Computational Geometry
csci3250

Laura Toma

Bowdoin College

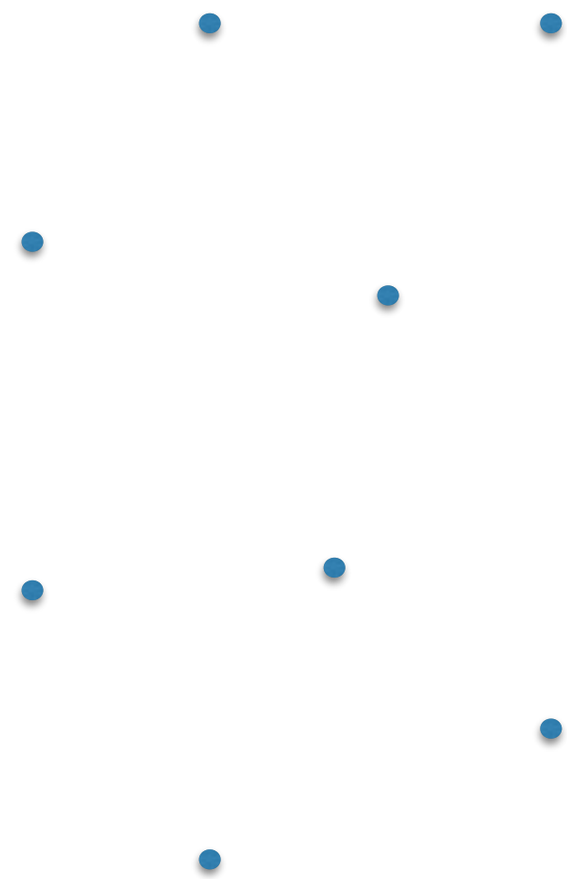
Outline

- Triangulating a set of points
- The Delaunay triangulation
  - Definition and Properties
  - DT and angles
  - Construction via edge flipping
  - RIC

Triangulating a set of points

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

Want to triangulate P.

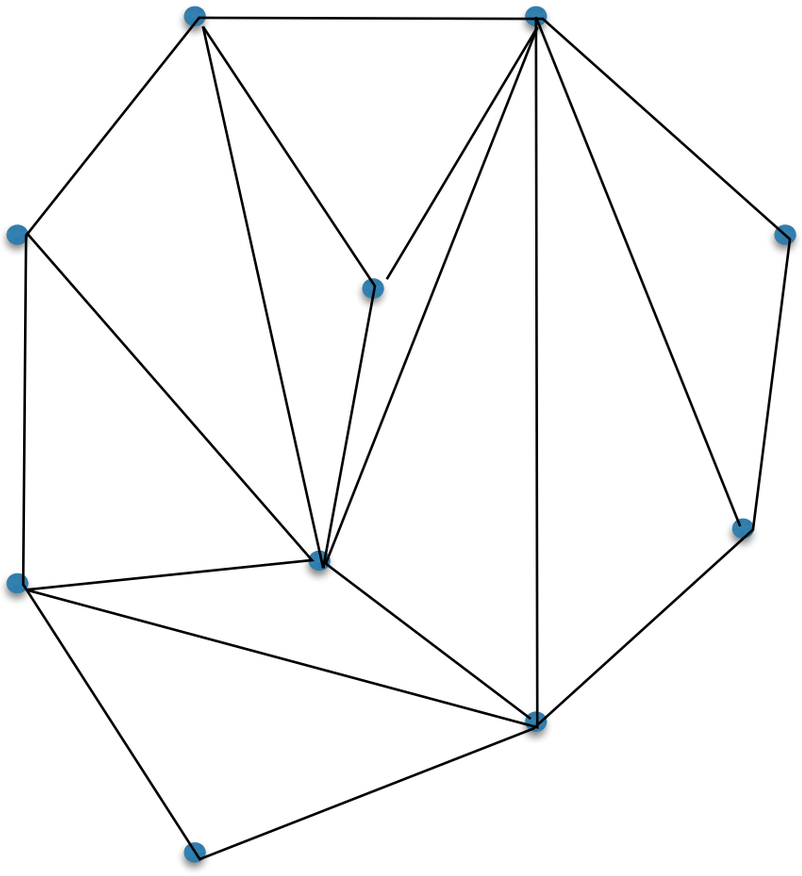


A triangulation T(P) is a planar subdivision whose vertices are P and all faces are triangles (except the unbounded face, which is bounded by the hull)

Triangulating a set of points

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

Want to triangulate P.

