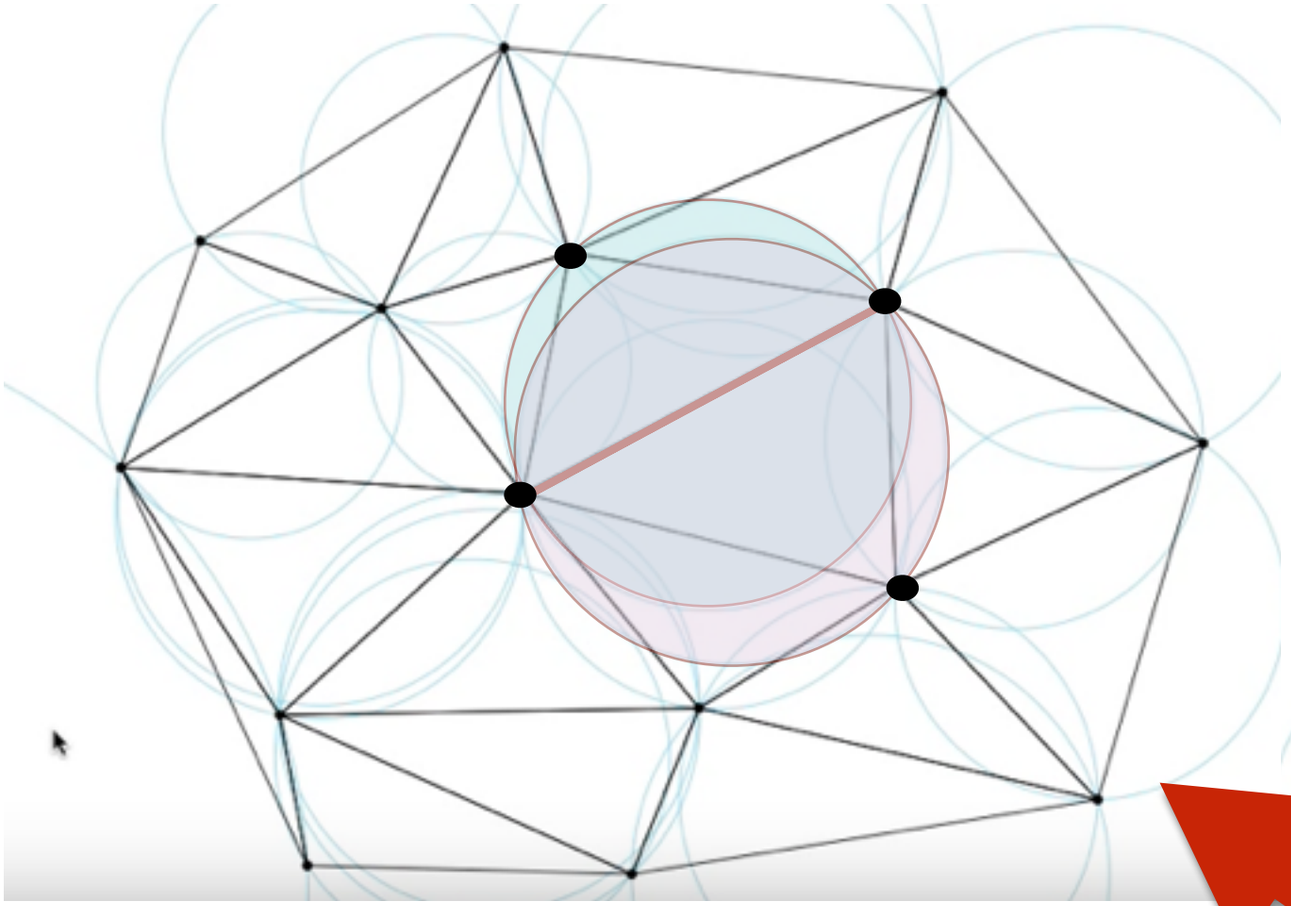
not Delaunay



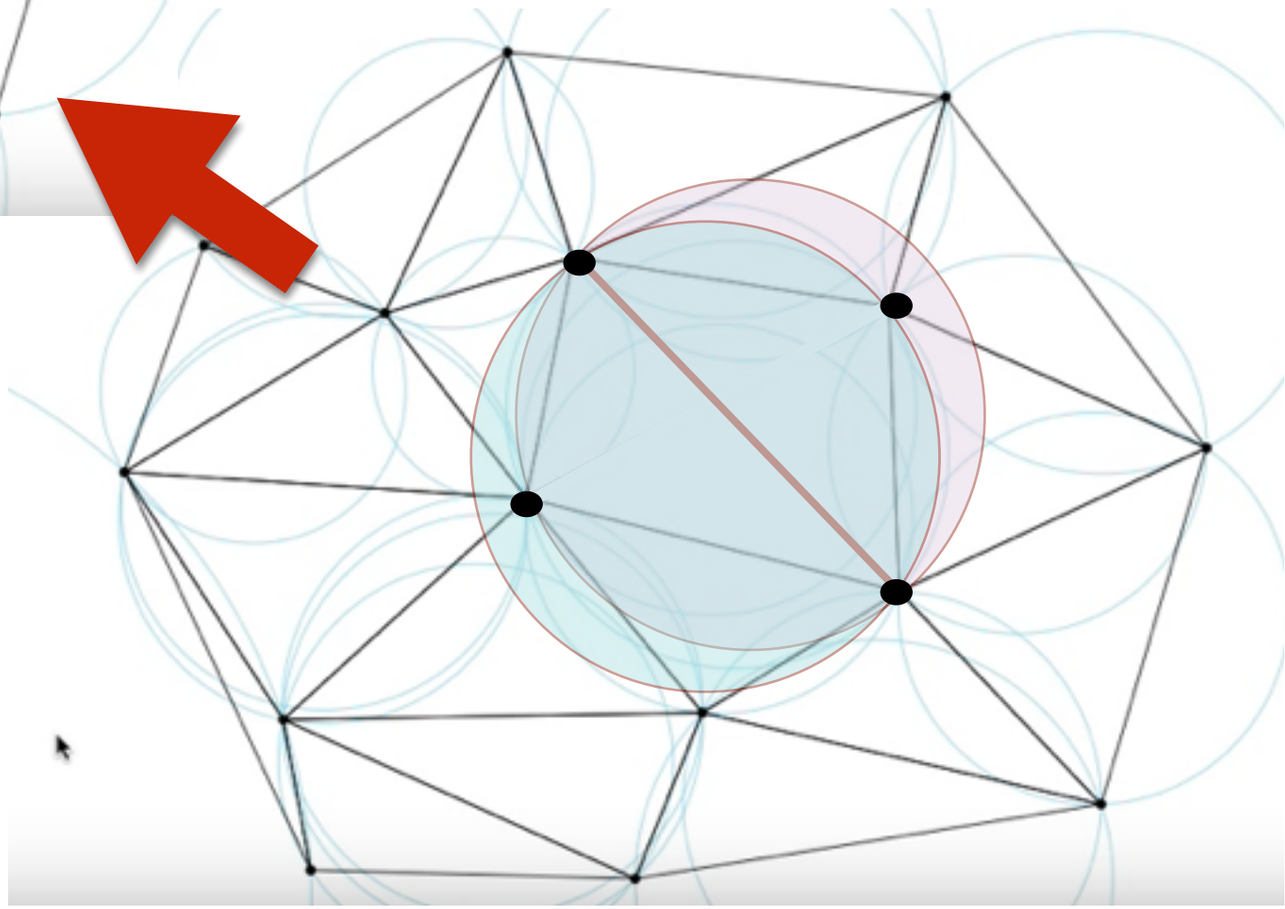
Can you make this Delaunay?

not Delaunay





Delaunay



not Delaunay

Constructing DT

We would like to build a triangulation such that all triangles have the empty-circle property.

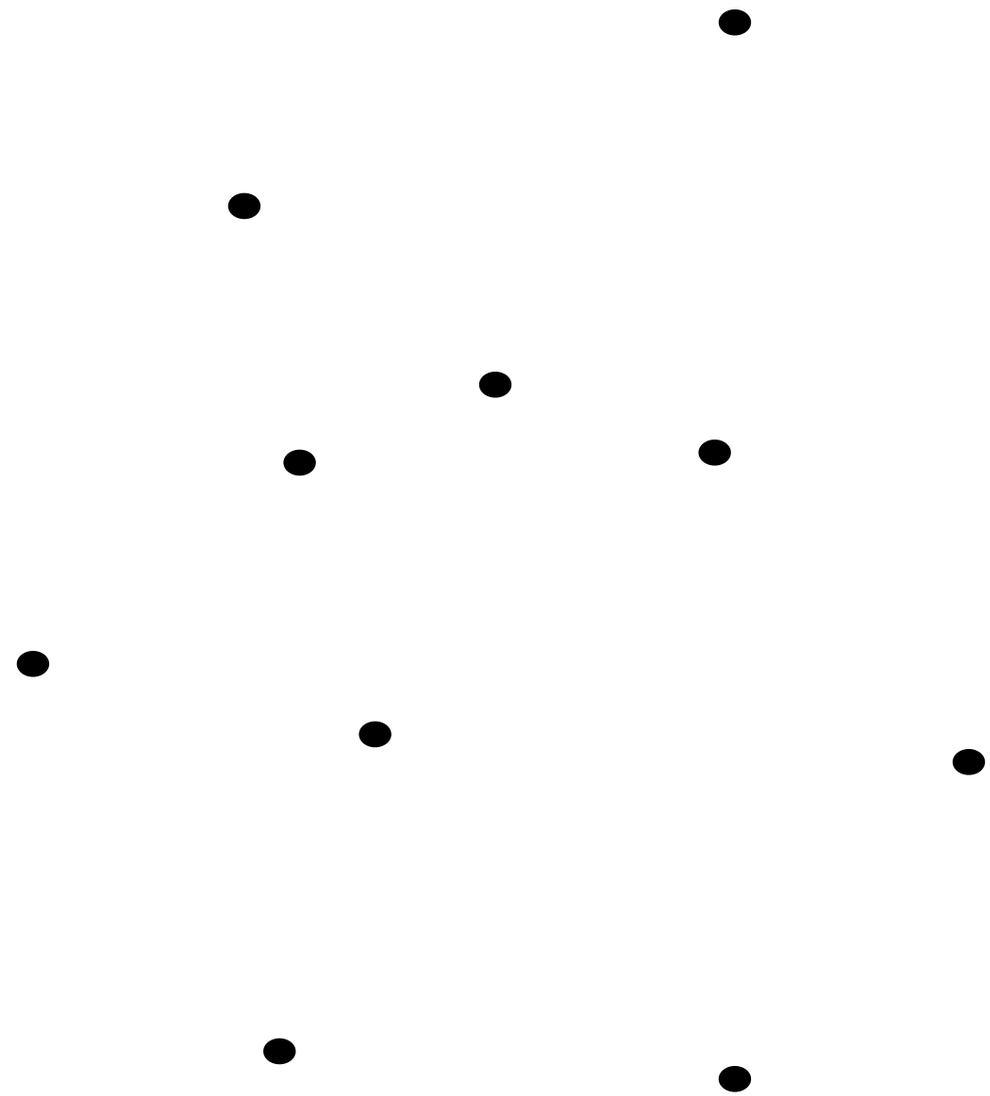
Idea:

1. Compute an arbitrary triangulation T(P)
2. Flip edges in T until all triangles have the empty circumcircle property

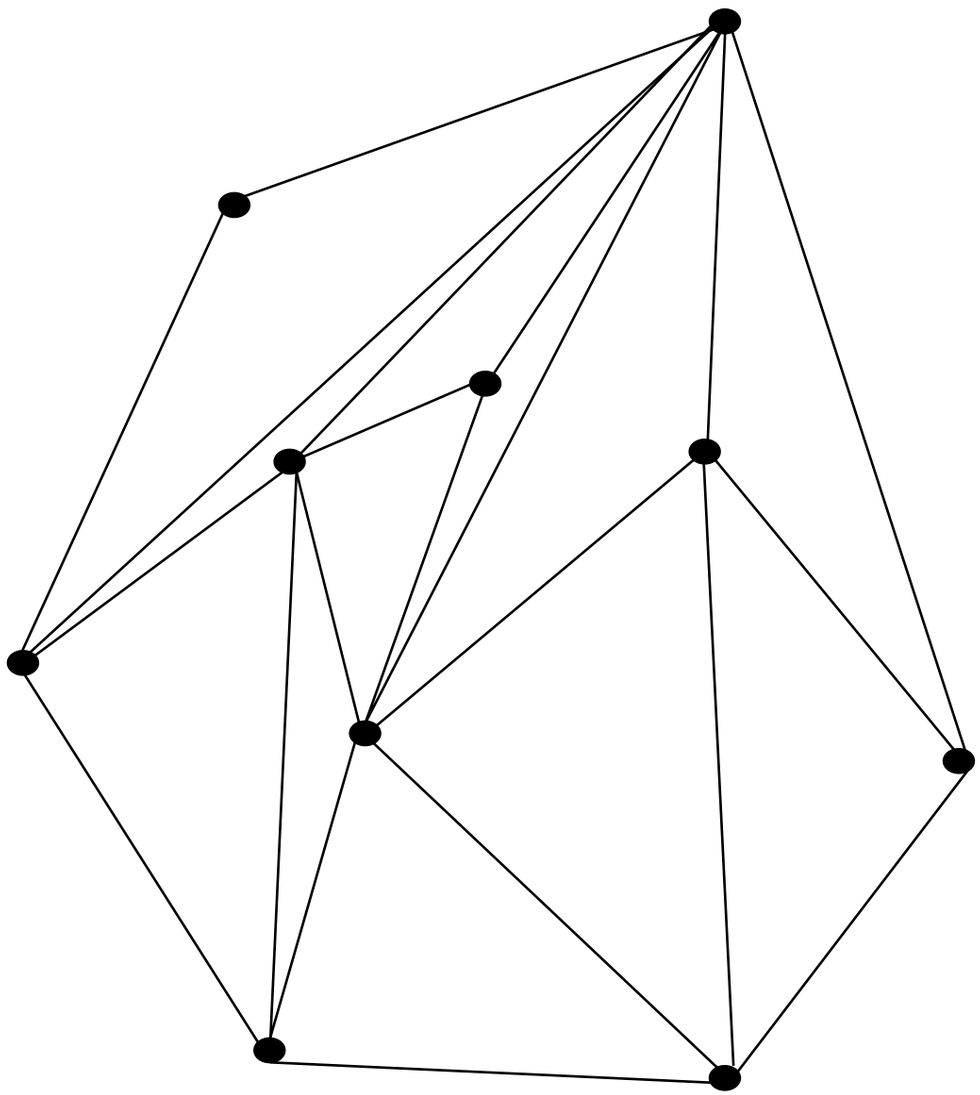
By lemma, the resulting triangulation is the Delaunay triangulation.

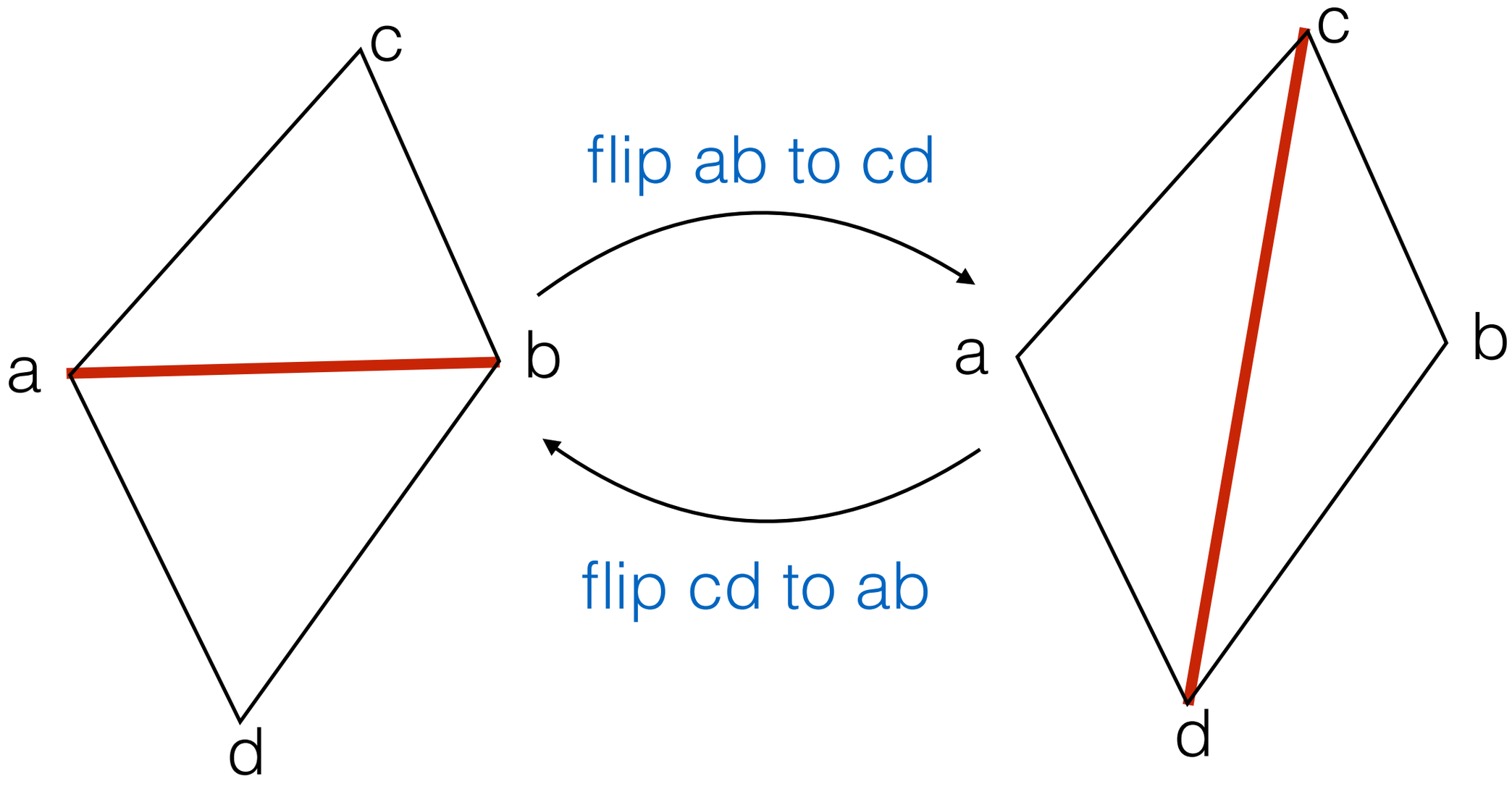
First, how to build *any* triangulation?

- Possible ideas: incremental, plane sweep, divide-and-conquer, …

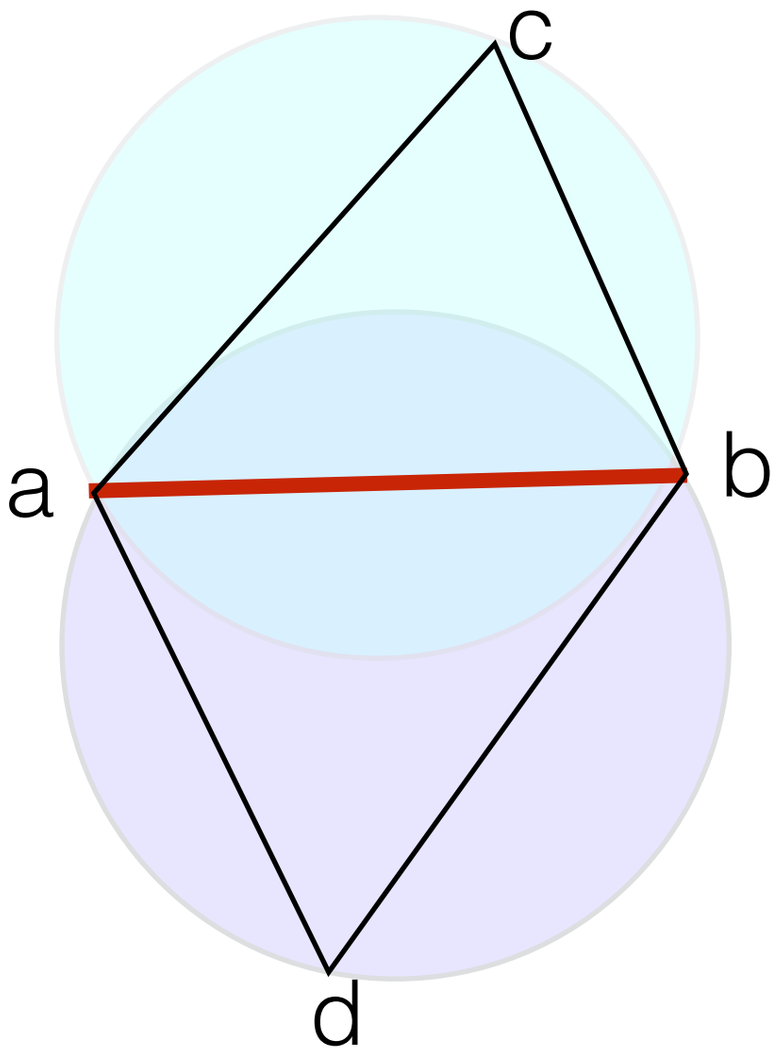


Given a triangulation of P, we can get a different triangulation by flipping an edge.



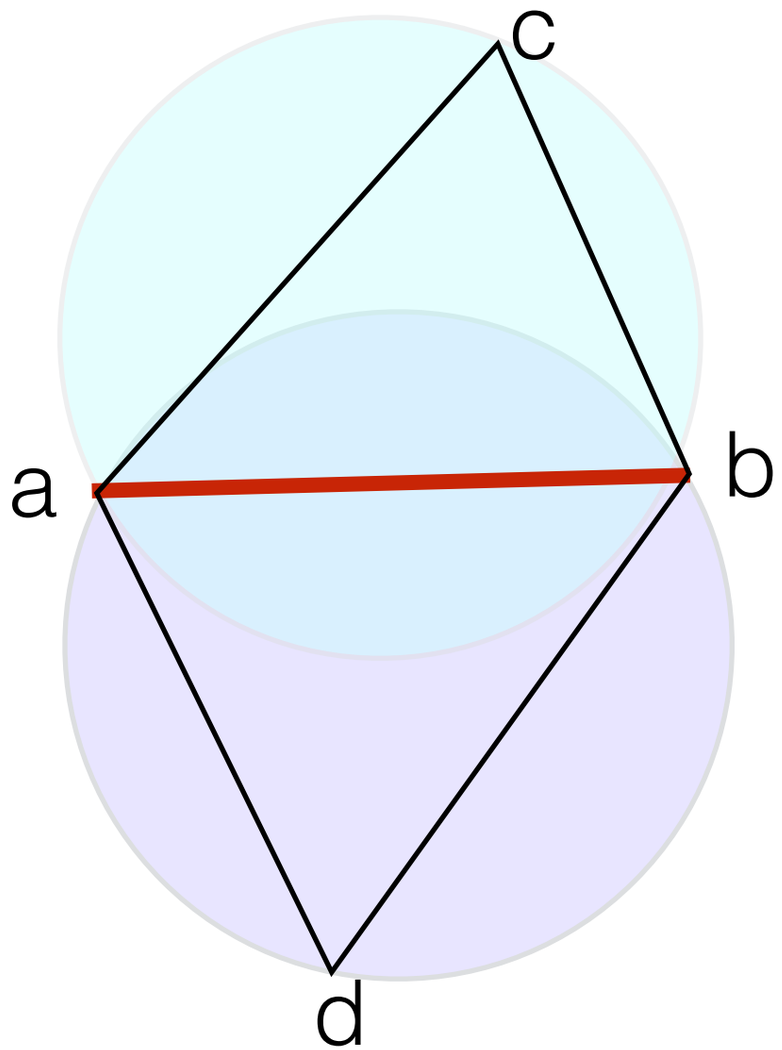

c
a
b
d
flip ab to cd
flip cd to ab
c
a
b
d

We’ll target triangles whose “empty circle property” is violated.



Do we want to flip ab?

We'll target triangles whose "empty circle property" is violated.



Do we want to flip ab?

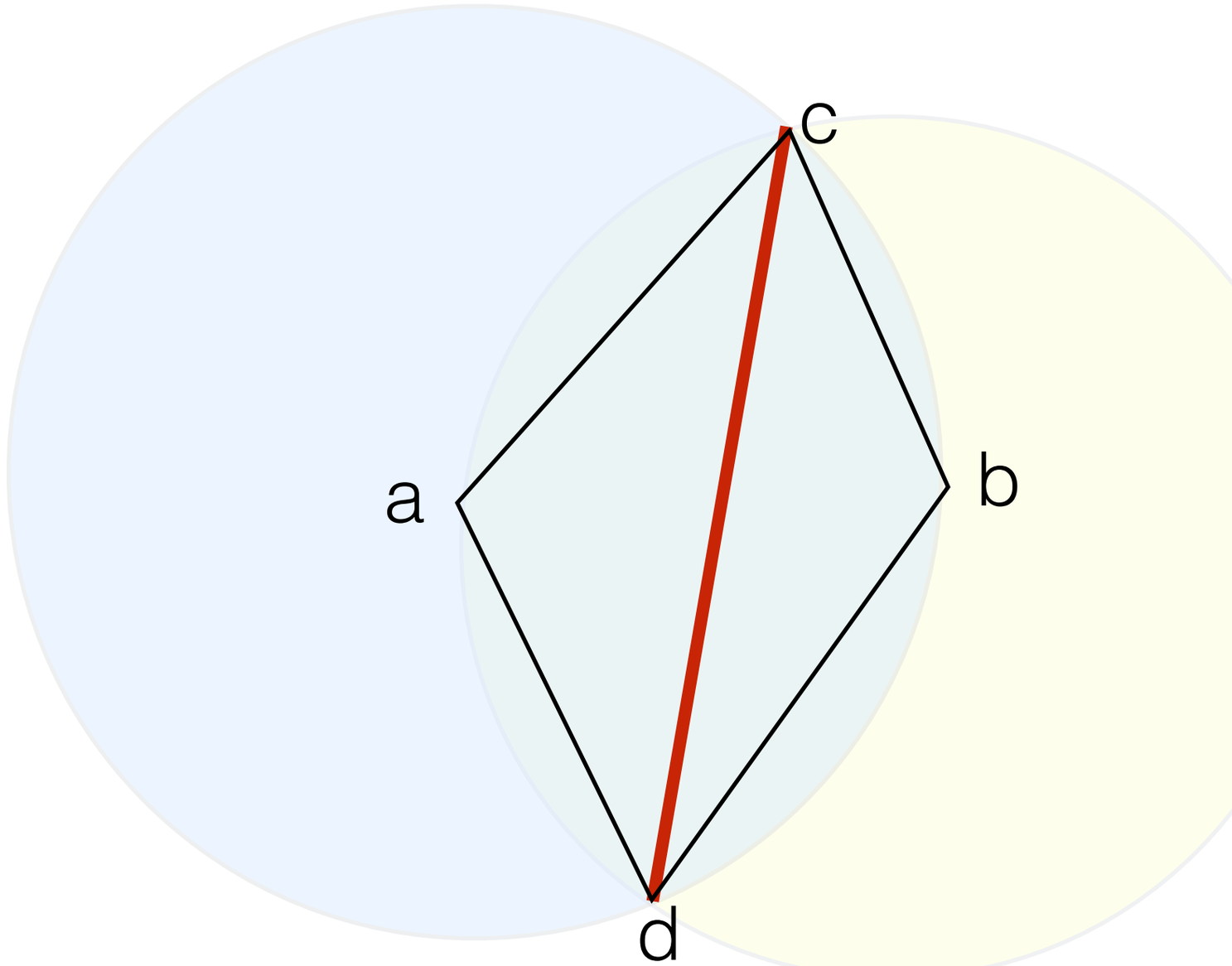
NO

c is outside C(abd)
and
d is outside C(abc)

ab is a **legal** edge

Claim: If d is outside circumcircle of abc, then c is outside circumcircle of abd, and the other way around.

Do we want to flip cd?

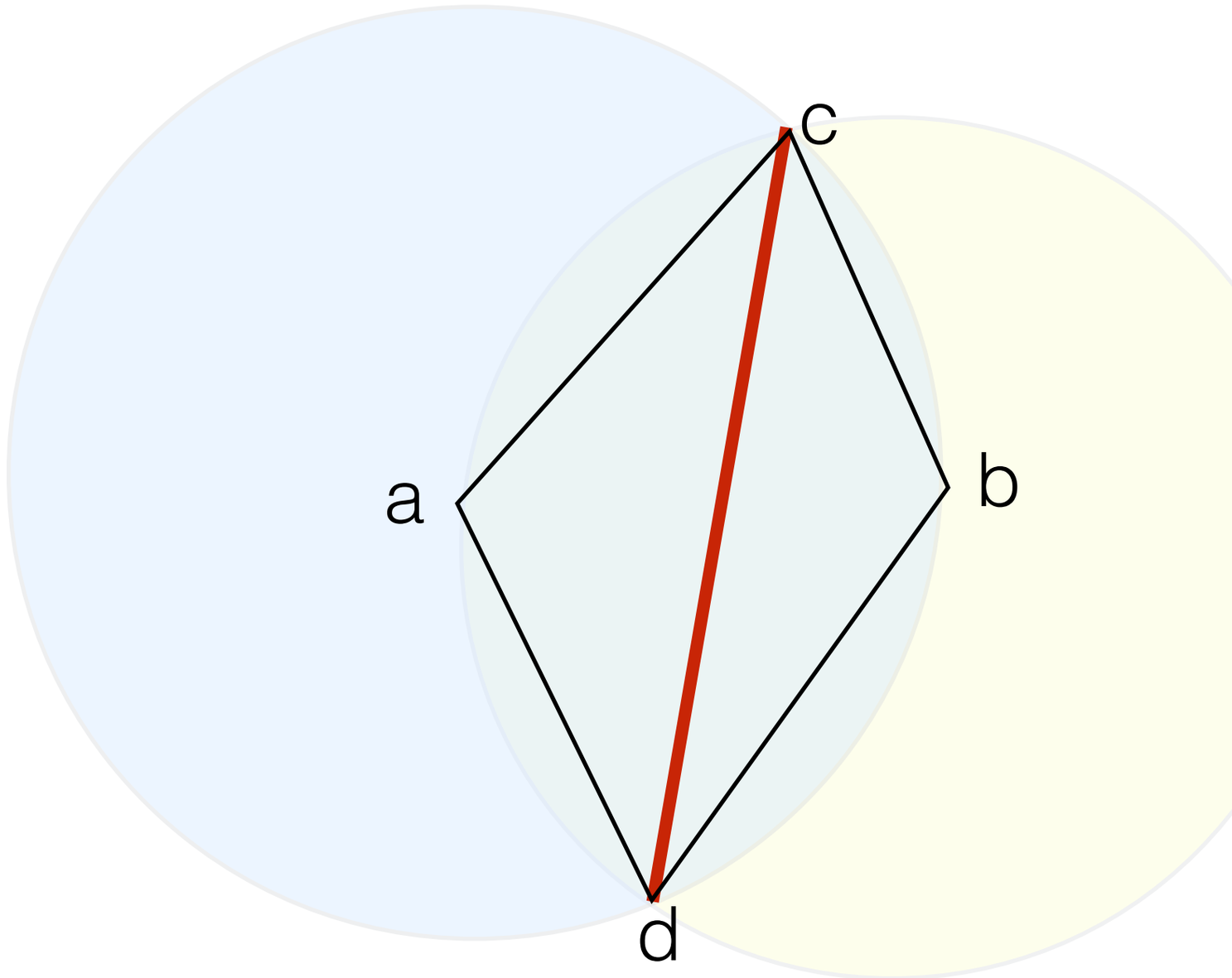


Do we want to flip cd?

YES

a is inside C(bcd)
and
b is inside C(acd)

cd is an **illegal** edge



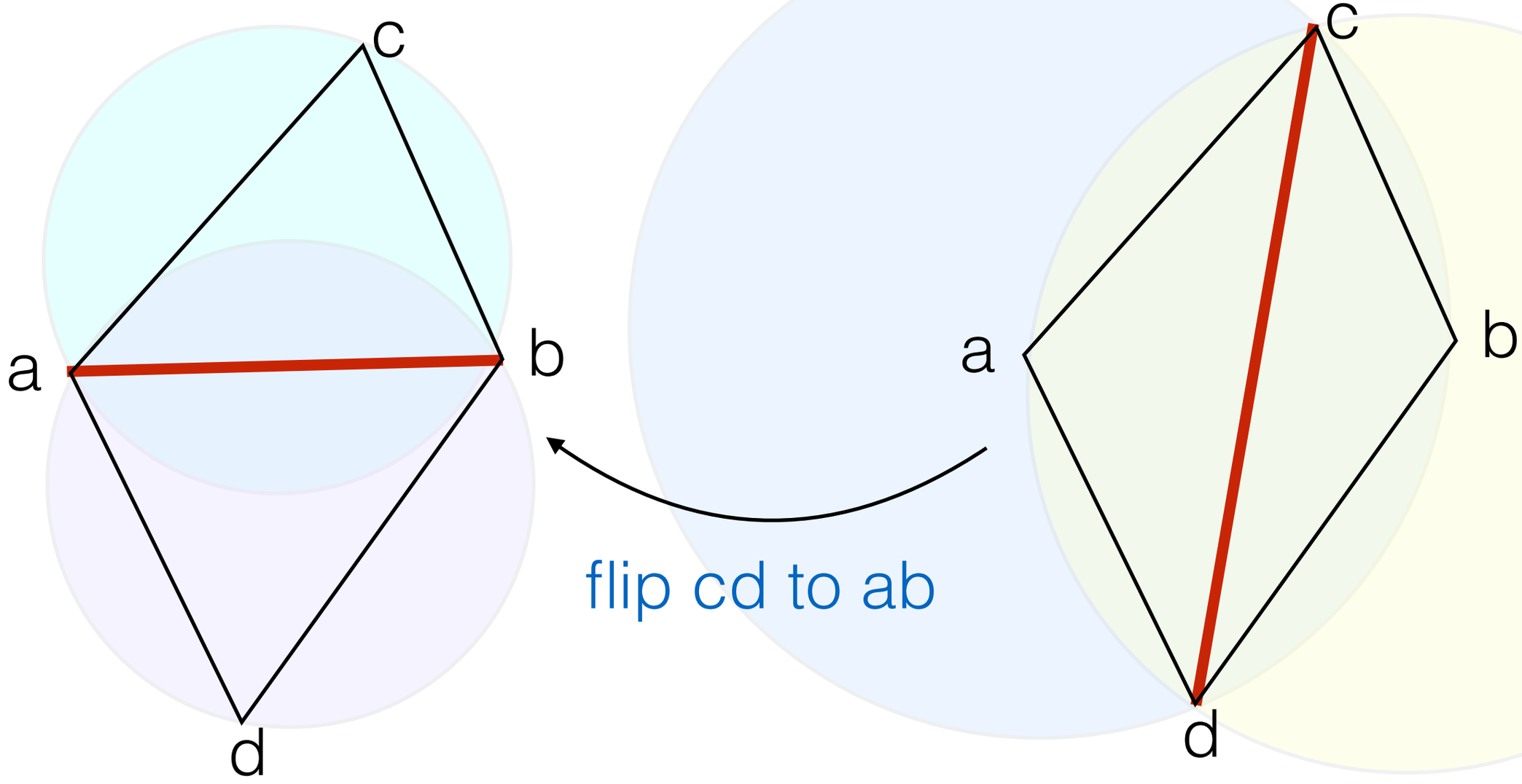
Claim: If a is inside circumcircle of bcd, then b is inside circumcircle of acd, and the other way around.

Claim:

c is outside C(abd)
d is outside C(abc)

<==

a is inside C(bcd)
b is inside C(acd)

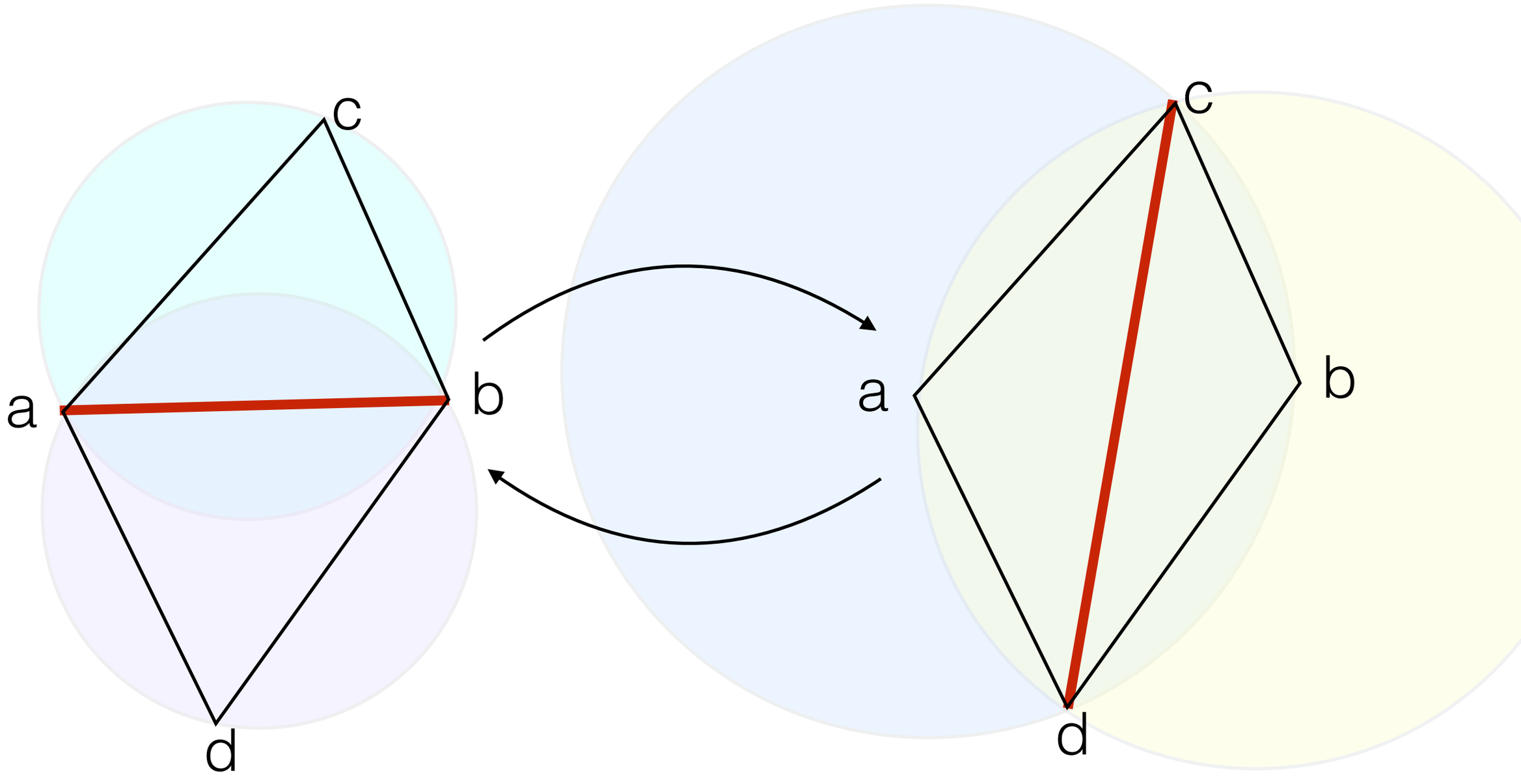


Claim:

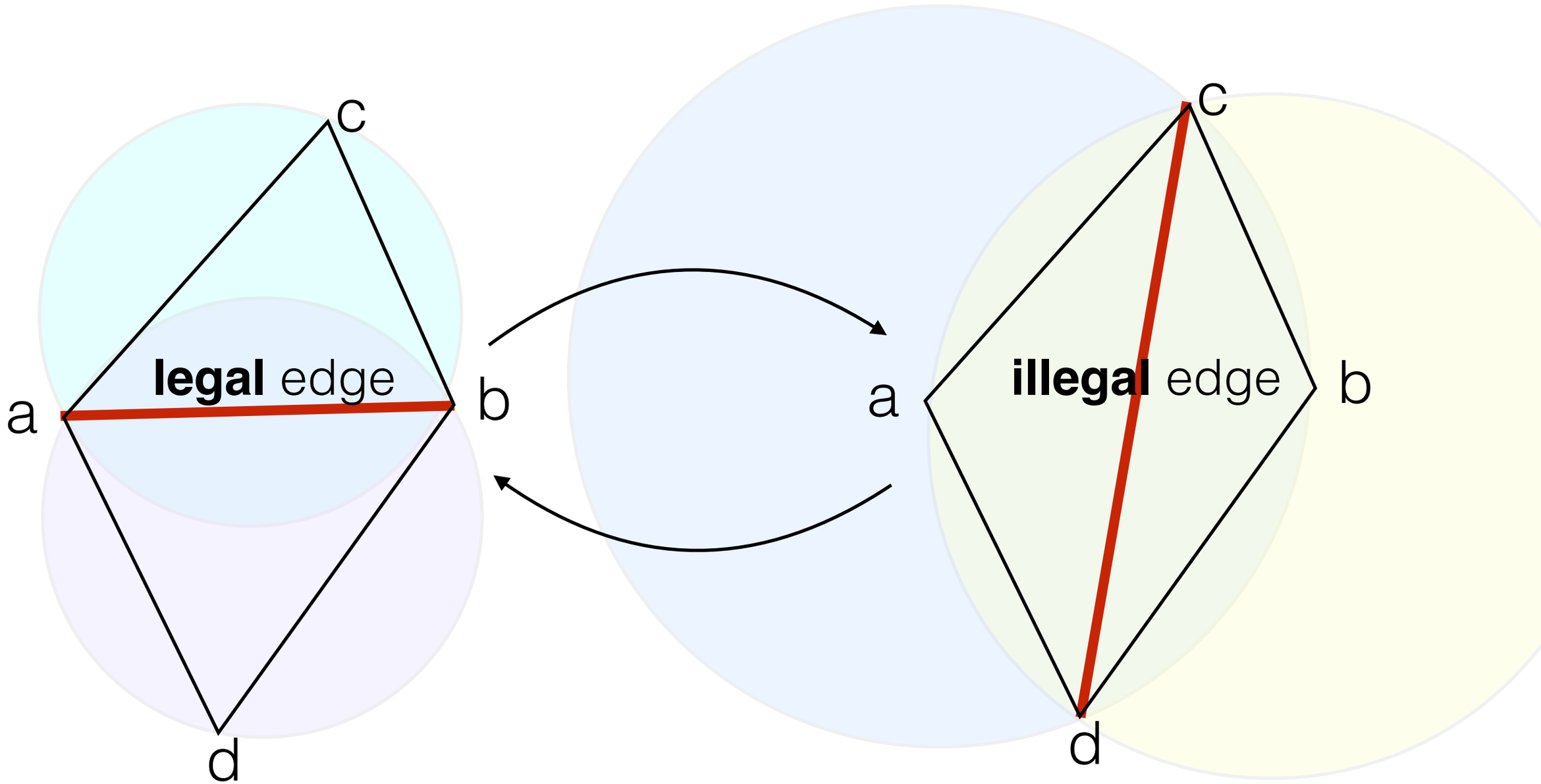
c is outside C(abd)
d is outside C(abc)

<==>

a is inside C(bcd)
b is inside C(acd)



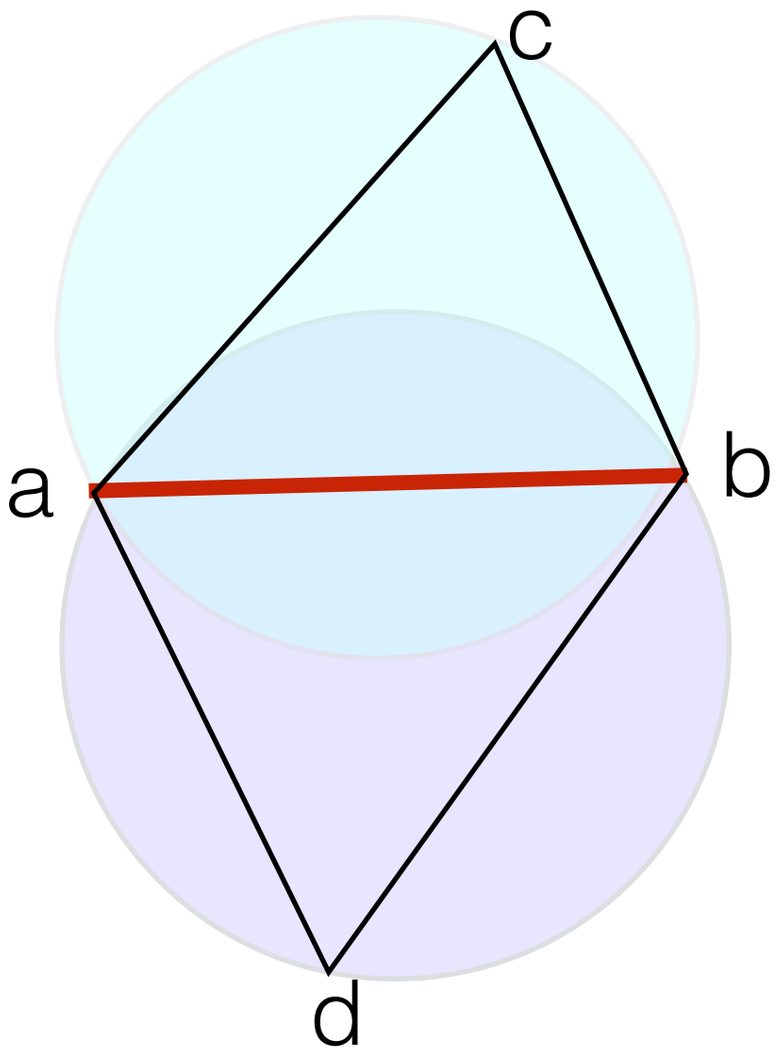
Claim: ab legal edge <==> cd illegal edge



Exercise break

Exercise 1

If d is outside C(abc), then c is outside C(abd).



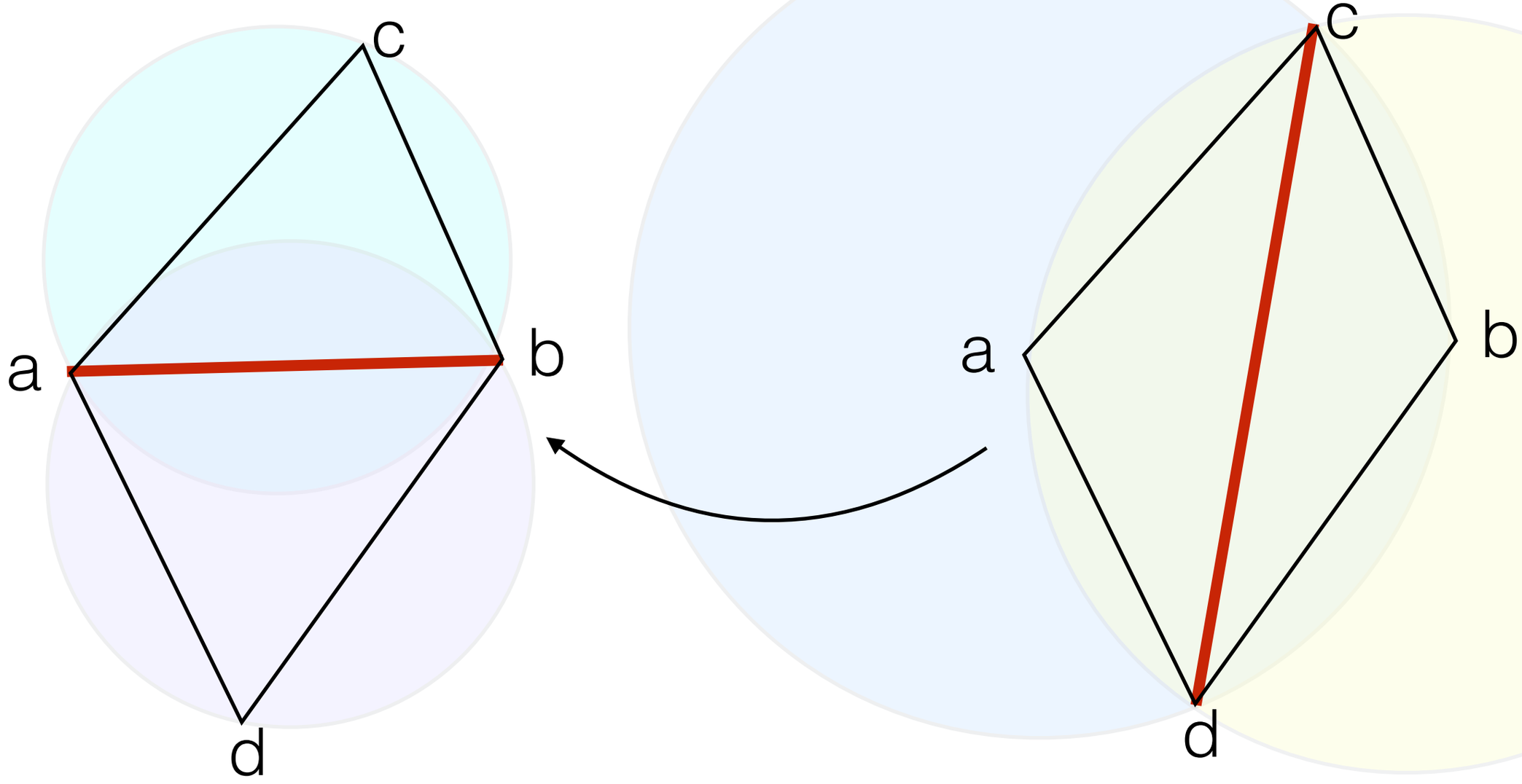
Exercise 2

Claim:

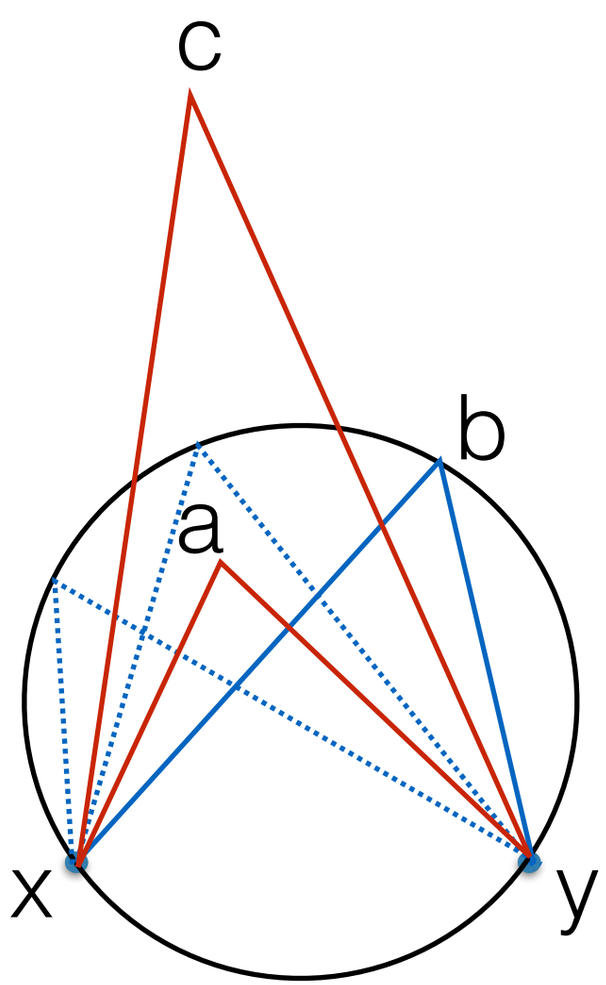
c is outside C(abd)
d is outside C(abc)

<==

a is inside C(bcd)
b is inside C(acd)



Thales Theorem



a > b > c

- We'll use it in the following way:
  - If a >b and b is on the circle, then a is inside the circle
  - If c < b and b is on the circle, then c is outside the circle

Constructing DT via edge flipping

Algorithm:

1. Compute an arbitrary triangulation T(P)
2. Flip illegal edges in T until all edges are legal

Theorem, revisited:

A triangulation where all edges are legal is the DT.

Proof: If all edges are legal then all triangles circumcircles are empty. Therefore by previous lemma this is the DT.

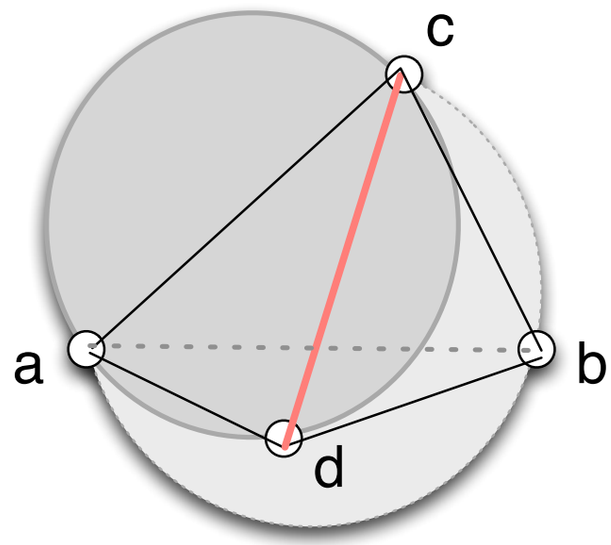
Constructing DT via edge flipping

Algorithm EdgeFlipDelaunay(P)

- construct an arbitrary triangulation T
- insert all edges of T in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S
  - if ab illegal:
    - flip it (and update the triangulation)
    - for each new edge ac, ad, bc, bd: insert it in S, unless already in S



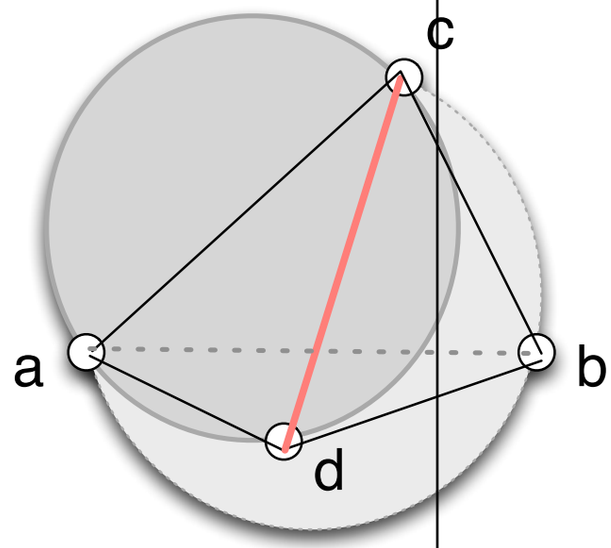
Constructing DT via edge flipping

Algorithm EdgeFlipDelaunay(P)

- construct an arbitrary triangulation T
- insert all edges of T in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S
  - if ab illegal:
    - flip it (and update the triangulation)
    - for each new edge ac, ad, bc, bd: insert it in S, unless already in S



Questions:

- Does this always terminate?
- When it terminates, is it the DT?
- Running time?

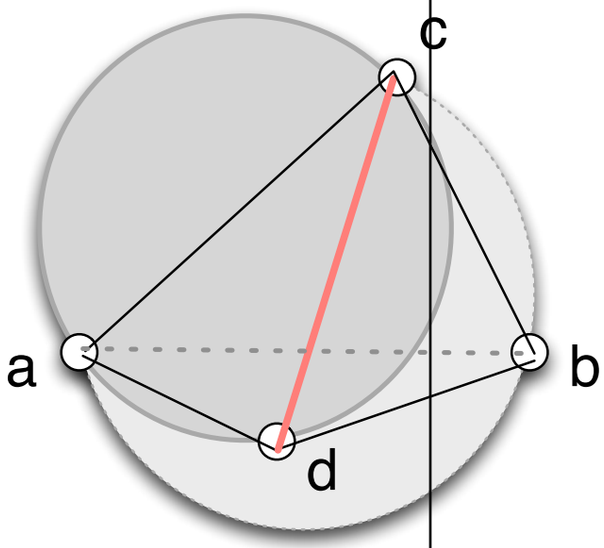
Constructing DT via edge flipping

Algorithm EdgeFlipDelaunay(P)

- construct an arbitrary triangulation T
- insert all edges of T in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S
  - if ab illegal:
    - flip it (and update the triangulation)
    - for each new edge ac, ad, bc, bd: insert it in S, unless already in S



Questions:

- Does this always terminate?
- When it terminates, is it the DT? ← yes, by theorem
- Running time?

Does it always terminate?

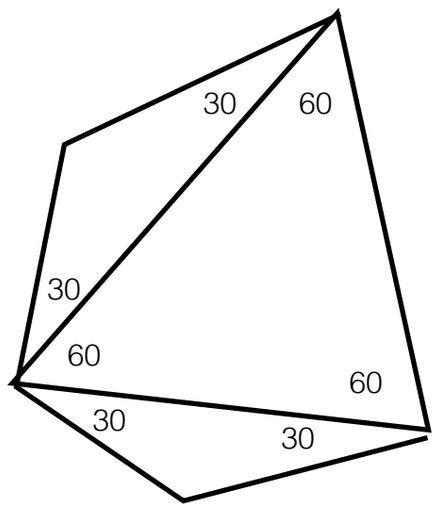
An argument using angles, that leads to new insight.

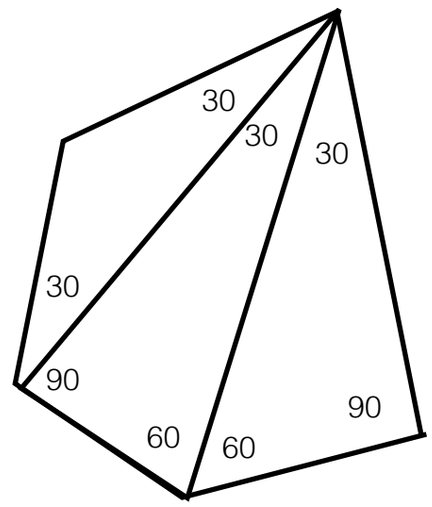
The Angle-vector of a triangulation

- Recall that every triangulation of P has the same number of triangles, call it m
- Sort the 3m angles increasingly and let A(T) be the resulting sequence
- $A(T) = [a_1 \leqq a_2 \leqq \ldots a \leqq {}_{3m}]$ <———————- call this the angle vector of T

The Angle Vector

Example





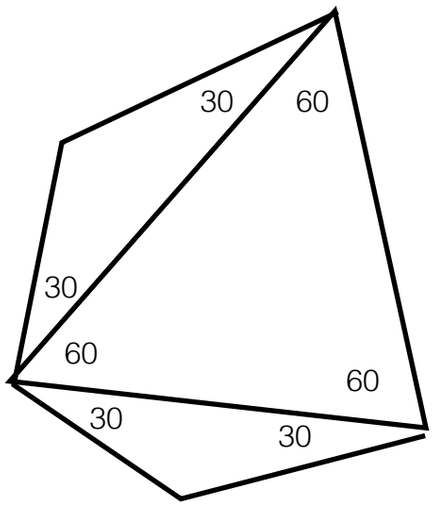
A(T) = [30, 30, 30, 30, 60, 60, 60, 120, 120]

A(T') = [30, 30, 30, 30, 60, 60, 90, 90, 120]

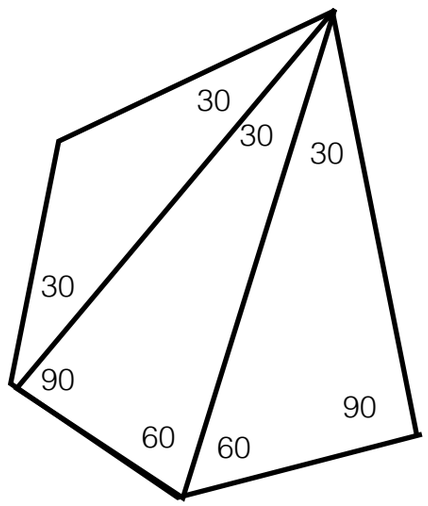
The Angle Vector

- Lexicographic order
  - A(T) < A(T') if there exists i, 1 ≦ i ≦ 3m such that $a_j = a_j'$, for all j < i, and $a_i < a_i'$

Example



A(T) < A(T')

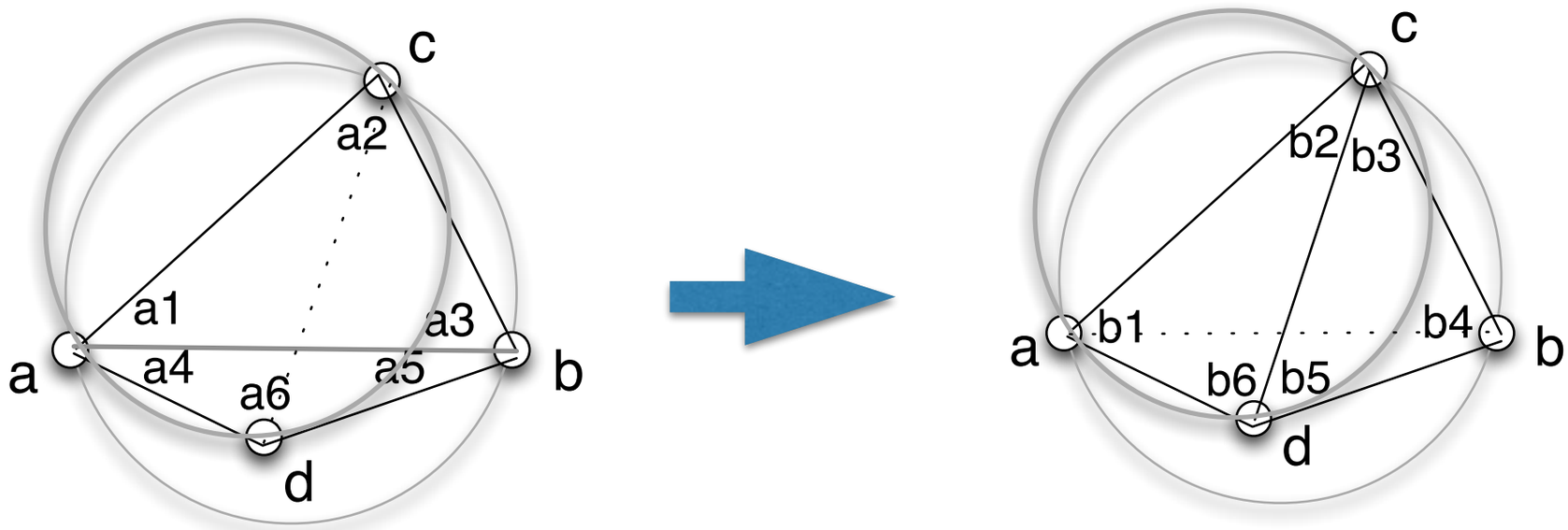


A(T) = [**30, 30, 30, 30, 60**, **60**, **60**, 120, 120] A(T') = [**30, 30, 30, 30, 60**, **60**, **90**, 90, 120]

Comparing Angle Vectors

- If A(T) < A(T') then
  - T' has a larger smallest angle; or
  - they both have the same smallest angle, but T' has a larger second smallest angle; or
  - or so on

What happens to the angle vector when we flip an illegal edge?

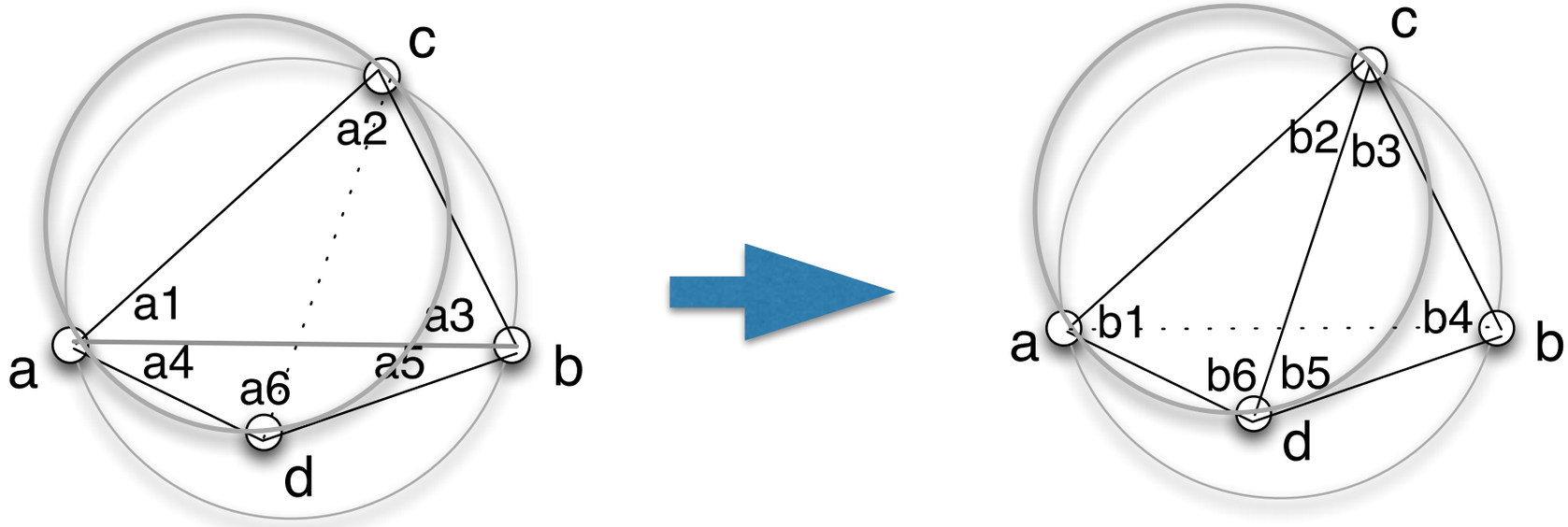


before : A(T) = [a1, a2, a3, a4, a5, a6]

after : A(T') = [b1, b2, b3, b4, b5, b6]

The flip is "local" and does not affect the angle vector outside abcd
==> We look only at the angles inside the two triangles

What happens to the angle vector when we flip an illegal edge?



before : A(T) = [a1, a2, a3, a4, a5, a6]

after : A(T') = [b1, b2, b3, b4, b5, b6]

Lemma: For each new angle, there is an old angle that's < than it.

Proof:

b1 = a1+a4 > a1

b2 > a5 (b outside C(acd))

b3 > … similar

b4 = a3 + a5 > a3

b5 > … similar

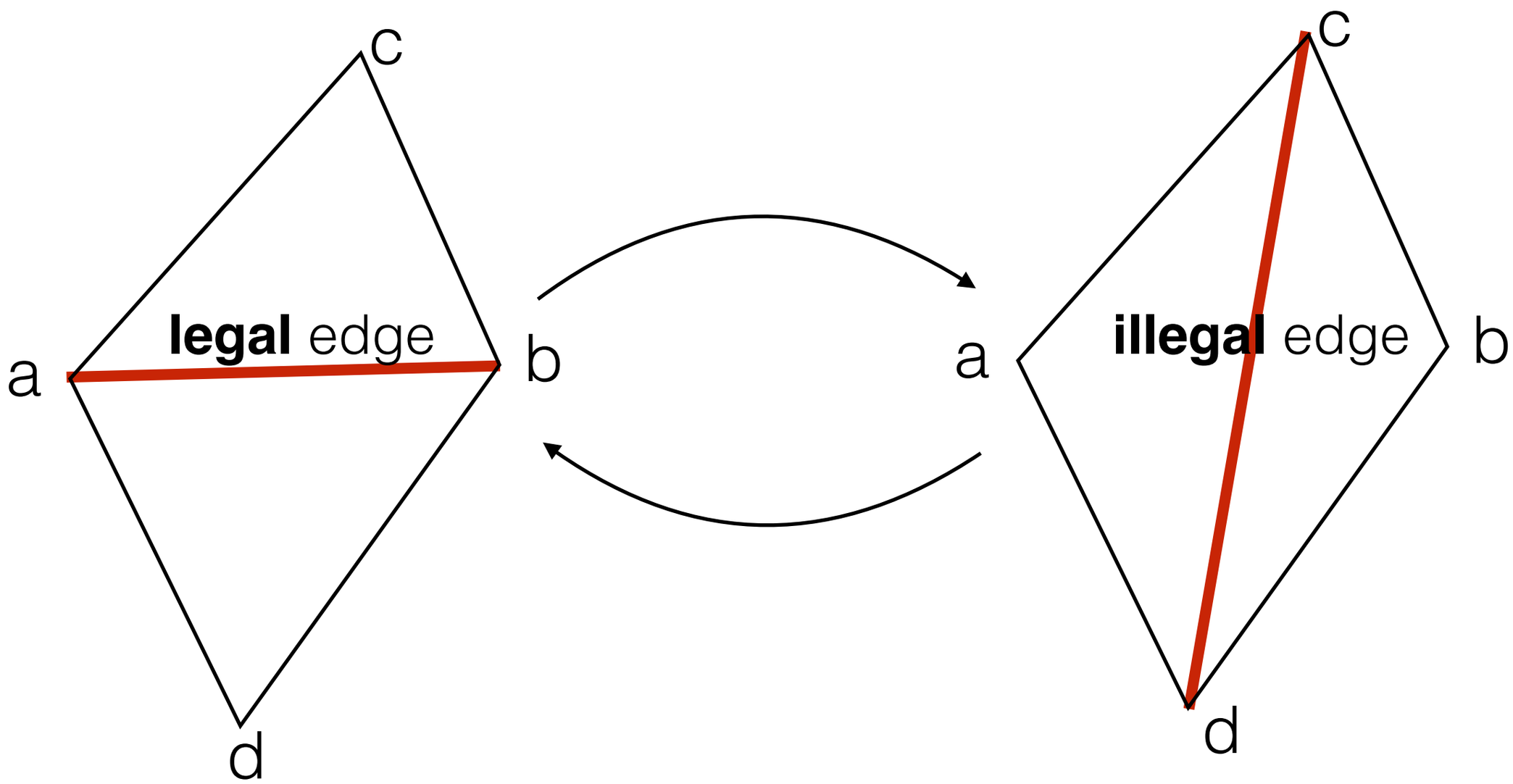
b6 > a3 (b outside C(acd))

A(T) < A(T')

T

A(T) > A(T')

T'



Claim: DT maximizes the angle vector over all possible triangulations of P.

Proof: DT has only legal edges. Any flip will cause the angle vector to decrease. Could this be a local max? All triangles have empty circles, therefore this must be the DT, which is unique.

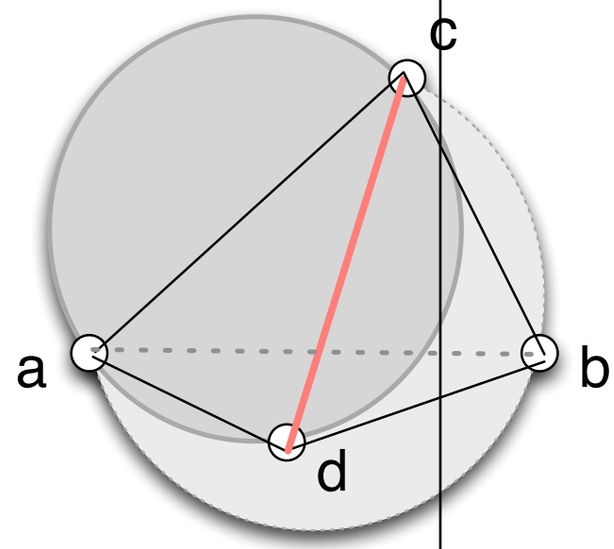
Termination and running time

Algorithm EdgeFlipDelaunay(P)

- construct an arbitrary triangulation T
- insert all edges of T in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S
  - if ab illegal:
    - flip it (and update the triangulation)
    - for each new edge ac, ad, bc, bd: insert it in S, unless already in S



We'll argue based on A(T)

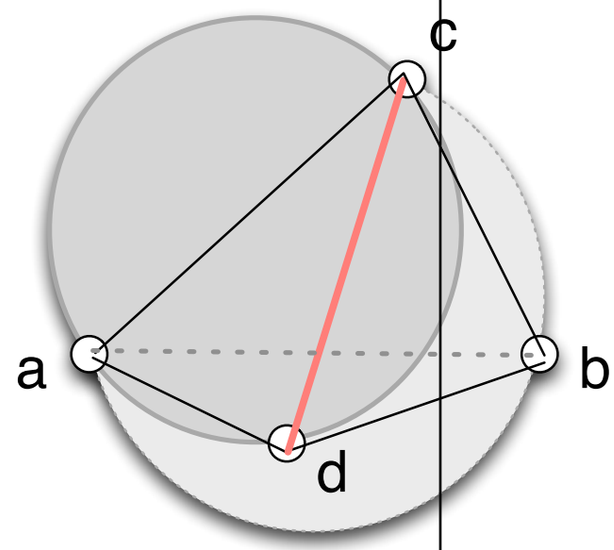
Termination and running time

Algorithm EdgeFlipDelaunay(P)

- construct an arbitrary triangulation T
- insert all edges of T in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S
  - if ab illegal:
    - flip it (and update the triangulation)
    - for each new edge ac, ad, bc, bd: insert it in S, unless already in S



We'll argue based on A(T)

- $O(n^2)$ possible edges.
- Every edge flip increases the angle vector.
- An edge flip takes O(1) time, and inserts O(1) edges in S
- Once flipped, (made legal) an edge cannot be unflipped

$O(n^2)$

DT via Randomized Incremental Construction

Computing DT via RIC

RIC (DT Randomized Incremental Construction)

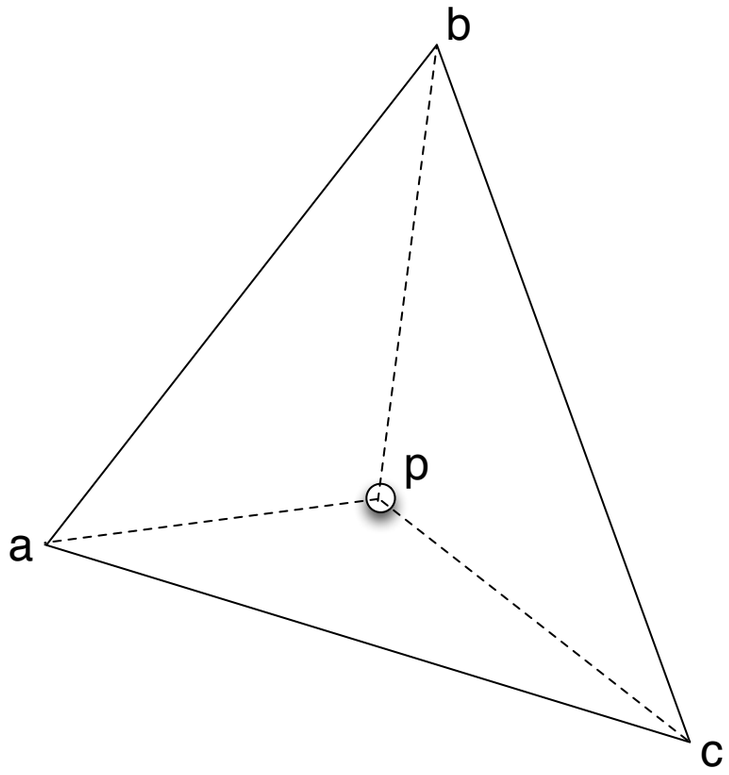
- Let (p_1, p_2, p_3, …p_n) be a random permutation of P

  //Let T be the current triangulation

- Initialize T as a large triangle that contains P.
- For next point p_i in P do:

  //insert p_i in T

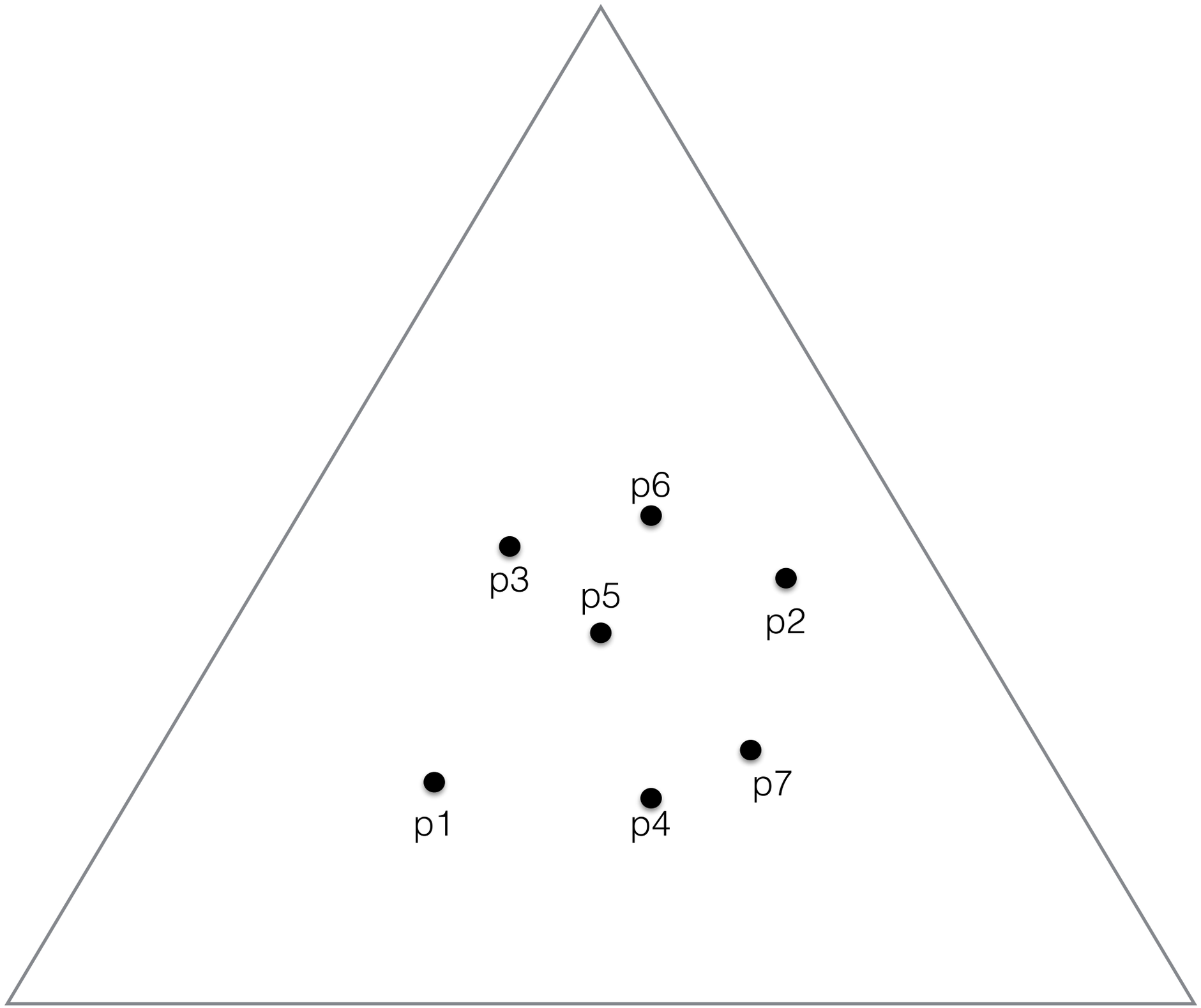
  - find triangle abc of T that contains p_i
  - splitting into 3 triangles: abp_i, bcp_i, cap_i and update T
  - LegalizeEdge(p_i, ab, T)
  - LegalizeEdge(p_i, bc, T)
  - LegalizeEdge(p_i, ca, T)
- Discard the initial triangle T and its edges
- return T

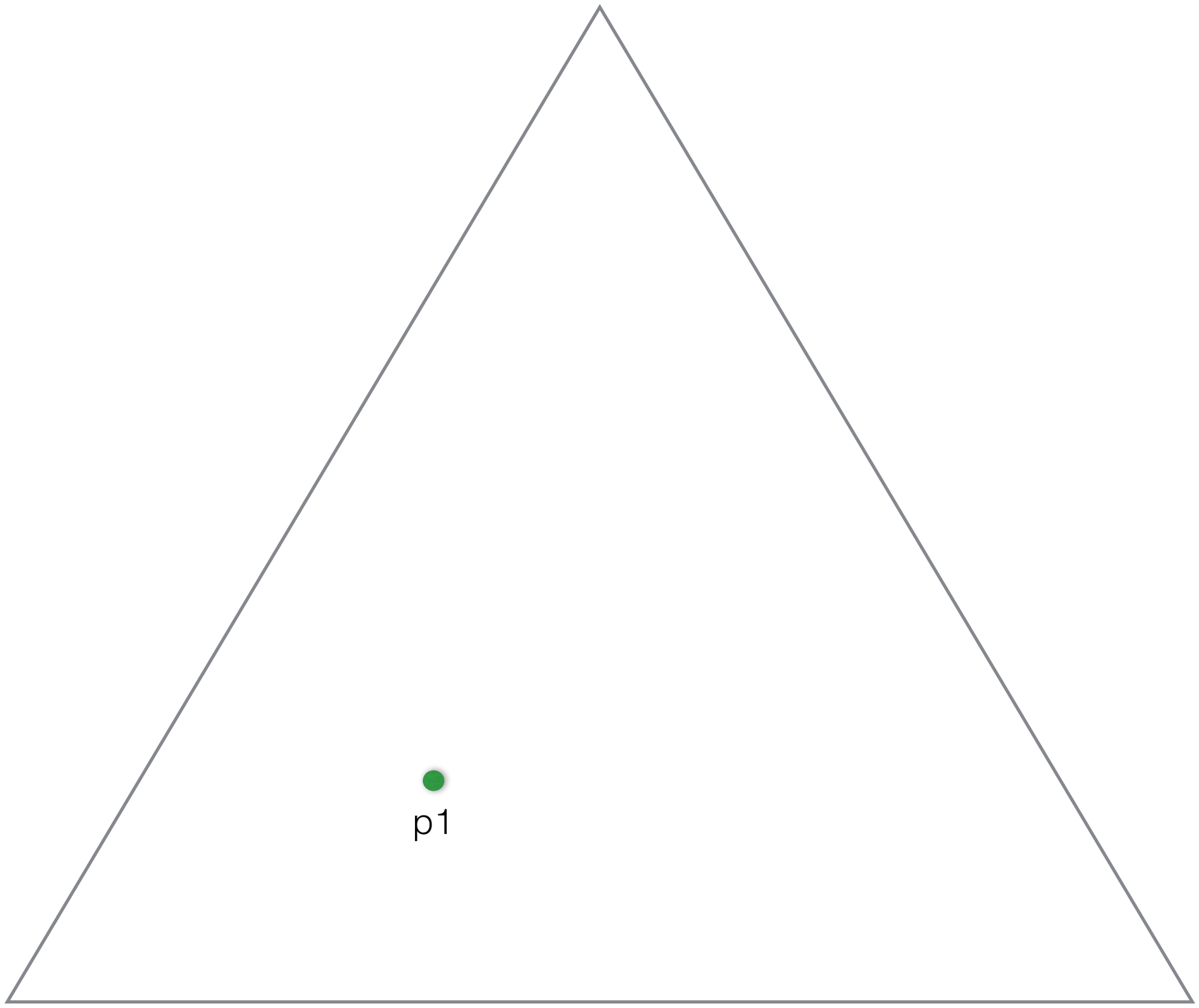


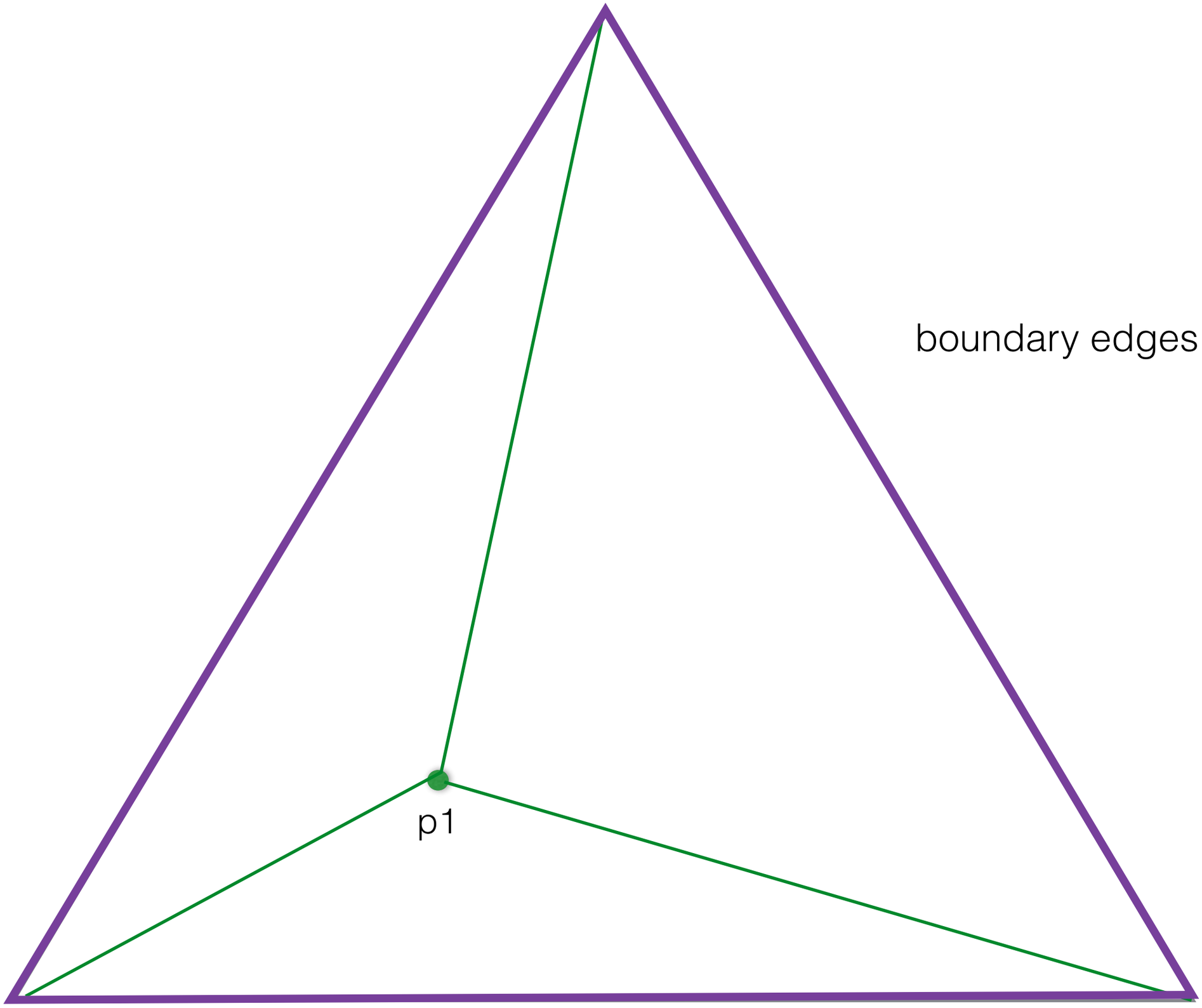
LegalizeEdge(p, uv, T)

//point p was inserted, and puv is a new triangle; edge uv may need to be flipped

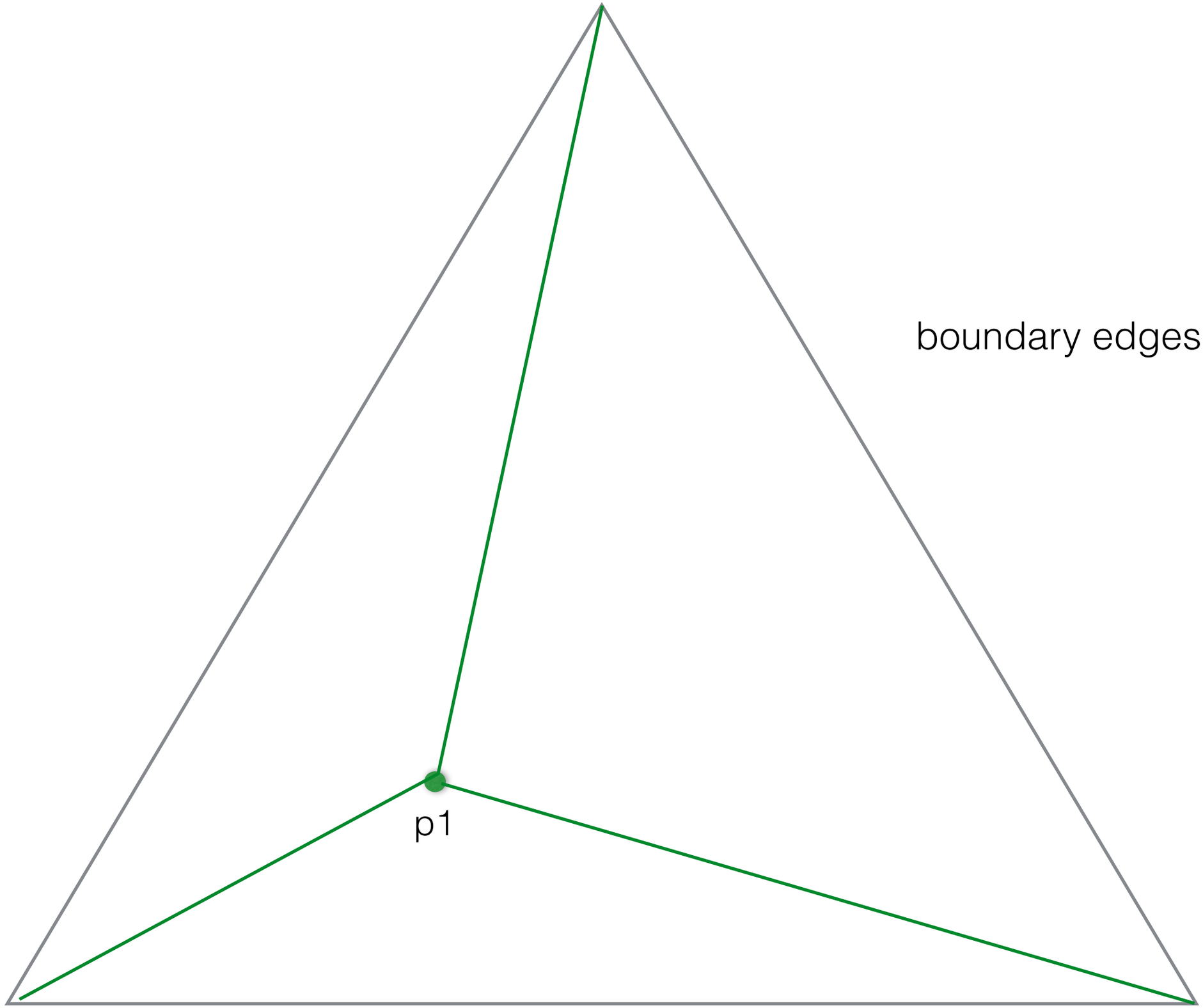
- let uvq be the triangle adjacent to uv, on the other side of p
- if uv is illegal
  - flip uv and update T
  - LegalizeEdge(p, uq, T)
  - LegalizeEdge(p, qv, T)


p6
p3
p5
p2
p7
p1
p4

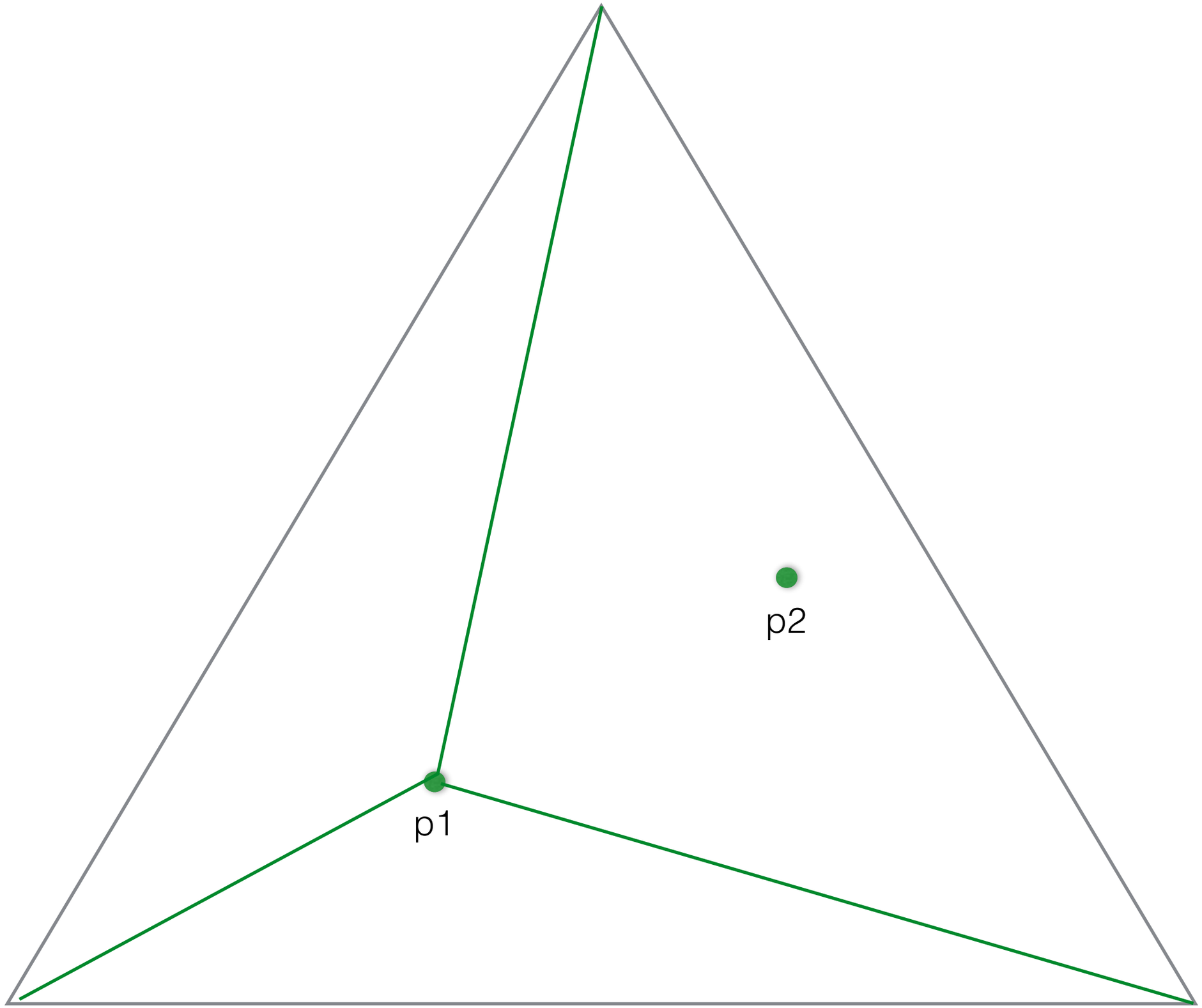

p1

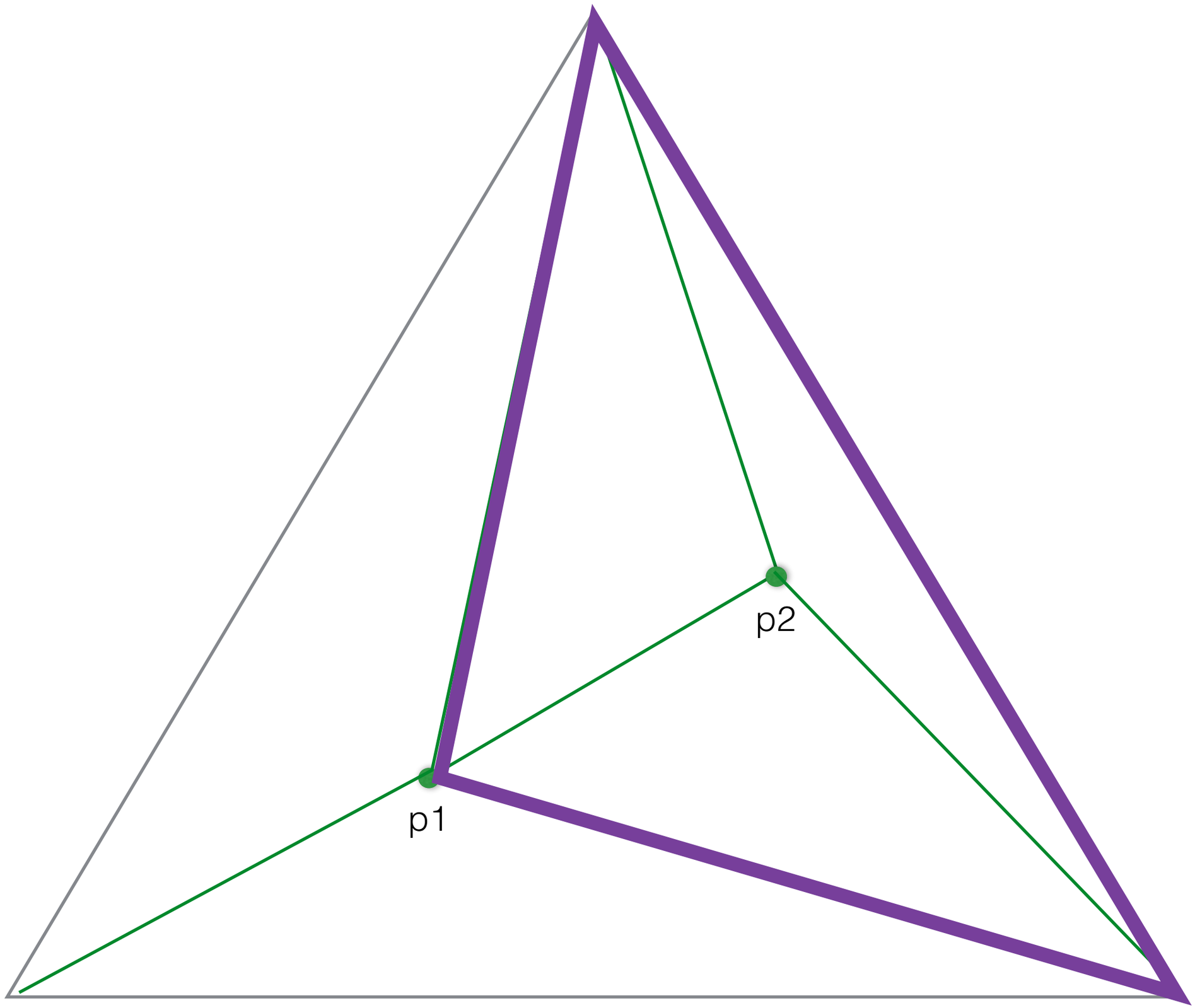


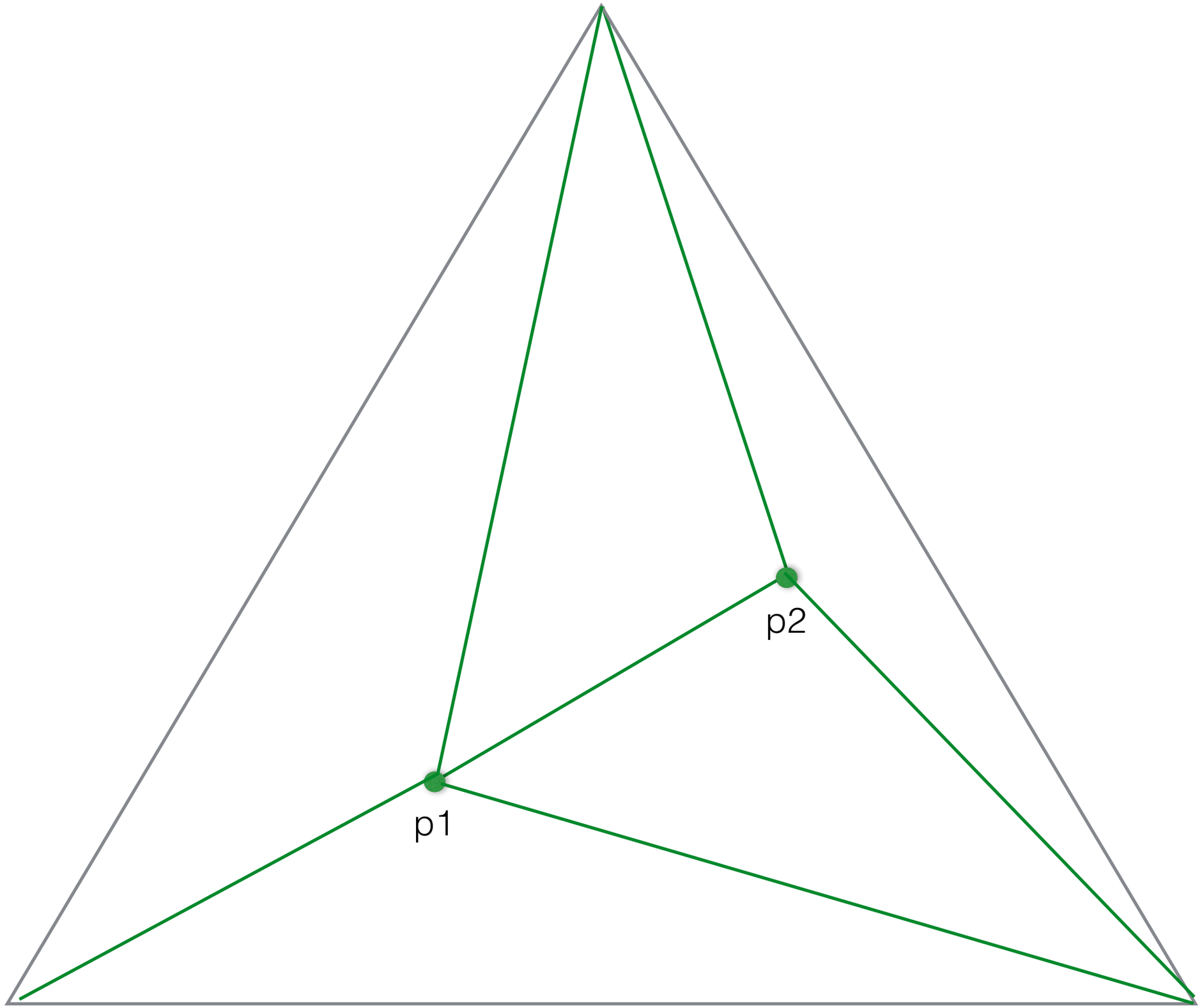
boundary edges are legal

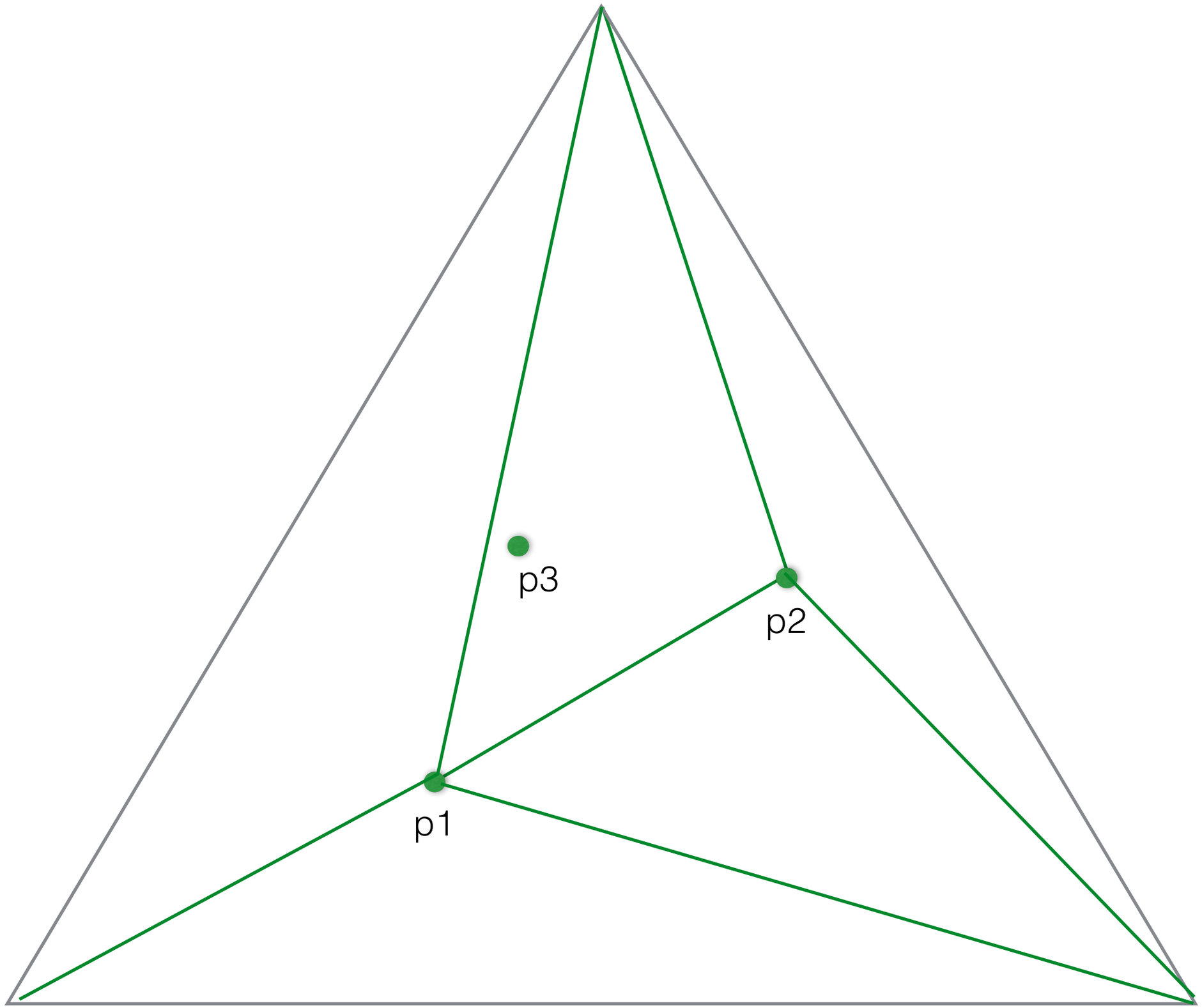


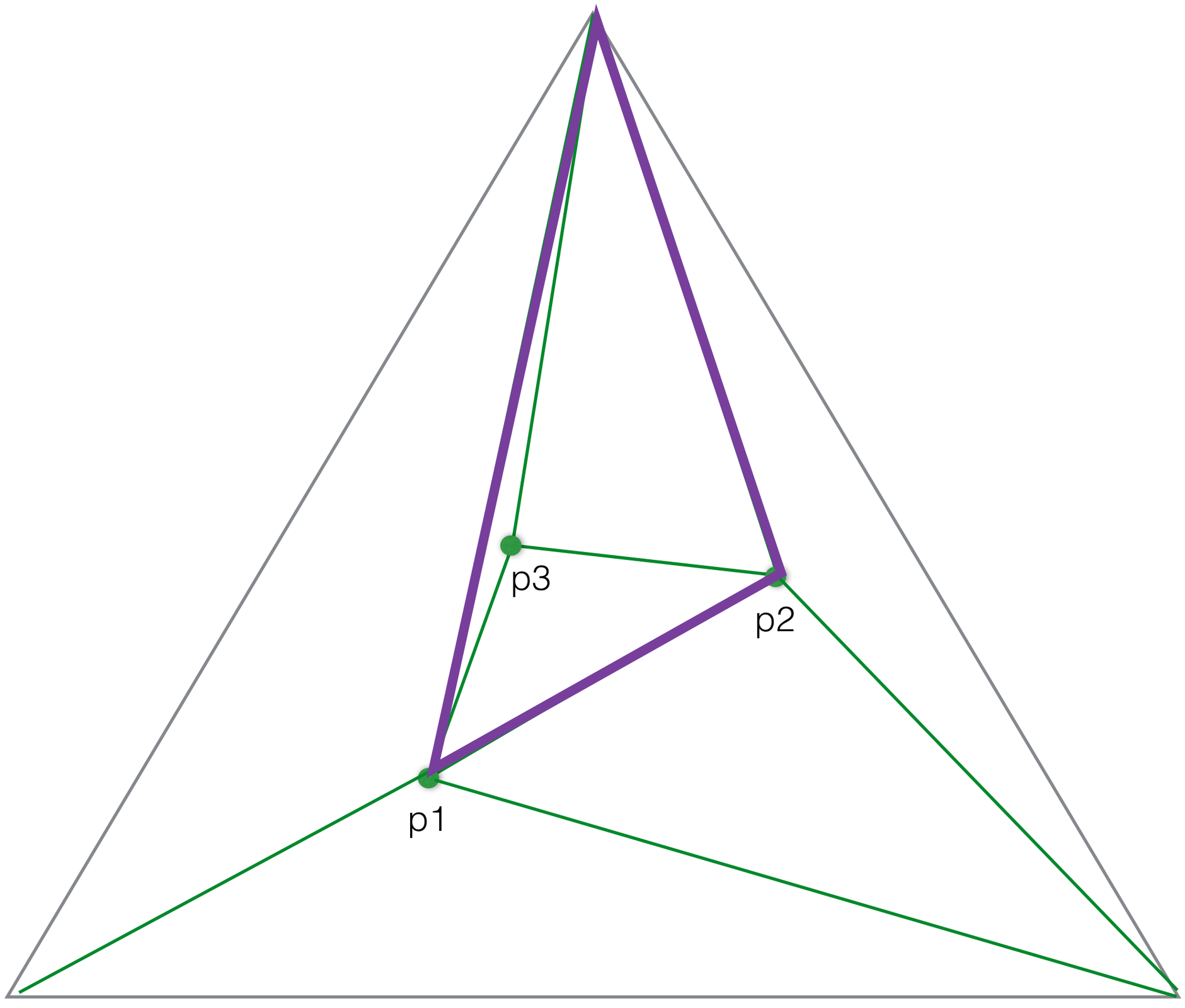
boundary edges are legal

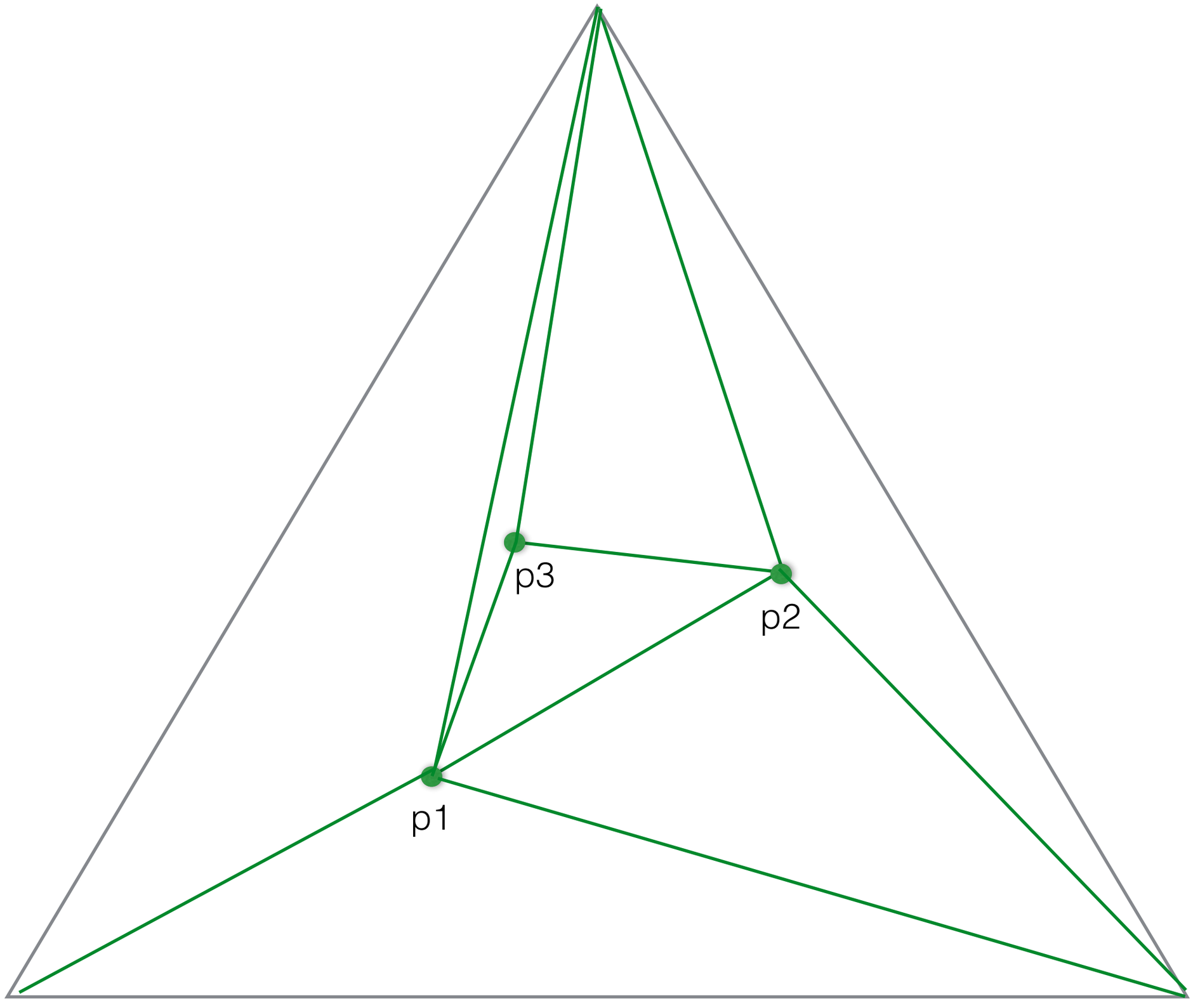

p2
p1

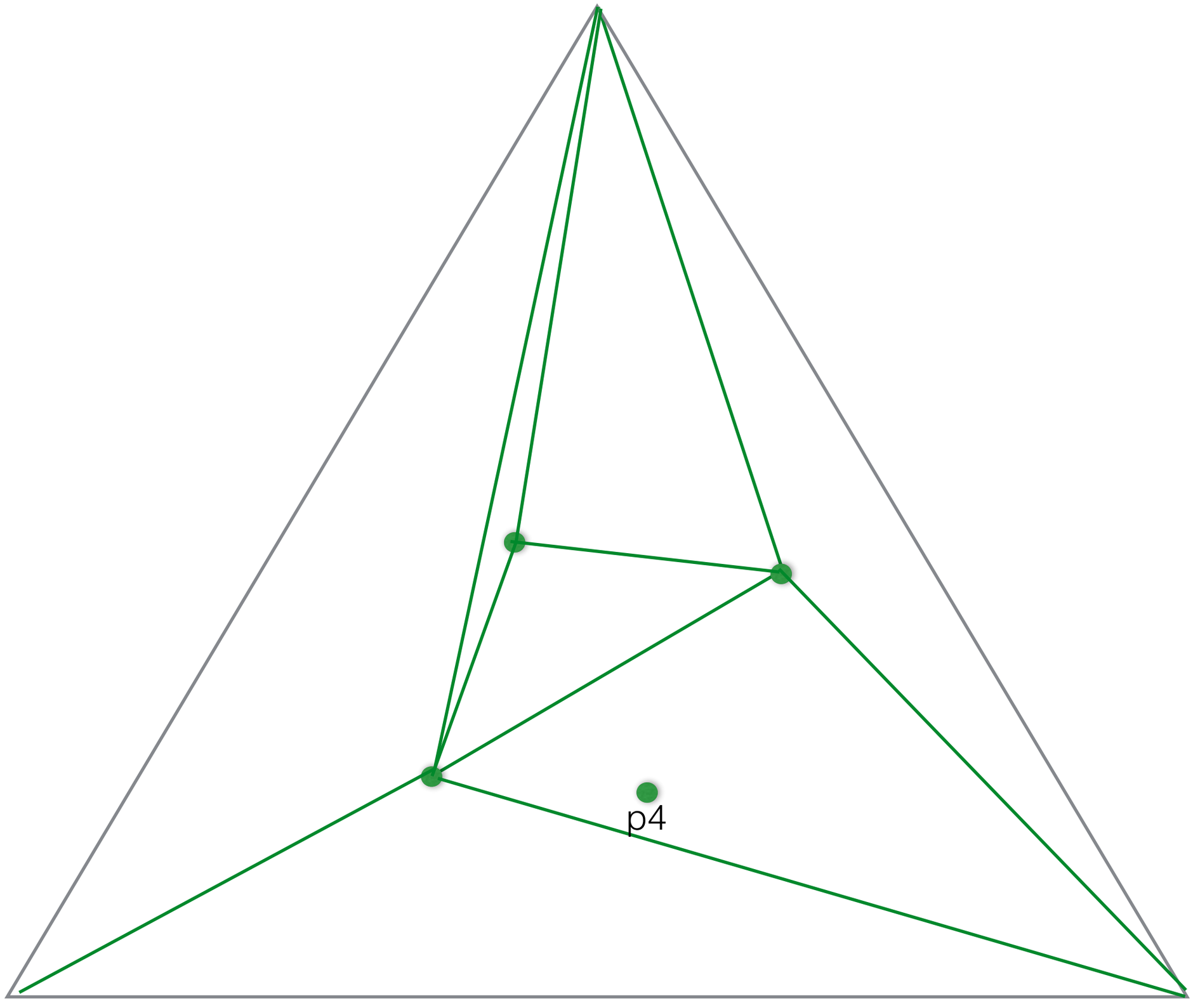

p2
p1

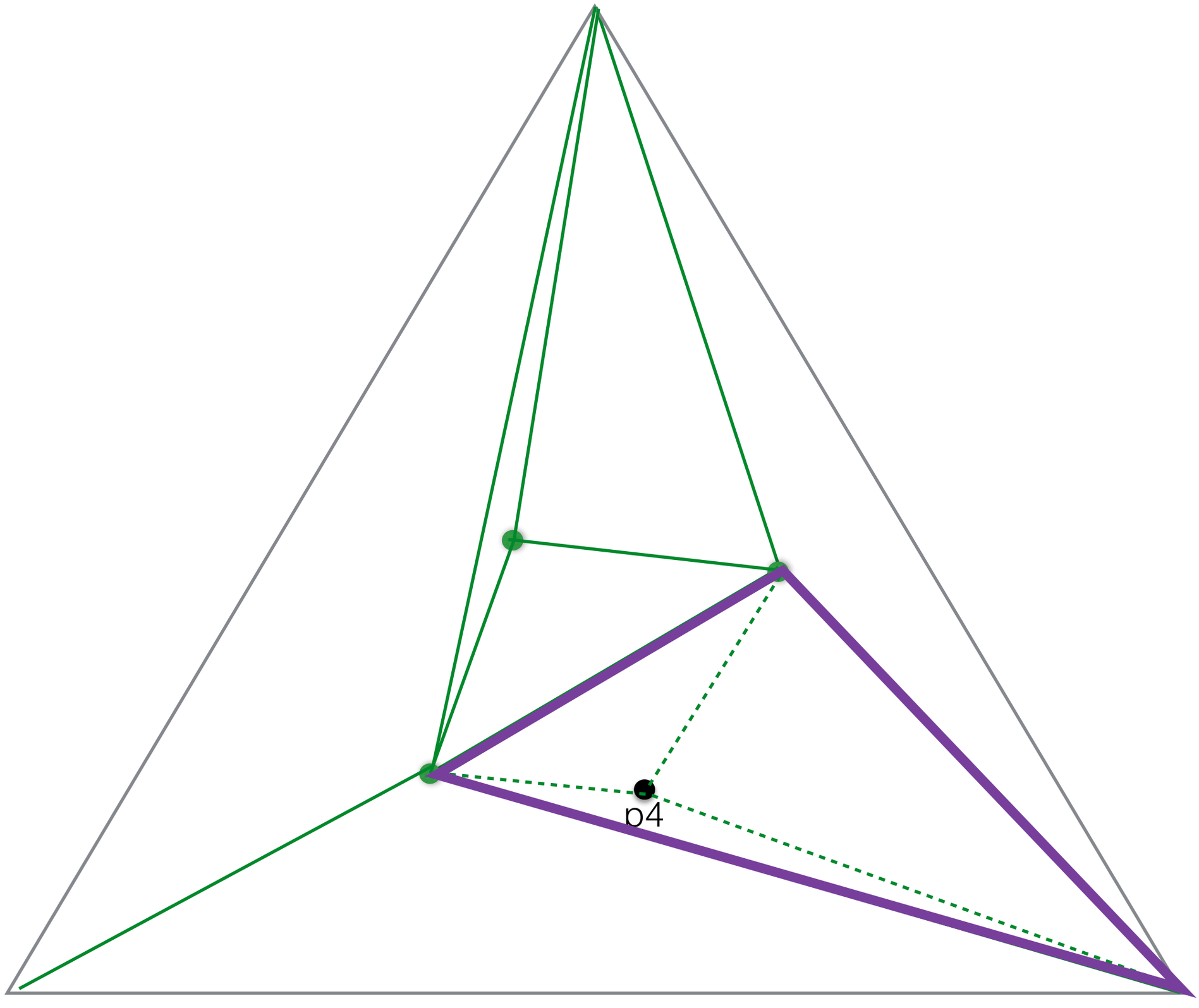

p2
p1

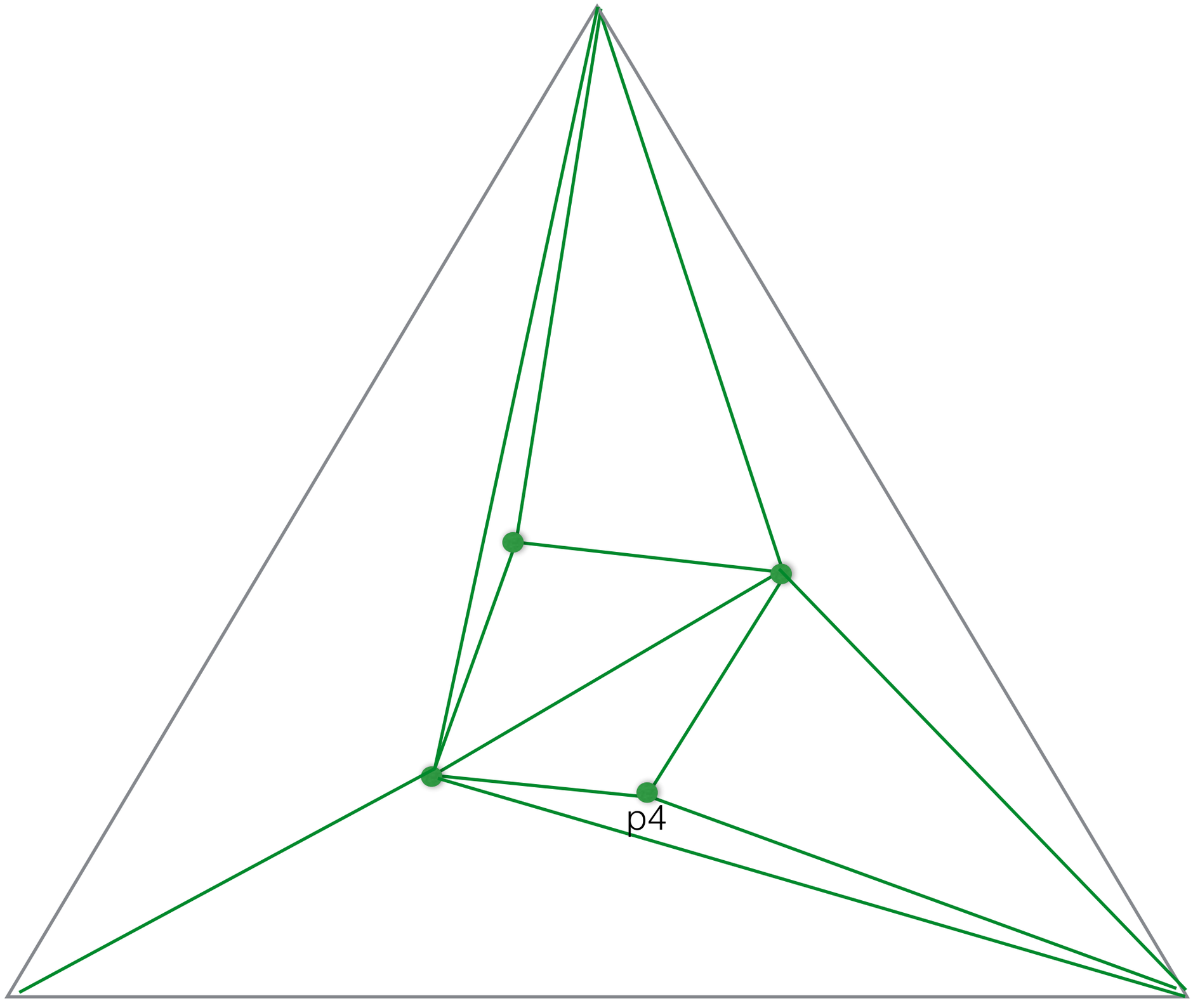

p3
p2
p1

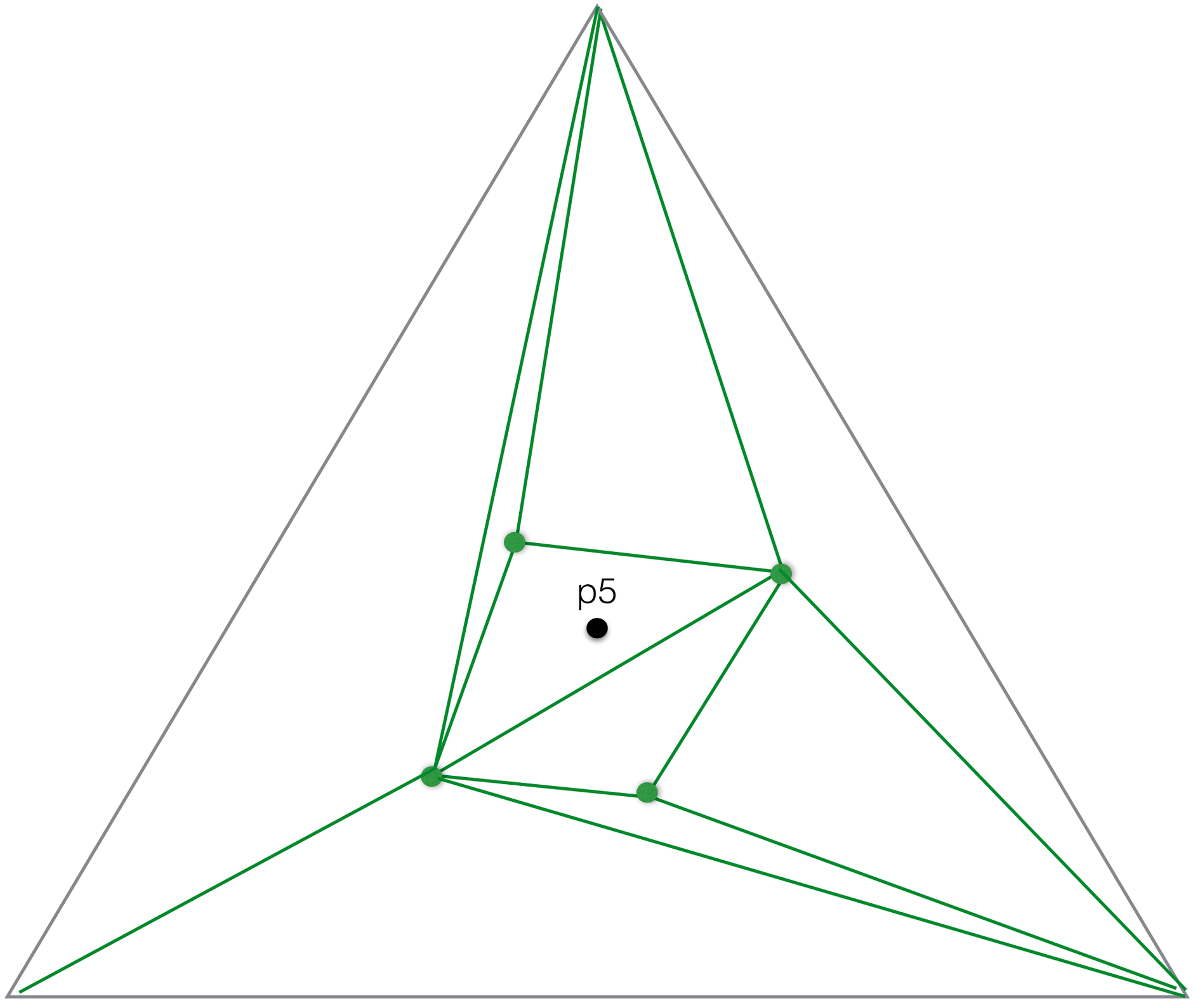

p3
p2
p1

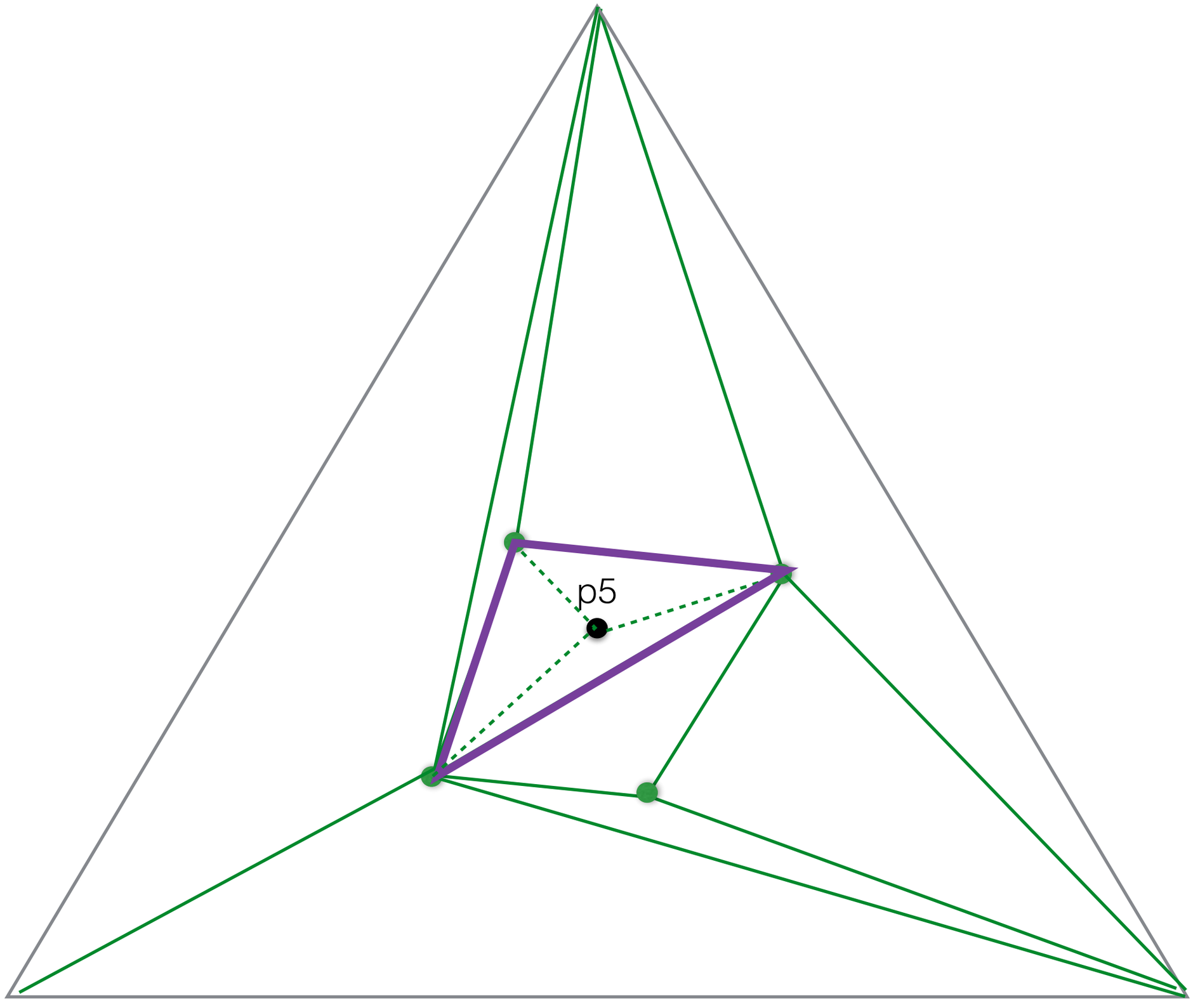

p3
p2
p1

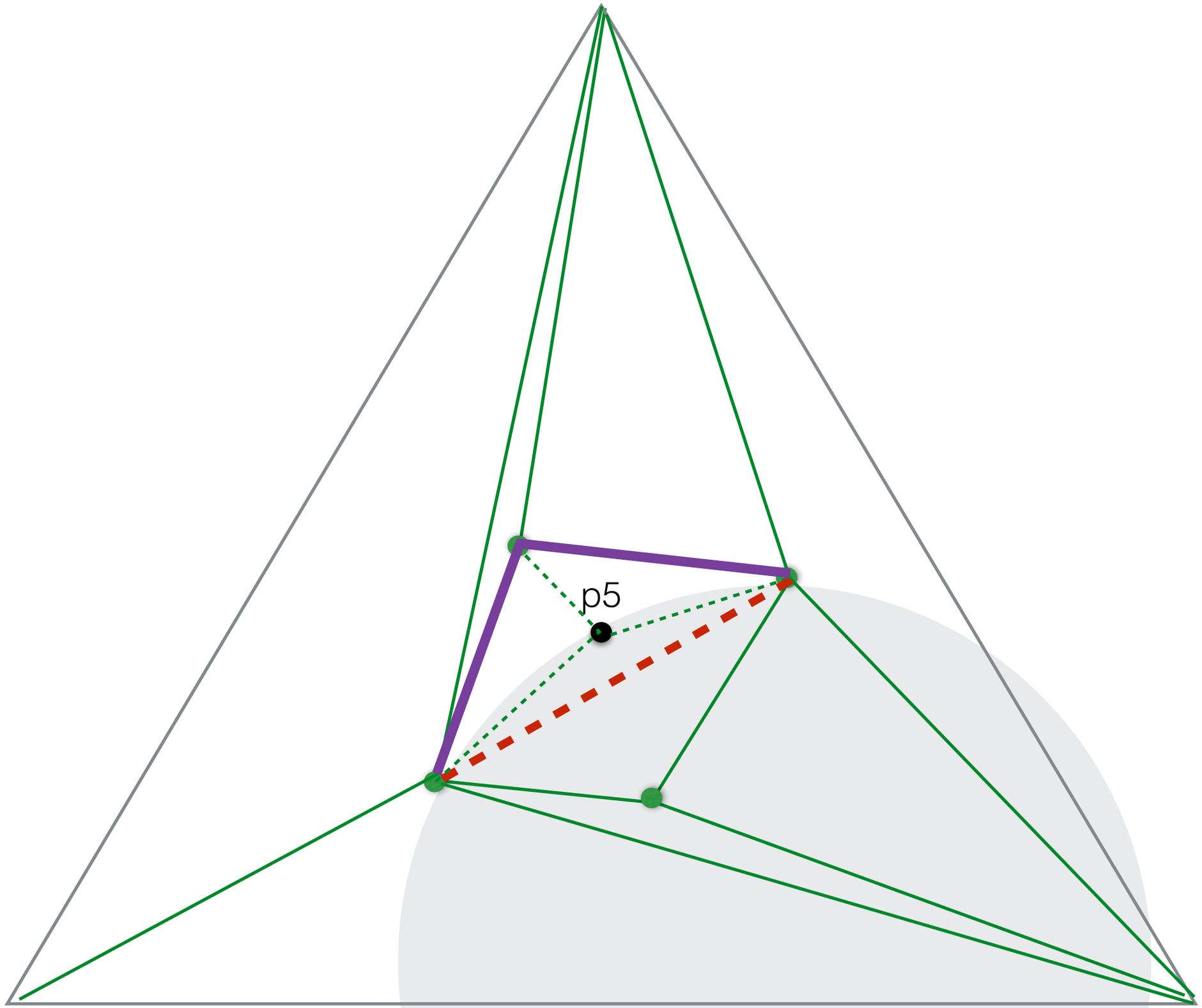

p4

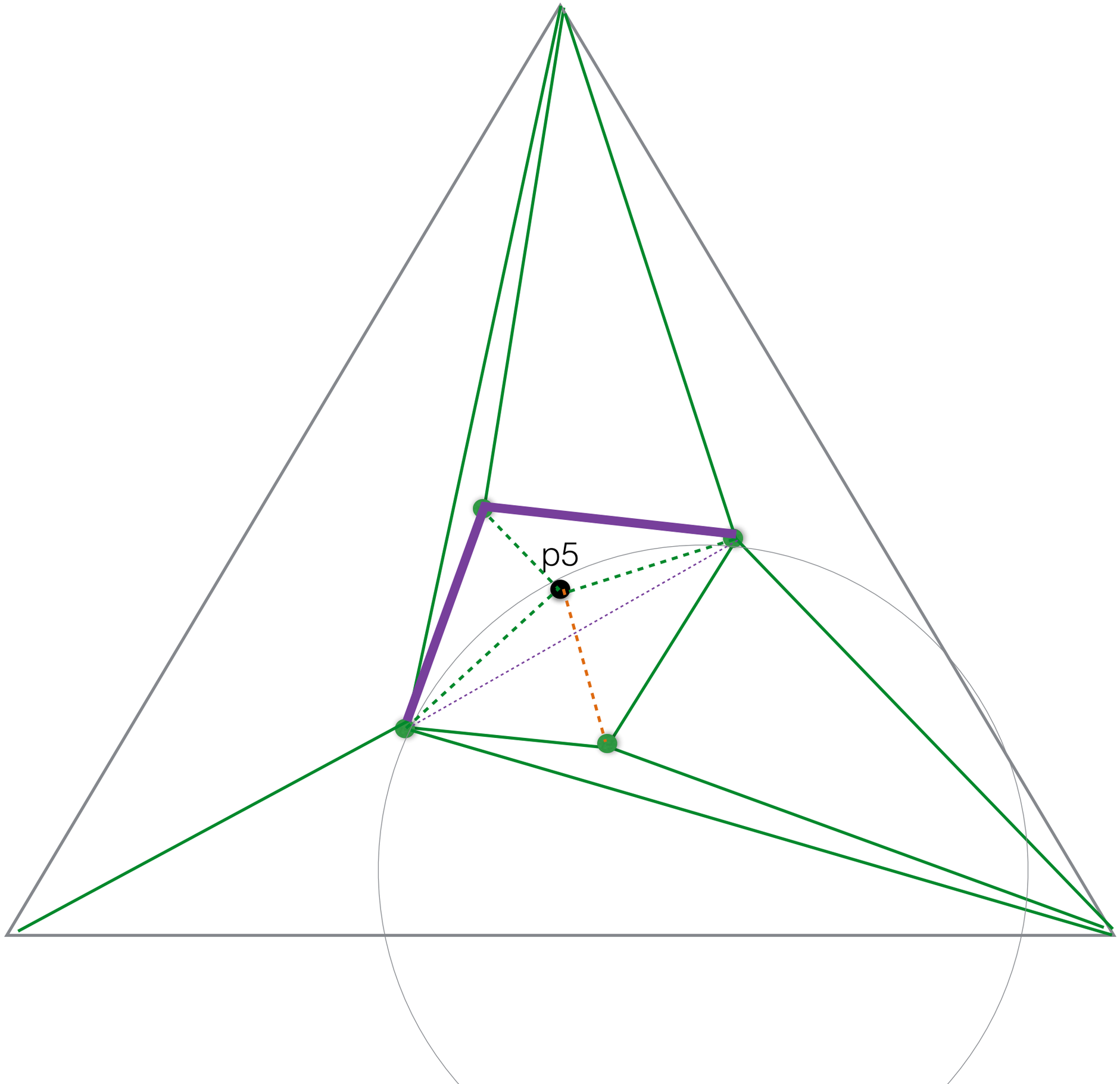

p4

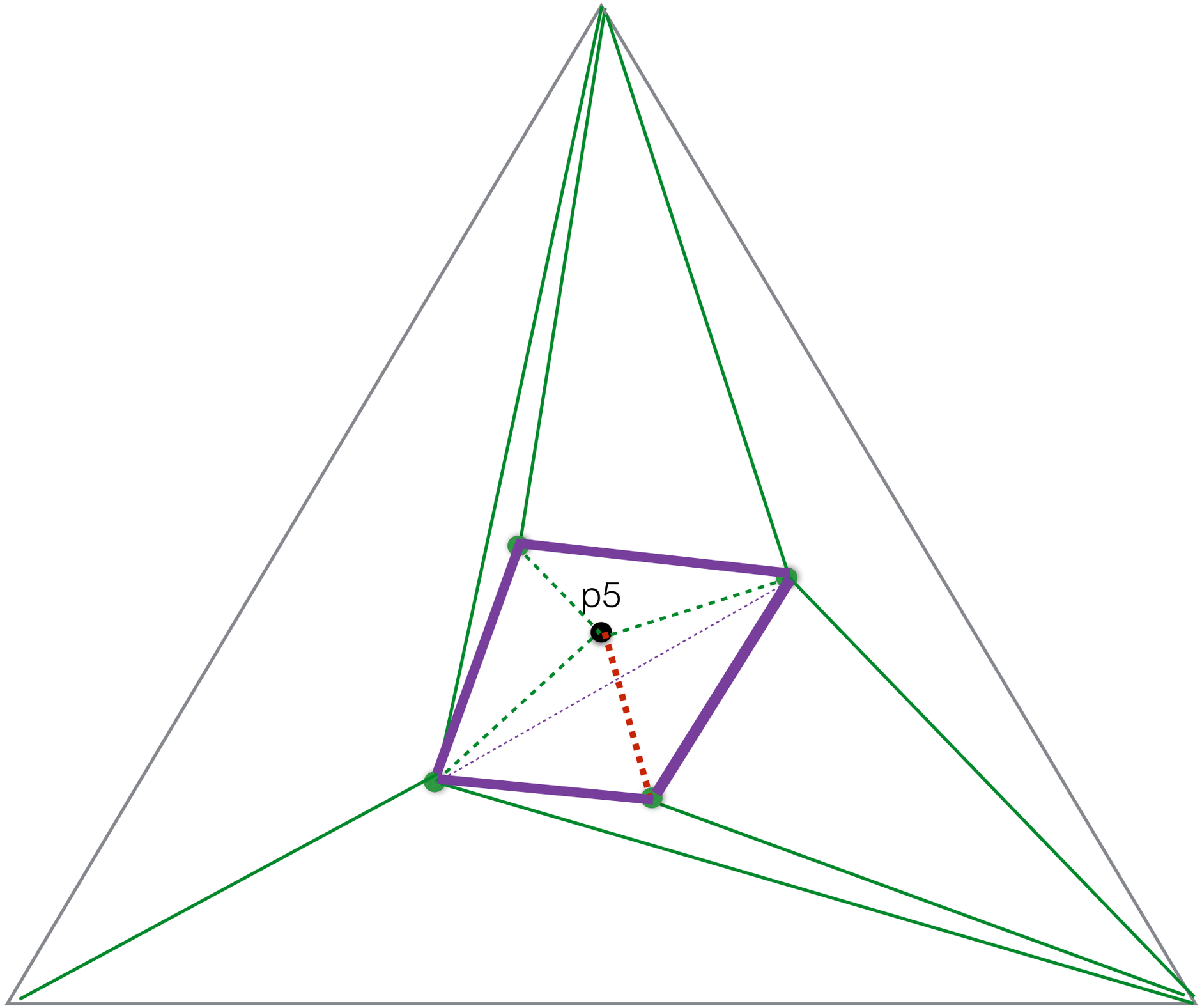

p4

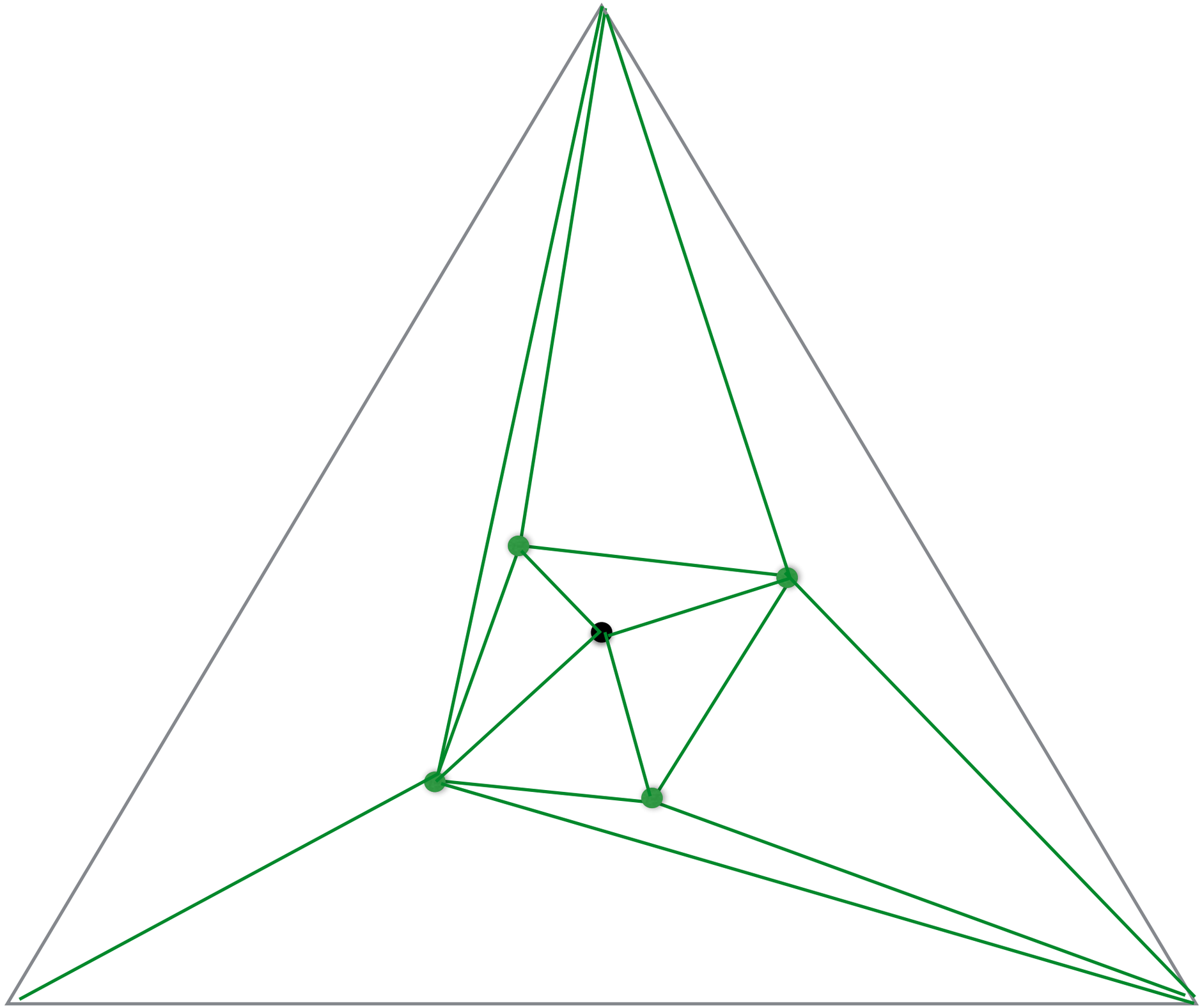

p5

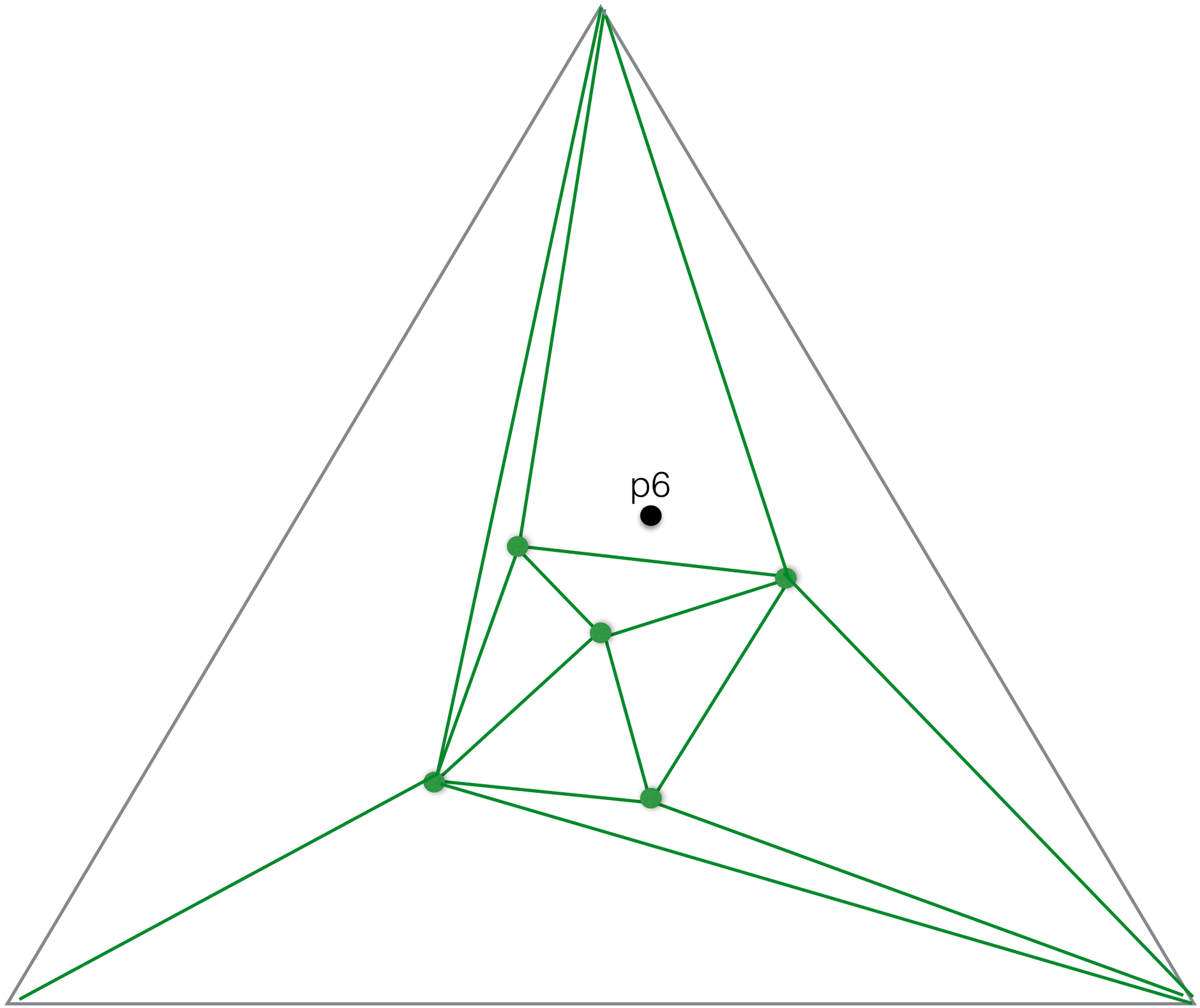

p5

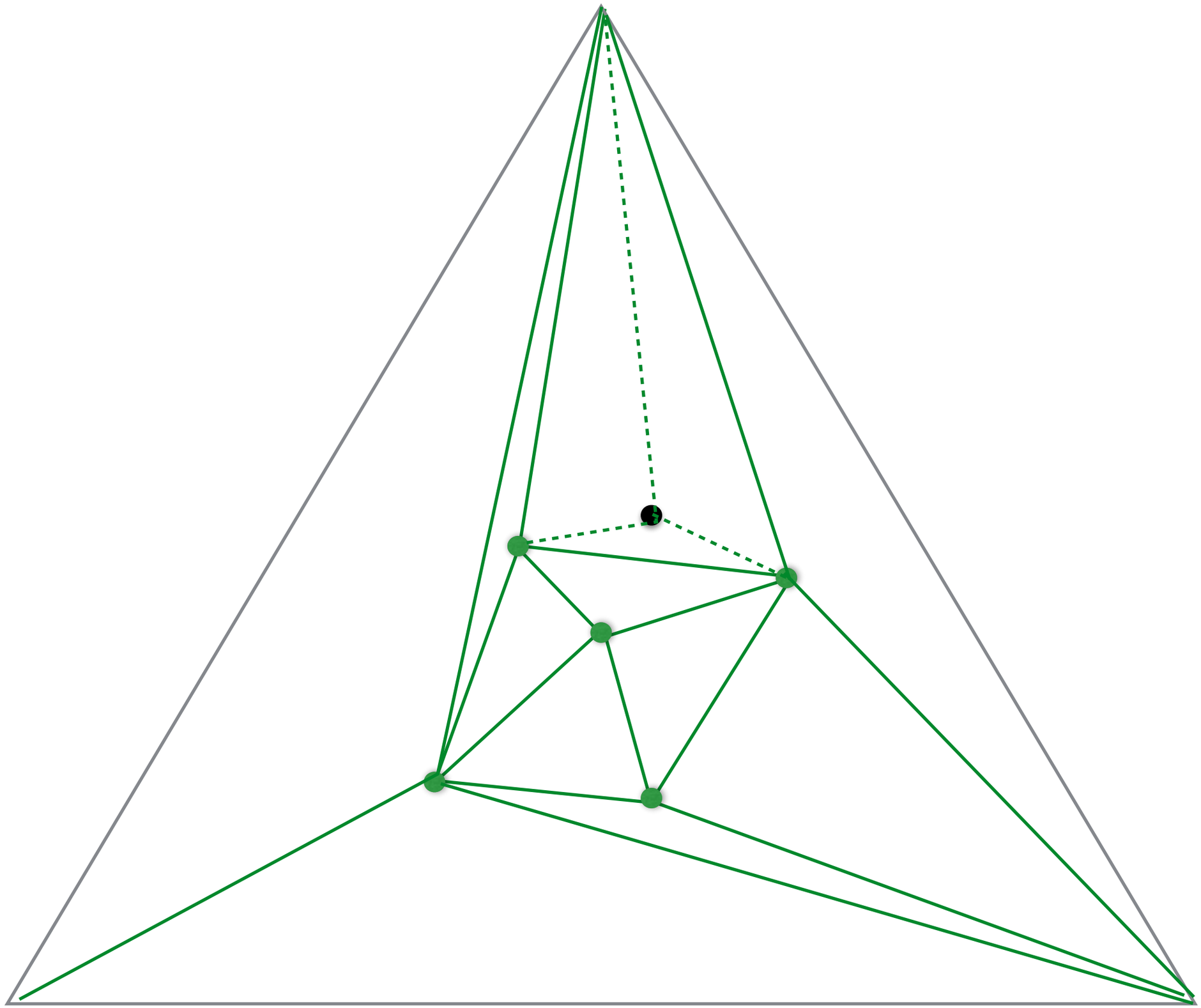

p5

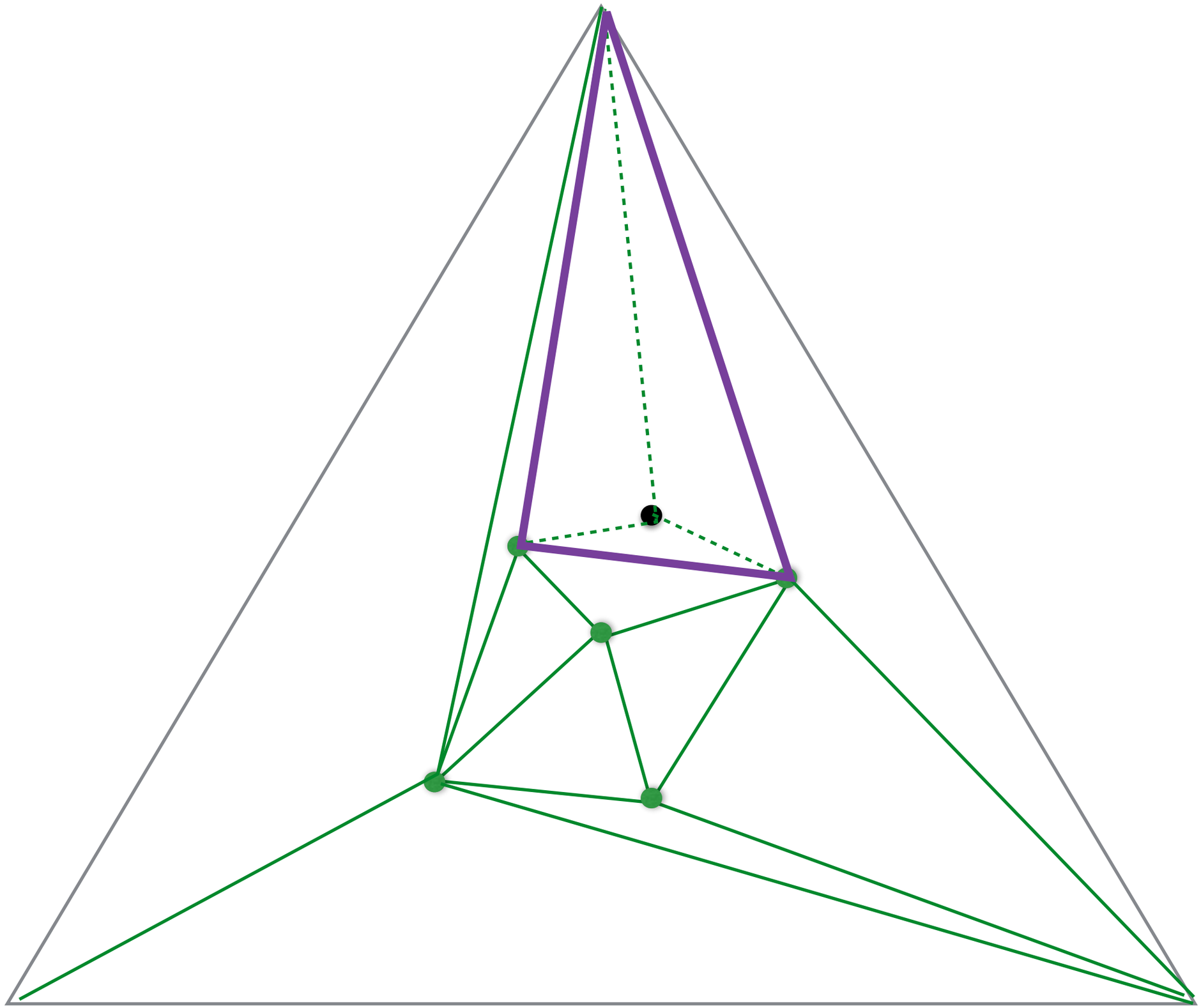

p5

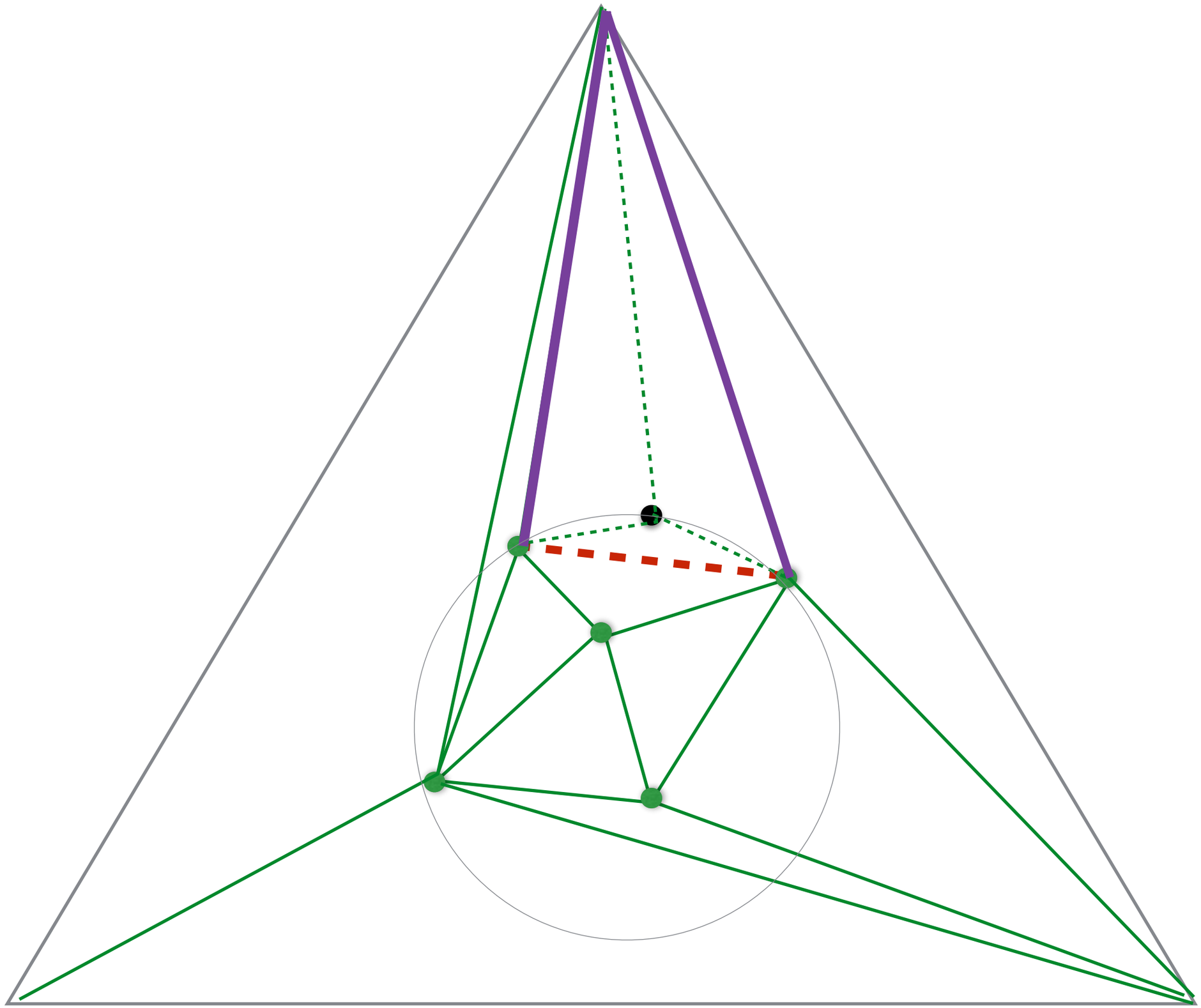

p5

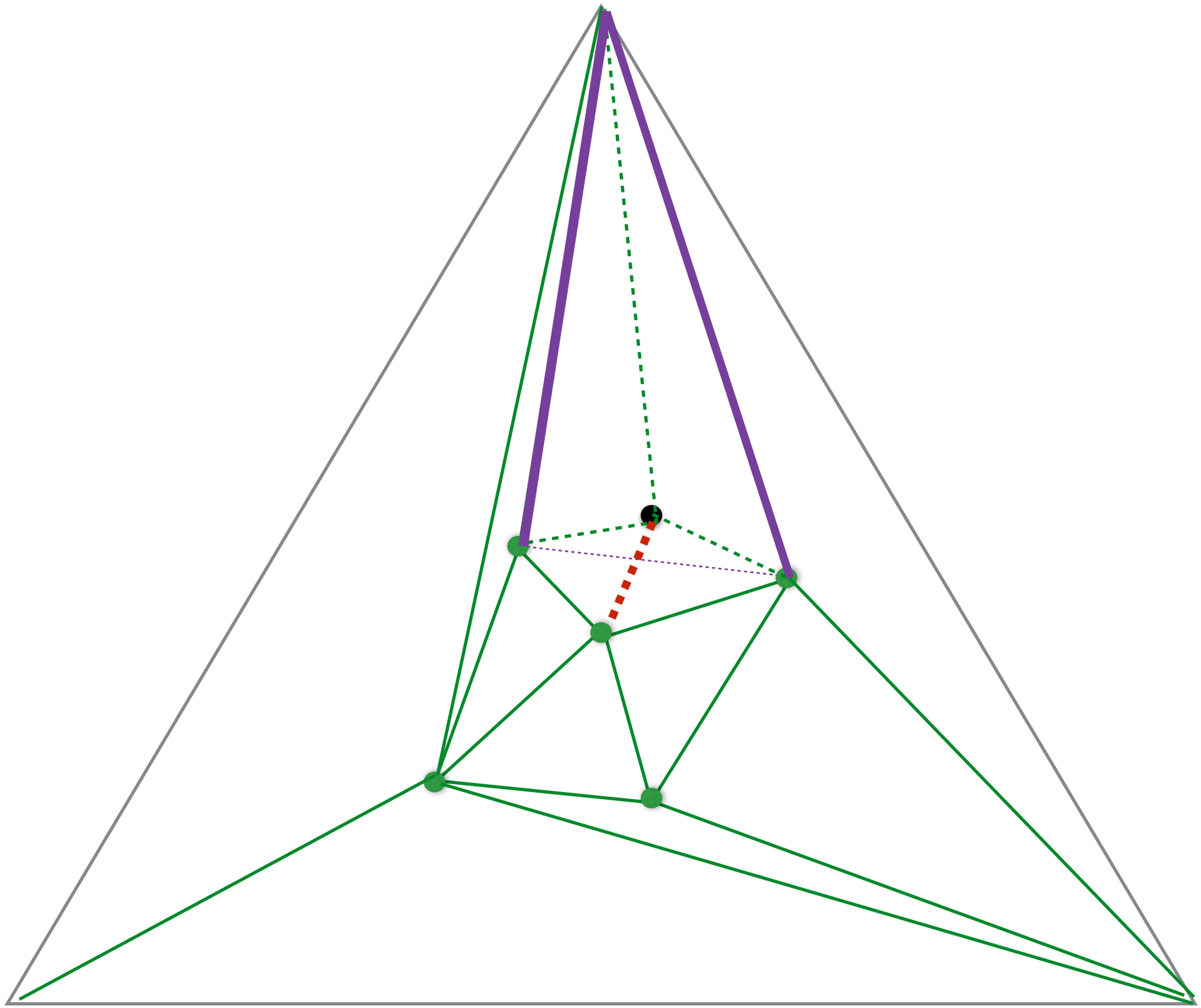


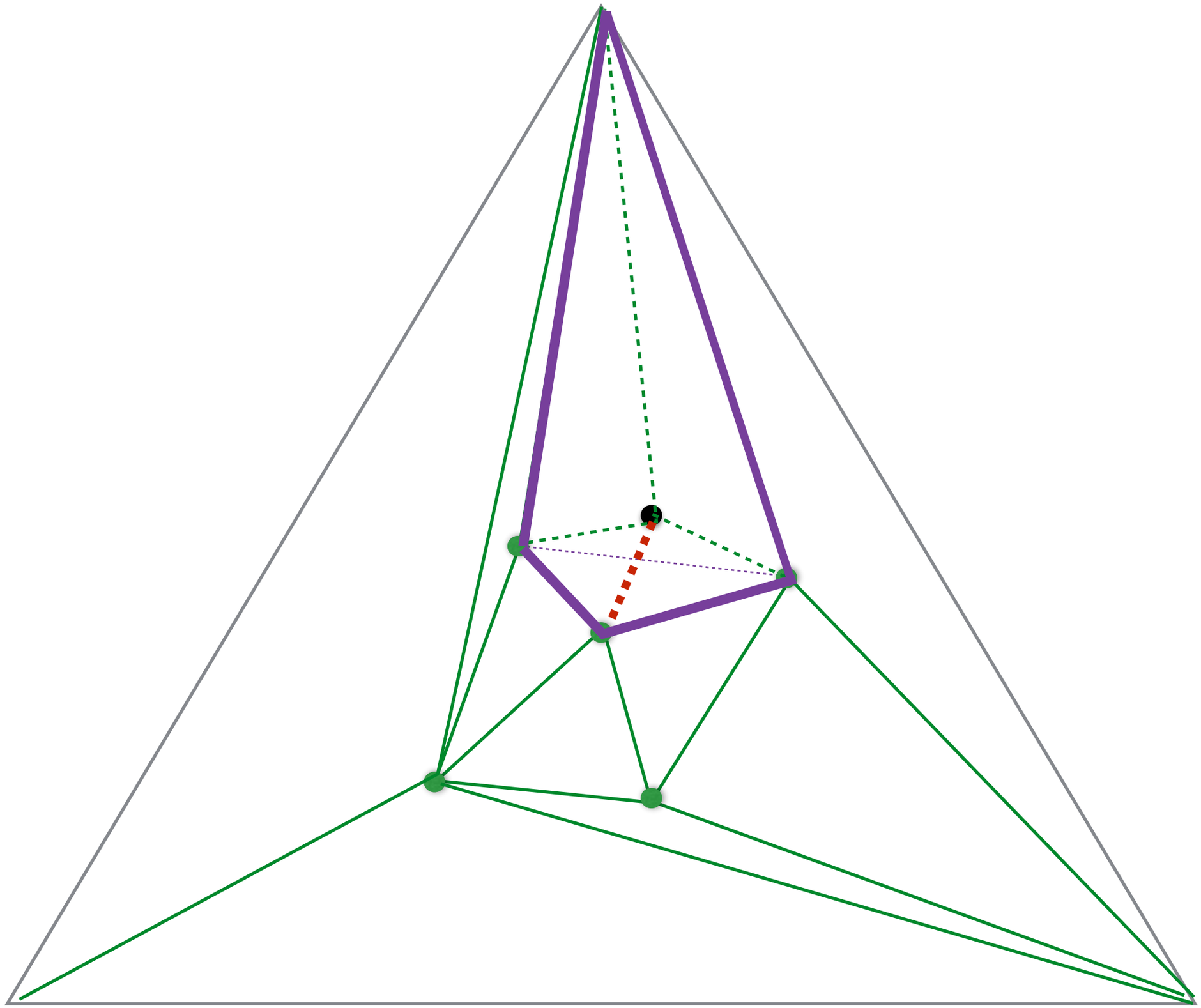

p6

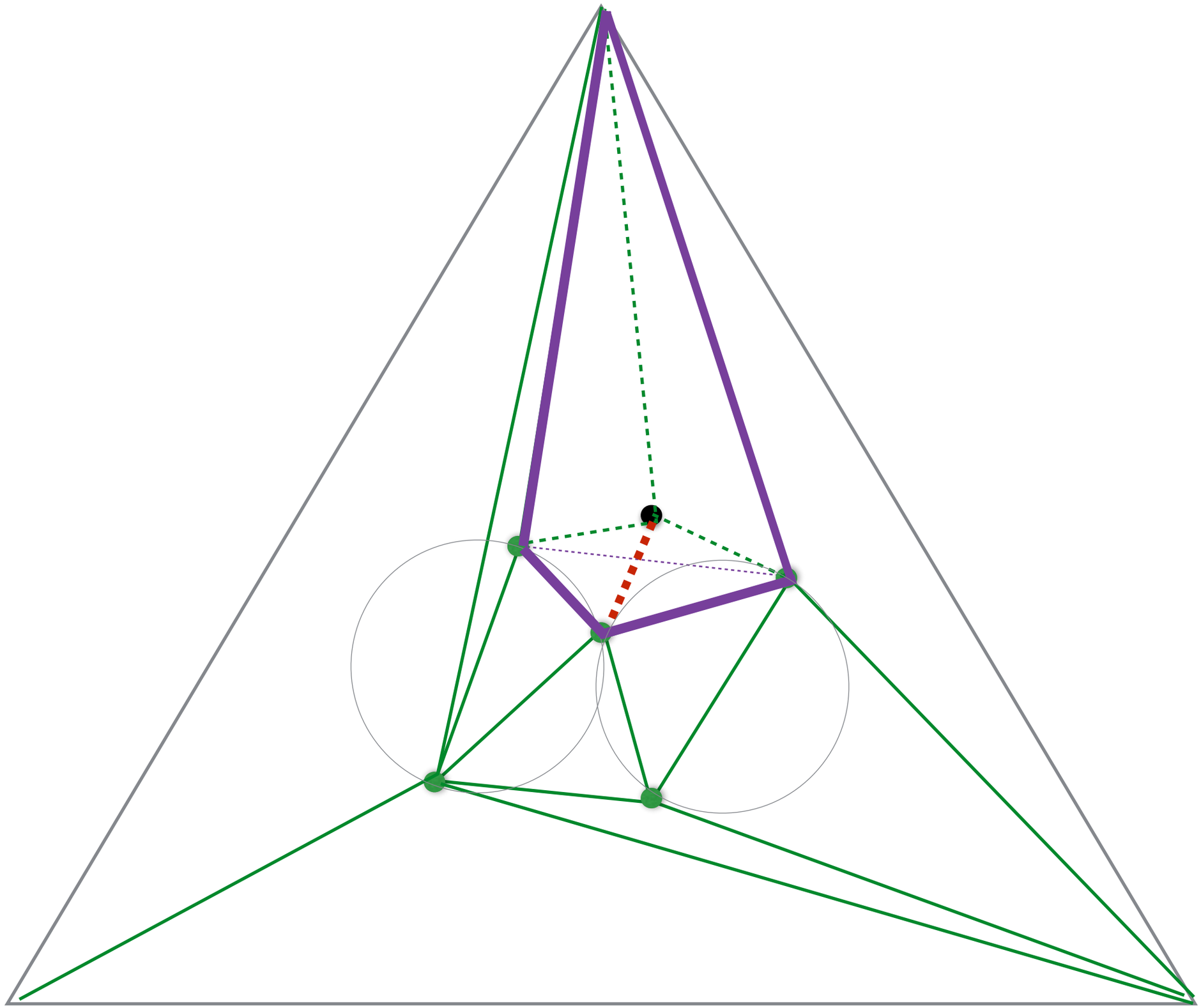


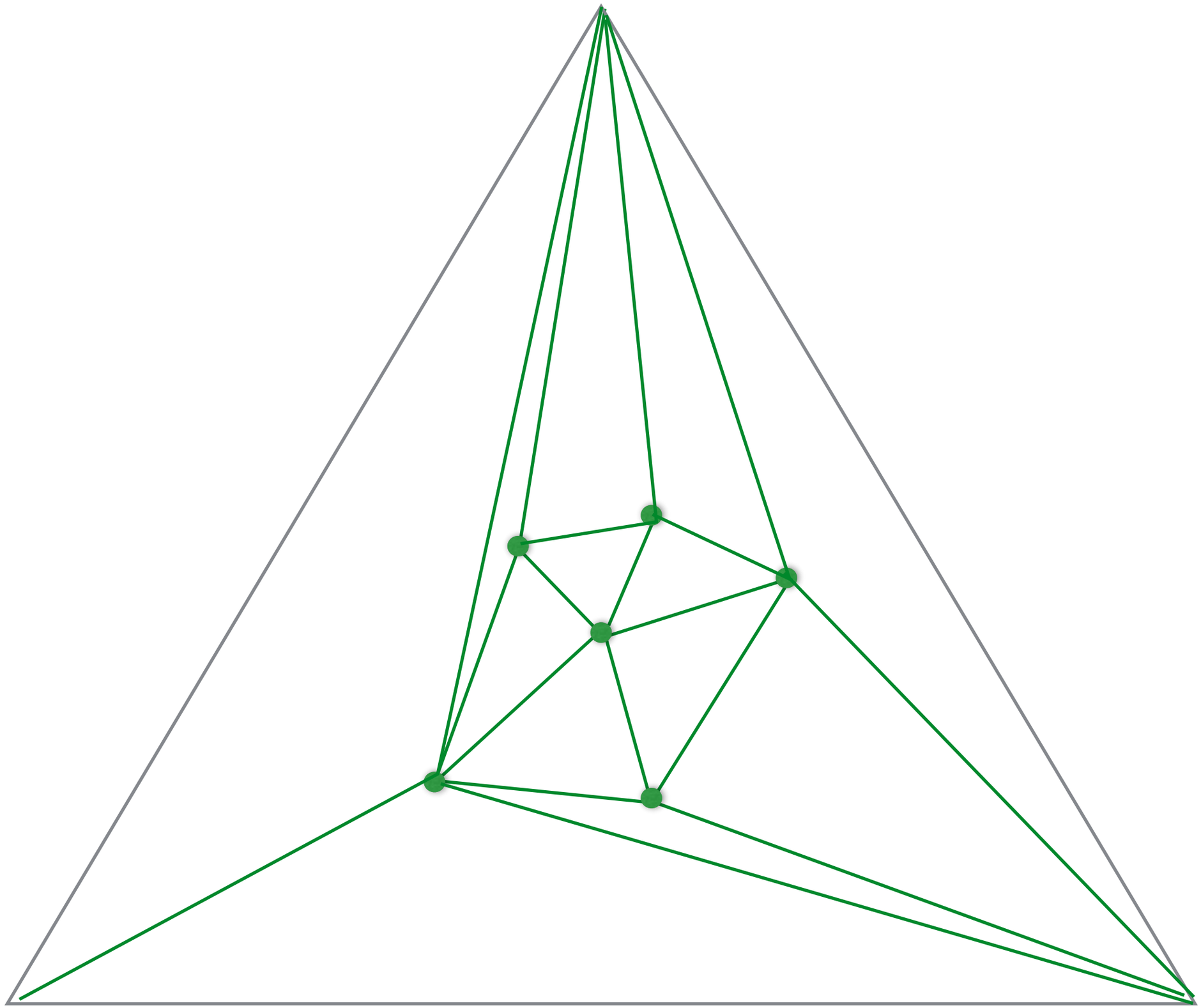


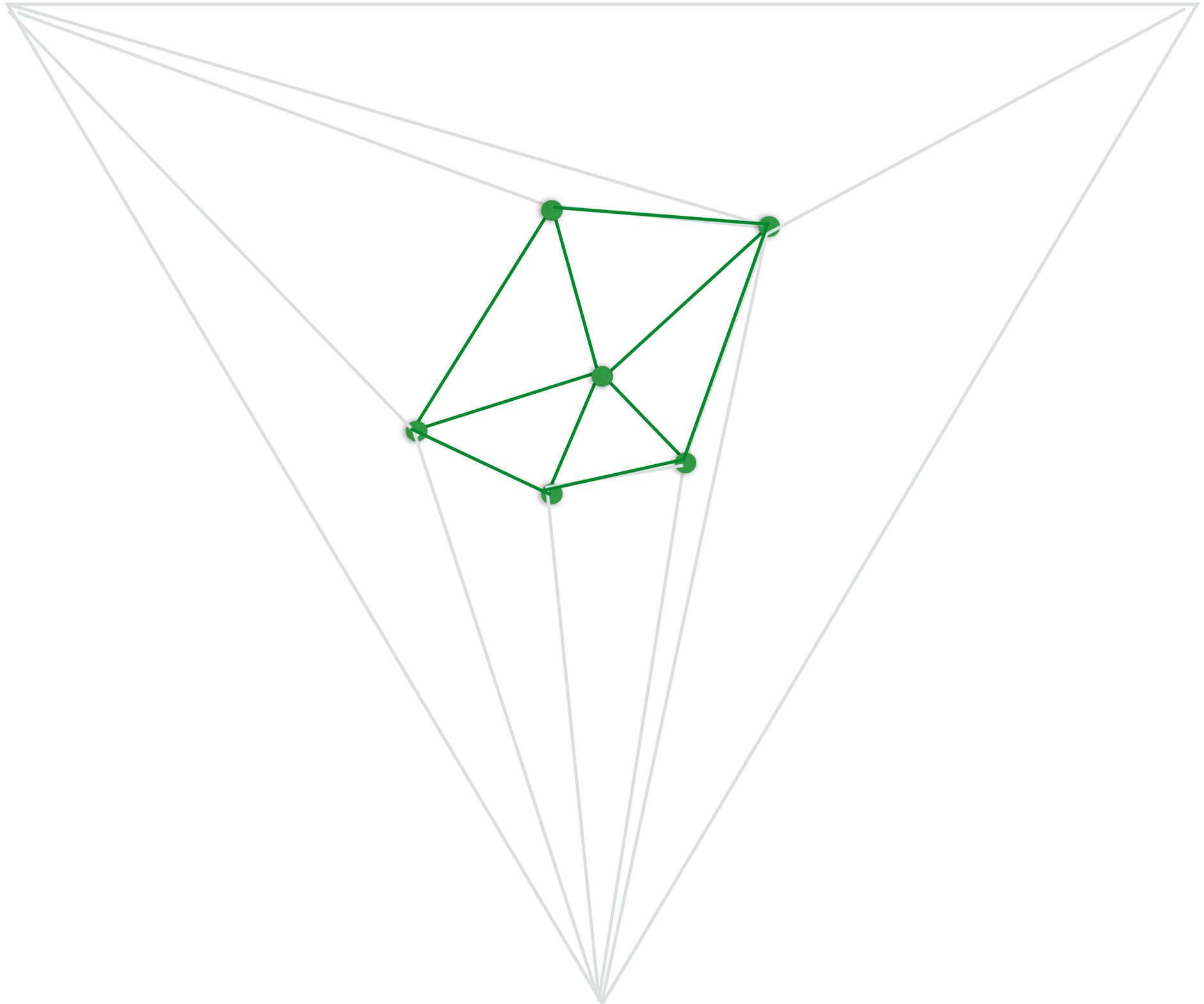












Computing DT via RIC

RIC (DT Randomized Incremental Construction)

- Let (p_1, p_2, p_3, …p_n) be a random permutation of P

  //Let T be the current triangulation

- Initialize T as a large triangle that contains P.
- For next point p_i in P do:

  //insert p_i in T

  - find triangle abc of T that contains p_i
  - splitting into 3 triangles: abp_i, bcp_i, cap_i and update T
  - LegalizeEdge(p_i, ab, T)
  - LegalizeEdge(p_i, bc, T)
  - LegalizeEdge(p_i, ca, T)
- Discard the initial triangle T and its edges
- return T

The usual questions:
Is it correct?
Running time?

LegalizeEdge(p, uv, T)

//point p was inserted, and puv is a new triangle; edge uv may need to be flipped

- let uvq be the triangle adjacent to uv, on the other side of p
- if uv is illegal
  - flip uv and update T
  - LegalizeEdge(p, uq, T)
  - LegalizeEdge(p, qv, T)

RIC Running time

RIC (DT Randomized Incremental Construction)

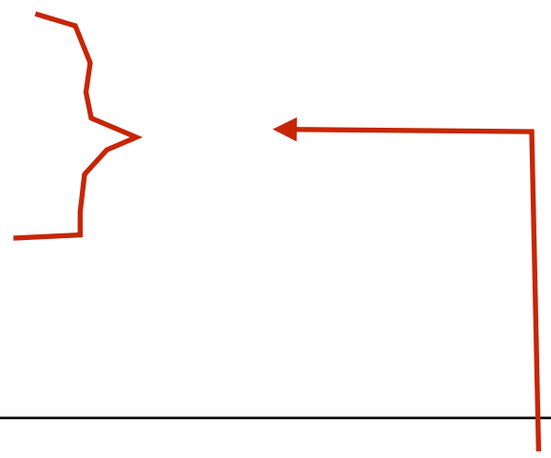
- Let (p_1, p_2, p_3, …p_n) be a random permutation of P

  //Let T be the current triangulation

- Initialize T as a large triangle that contains P.
- For next point p_i in P do:

  //insert p_i in T

  - find triangle abc of T that contains p_i
  - splitting into 3 triangles: abp_i, bcp_i, cap_i and update T
  - LegalizeEdge(p_i, ab, T)
  - LegalizeEdge(p_i, bc, T)
  - LegalizeEdge(p_i, ca, T)
- Discard the initial triangle T and its edges
- return T



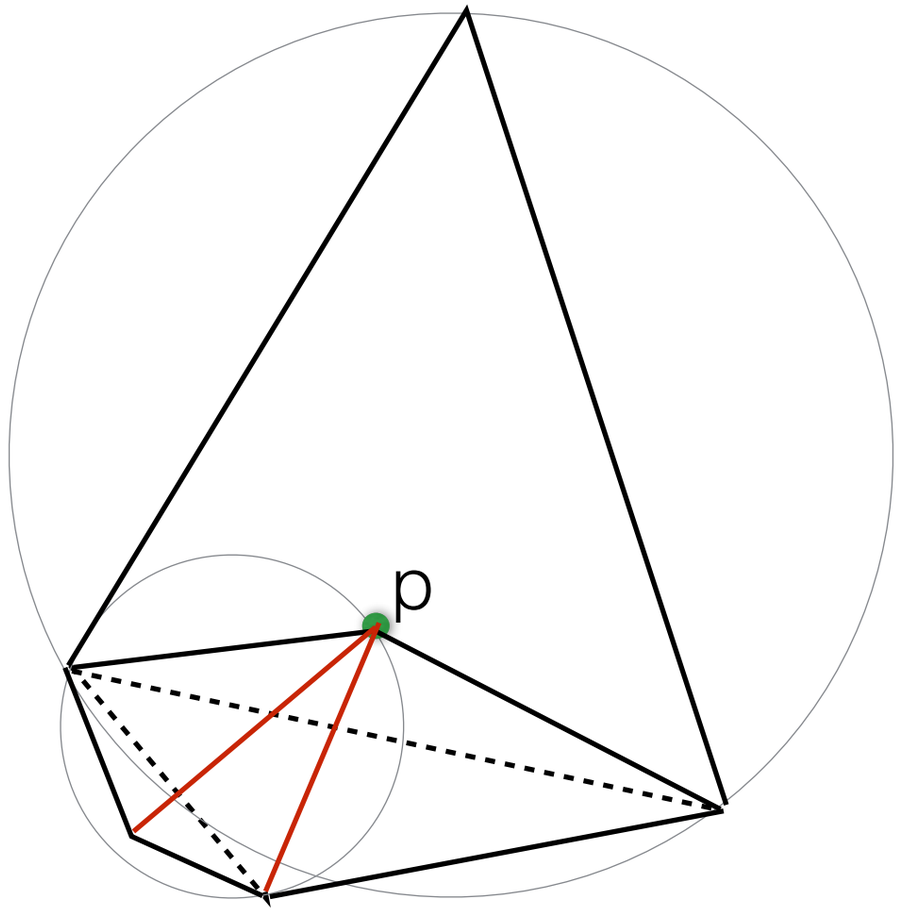
locate triangle that contains p

update T: O(1)

- Inserting p may trigger flipping many edges
- Flipping an edge takes O(1)

Overall: O(n lg n) expected

How many edge flips per point?



- Could be many, but
- Each edge flip creates an edge incident to p.

- Total nb. edge flips when inserting p takes time proportional to the degree of p in the triangulation at that time; if p is the i-th point inserted, this can be O(i) in the worst case,
- If we are unlucky, we insert vertices in such an order that p always triggers the worst case
- But, the average degree of a vertex in a triangulation is 6 = O(1).
- That’s why we insert the points in random order => the expected time spent in edge flips is O(n)

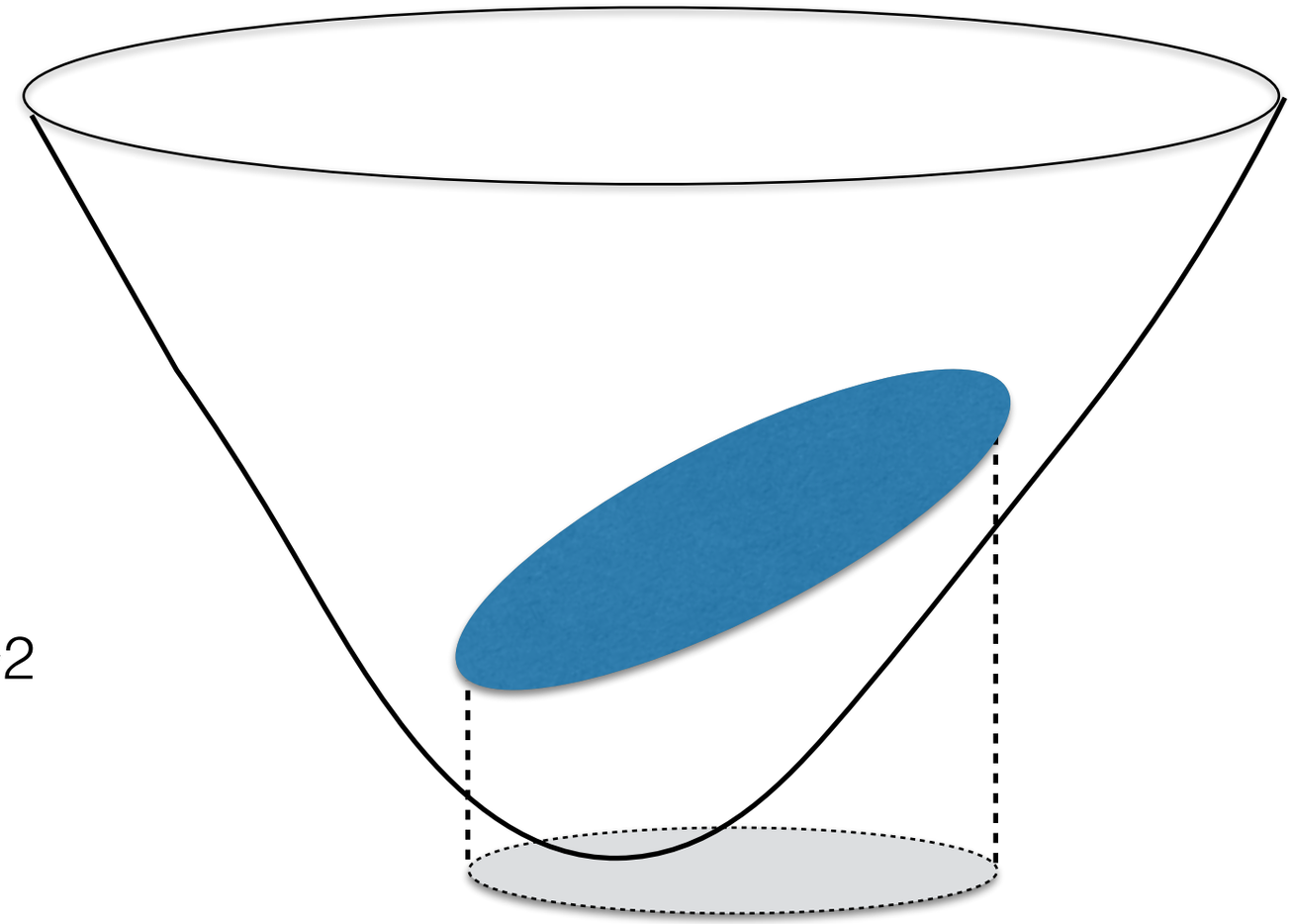
Constructing DT: summary

- O(n lg n) via Voronoi
- $O(n^2)$ via ege flipping
- O(n lg n) expected via Randomized Incremental Construction (RIC)
- O(n lg n) via 3D convex hull (compute 3D CH and project onto xy-plane)

A beautiful connection

DT (2D) <——> Convex Hull (3D)

$z = x^2+y^2$



Claim: A circle in the plane will project onto the paraboloid in an ellipse.

- Let a,b,c,d be points in the plane
- Let a', b', c', d' be their vertical projections onto the paraboloid $z = x^2+y^2$

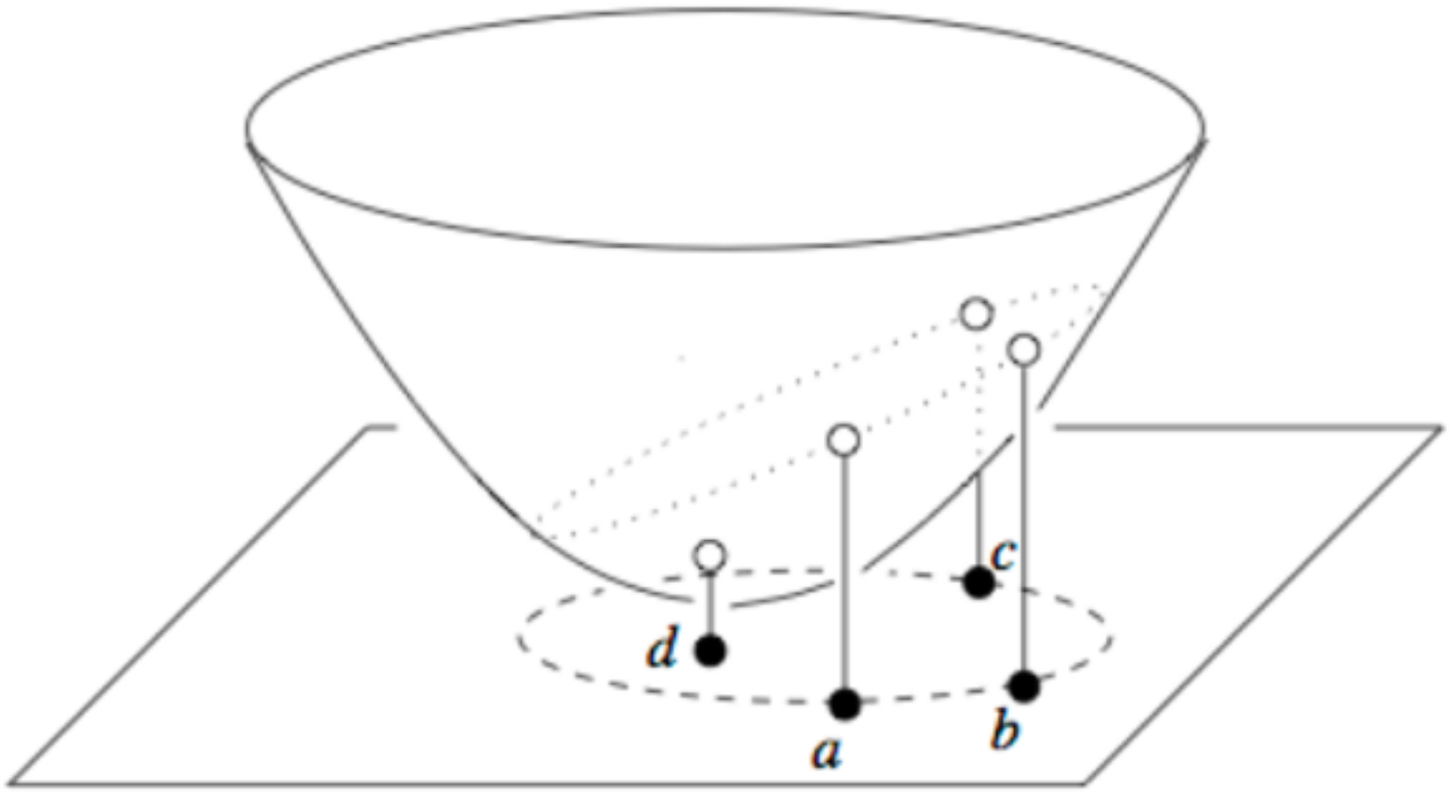
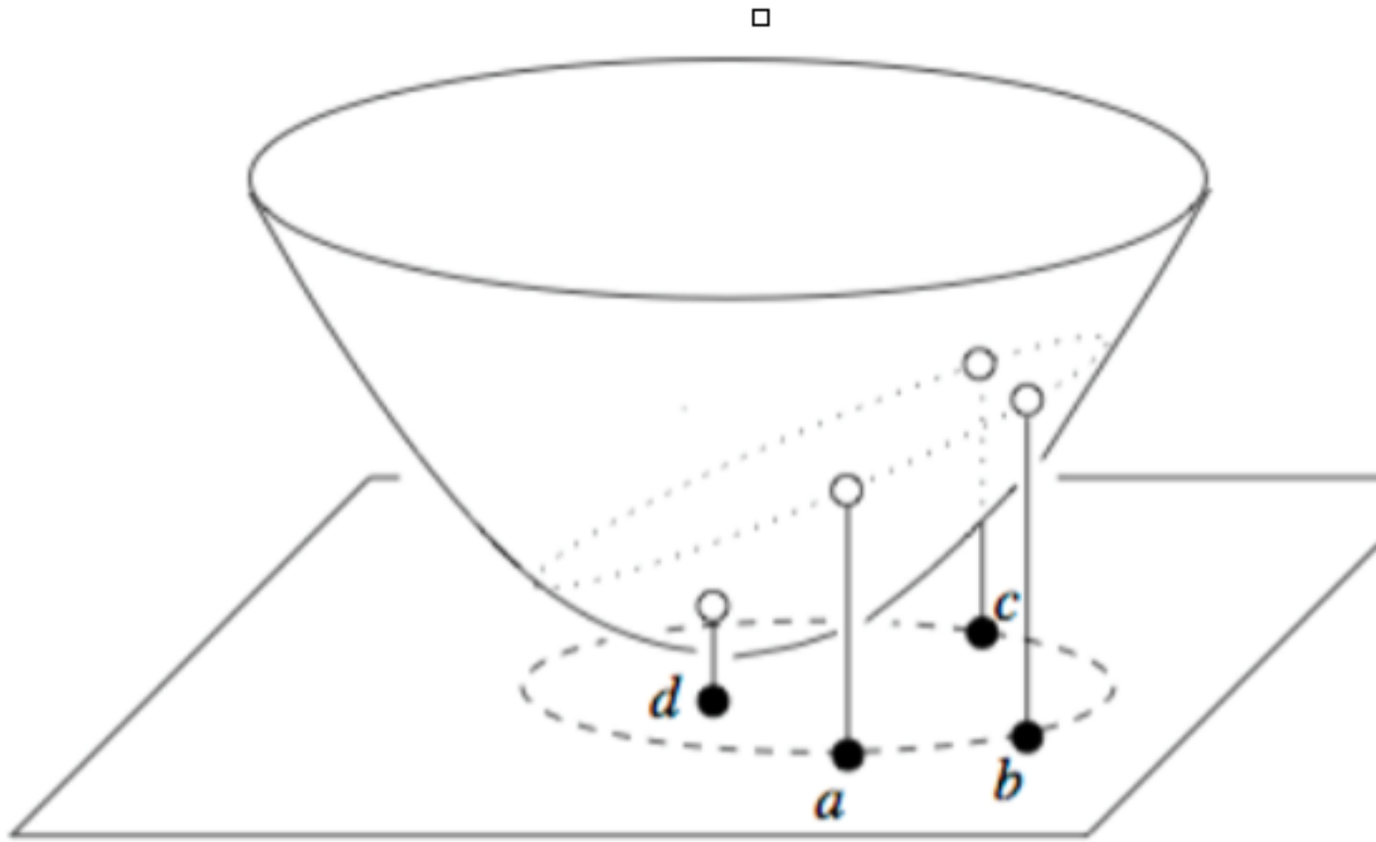


Figure 1.11. Points a, b, c lie on the dashed circle in the x_1x_2-plane and d lies inside that circle. The dotted curve is the intersection of the paraboloid with the plane that passes through $\hat{a}, \hat{b}, \hat{c}$. It is an ellipse whose projection is the dashed circle.

igure from the book *Geometry and Topology for Mesh Generation* by Edelsbrunner.

DT and Lifted Paraboloid

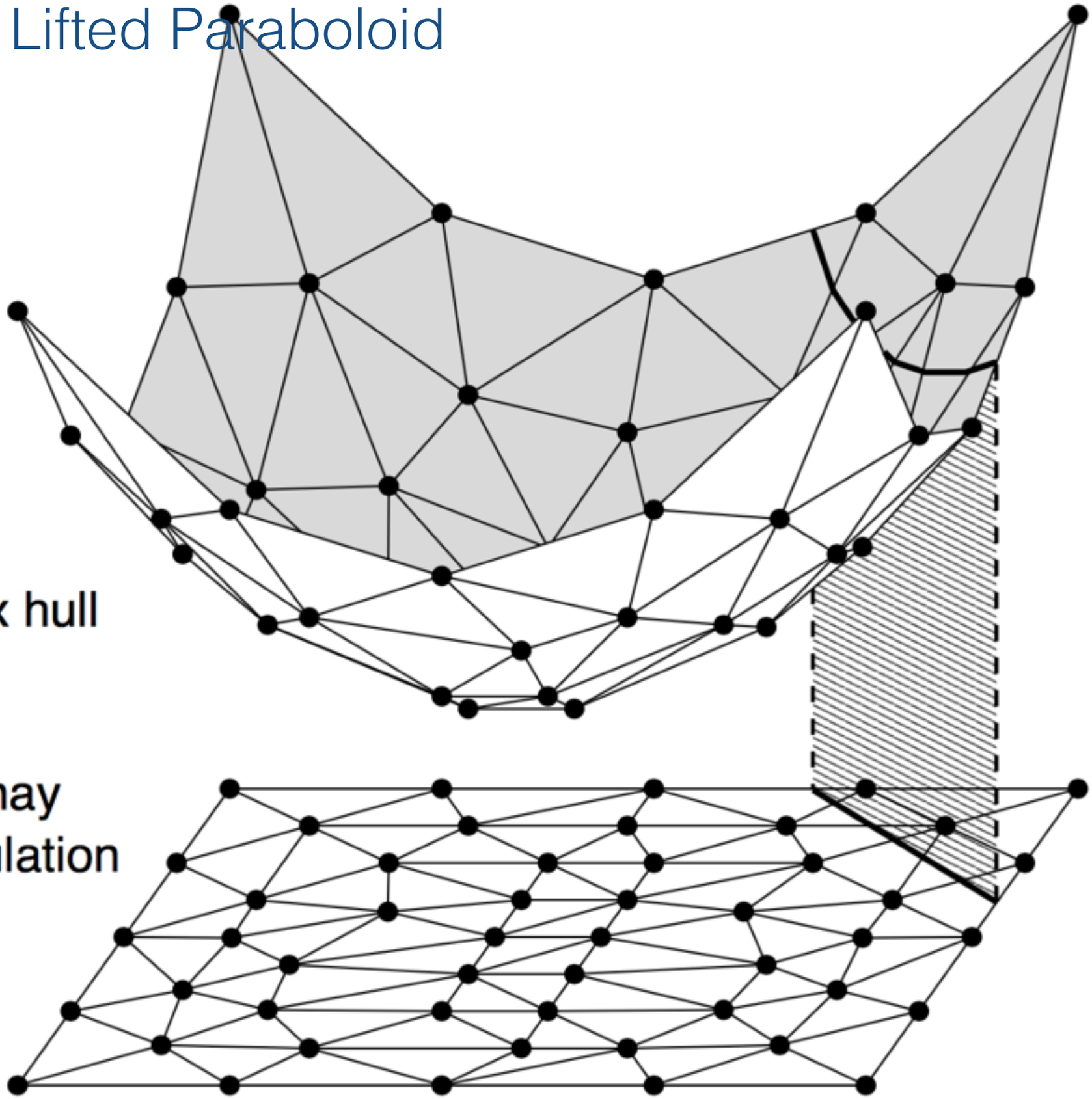


Claim: Point d lies inside C(abc) if and only of point d’ lies vertically below the plane passing through a’b’c’

- DT: There are no points inside triangle (abc)
- On the paraboloid: There are no points below the plane defined by (a’,b’,c’) => triangle a’b’c’ is a face of the 3D hull

DT can be computed by computing the lower 3D hull of points lifted on the paraboloid. The faces of the CH represent the triangles in the DT.

DT and Lifted Paraboloid

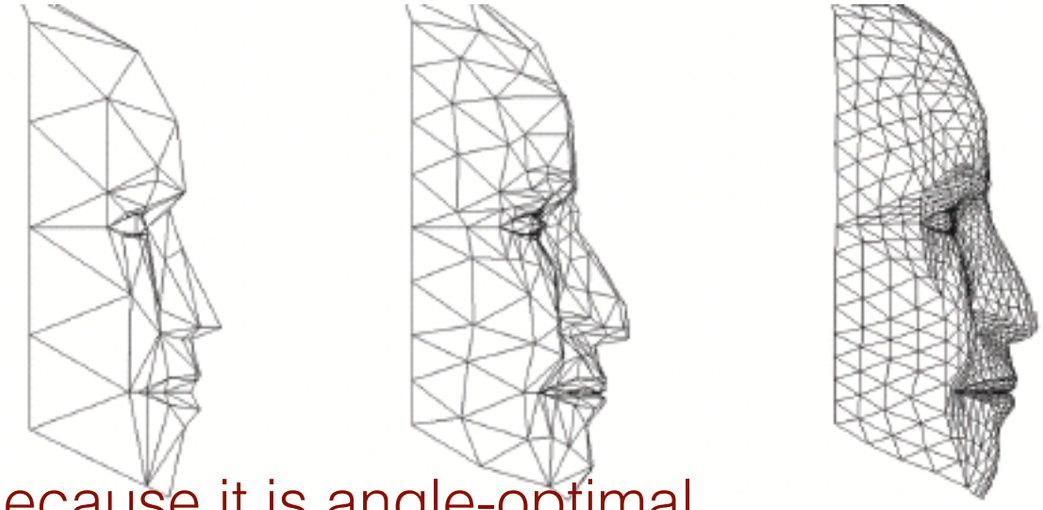


Lower
convex hull

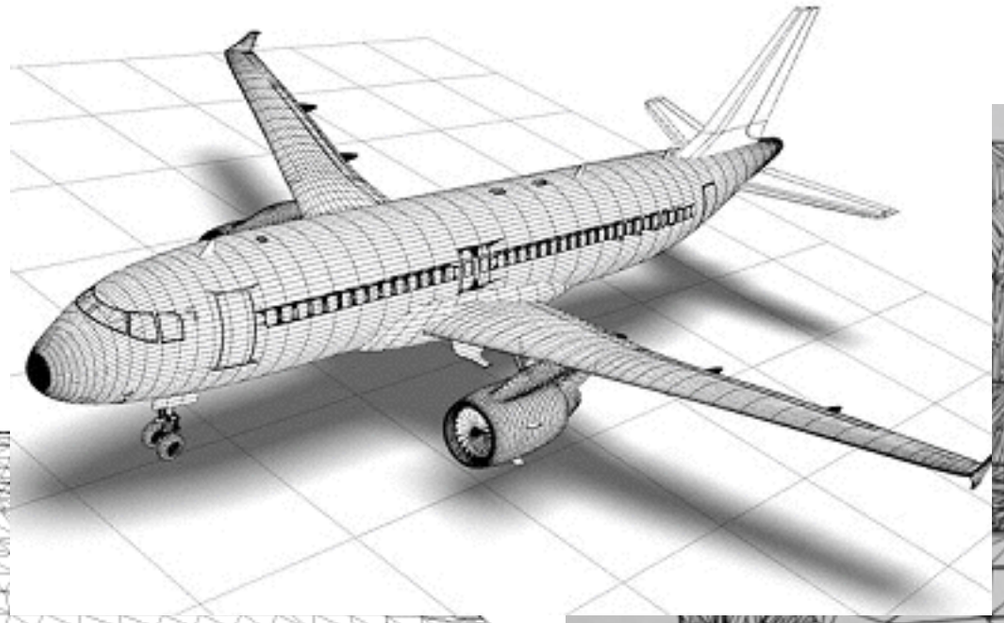
Delaunay
triangulation

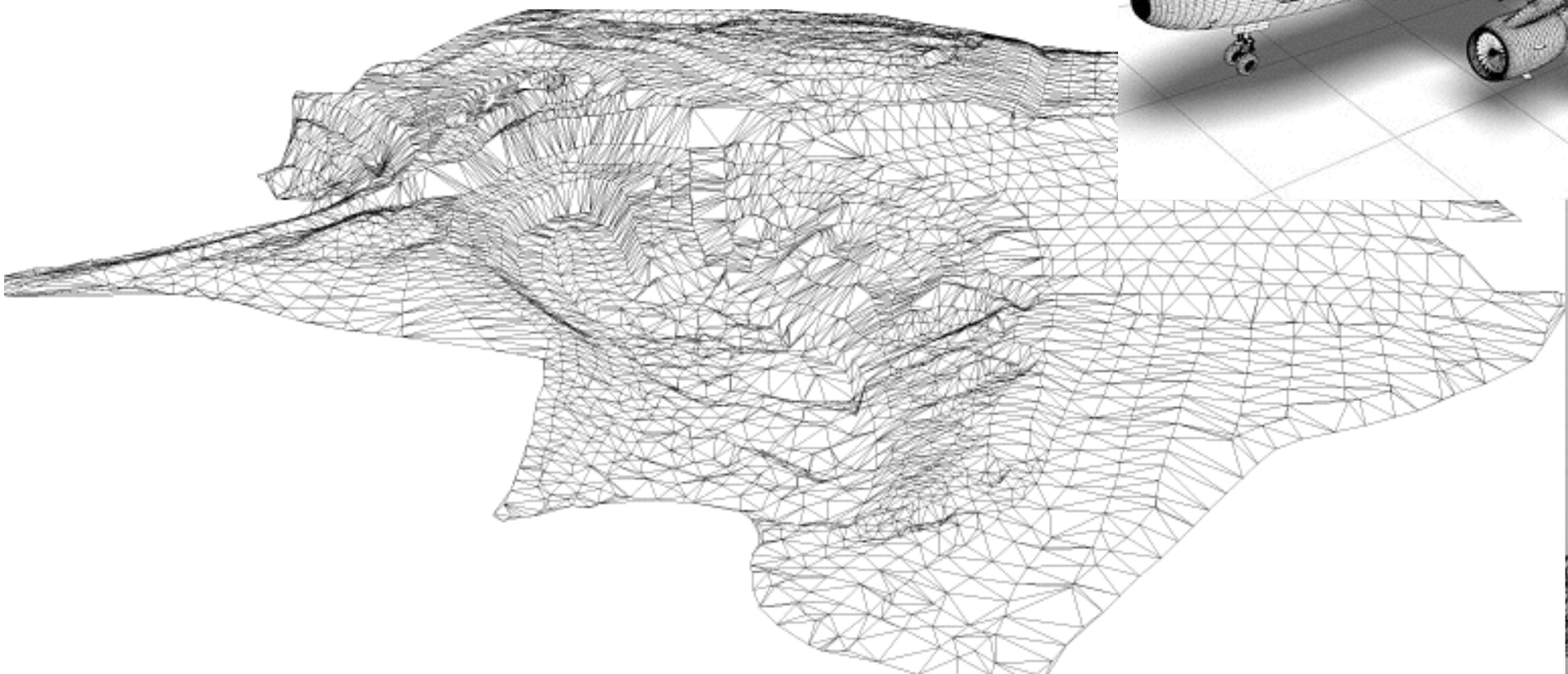
DT Applications

DT Applications



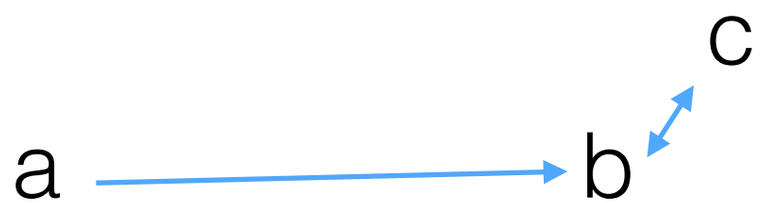
- DT used in high-quality mesh generation because it is angle-optimal.





All nearest neighbors

- Let P = {p_1, p_2,…,p_n} a set of points in the plane.
- Problem: Find the nearest neighbor to each point in this set.



- Note: NN not symmetrical
- The set of edges (p, nn(p)) define the nearest-neighbor graph (NNG)
- **Theorem: NNG(P) is included in DT(P).**
- To find all NNs, search the edges of DT(P) <———-Overall O(n) if DT is given.

Euclidian minimum spanning tree

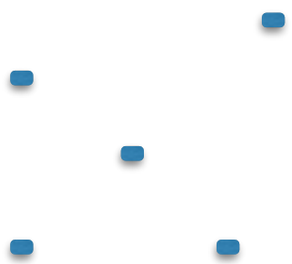
- MST: Given a graph G, find MST
- Euclidian MST: given a set of points in the plane, assume implicit edges between any pairs of points with weight of an edge = Euclidian distance. Find MST on this implicit complete graph.

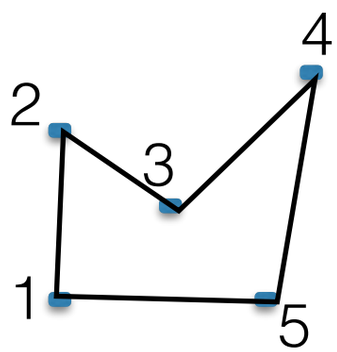
etc

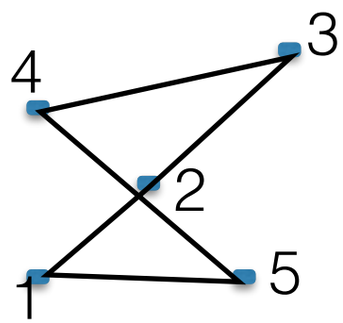
- Kruskal/Prim: $O(n^2 \lg n)$
- **Theorem: Euclidian MST of P is a subset of DT(P).**
- To find the Euclidian MST, run Kruskal/Prim on DT(P) <——- $O(n \lg n)$

Euclidian Traveling salesman problem (TSP)

- Given a set P of points in the plane, find the shortest closed path that visits every point.







etc

- NP-hard
- Theorem: A 2-approximation algorithm via MST based on DT(P)
- Idea:
  - find MST(P), and walk along it starting at an arbitrary point. Its length is 2MST.
  - Let T = optimal TSP tour
  - Let T' = T - one edge removed. T' must be an MST! thus T' > MST
  - MST < T' < T
  - 2MST < 2T

More?

- DT maximizes minimum angle
- DT does not minimize maximum angle
- DT does not minimize the longest edge
- DT does not minimize the total length of all triangle edges