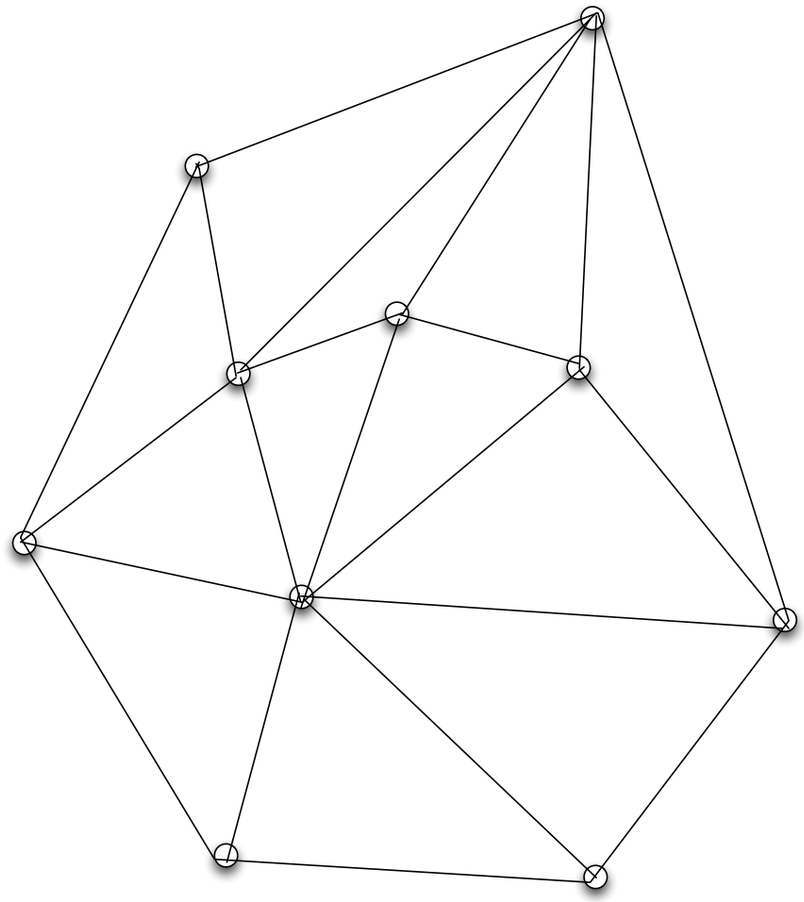


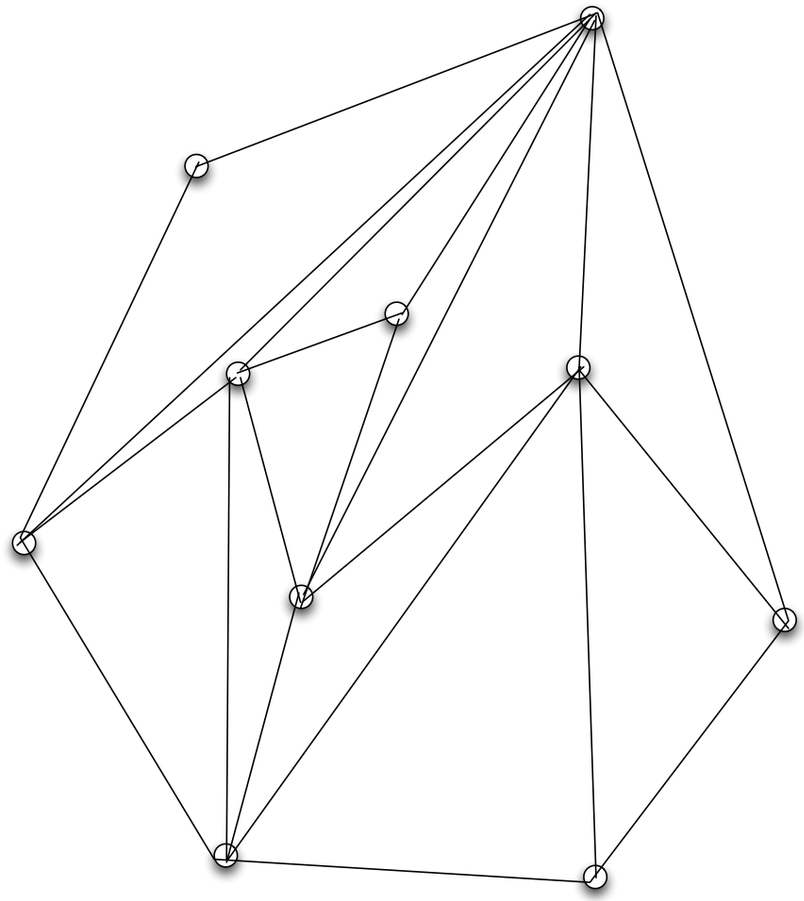
A possible triangulation

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

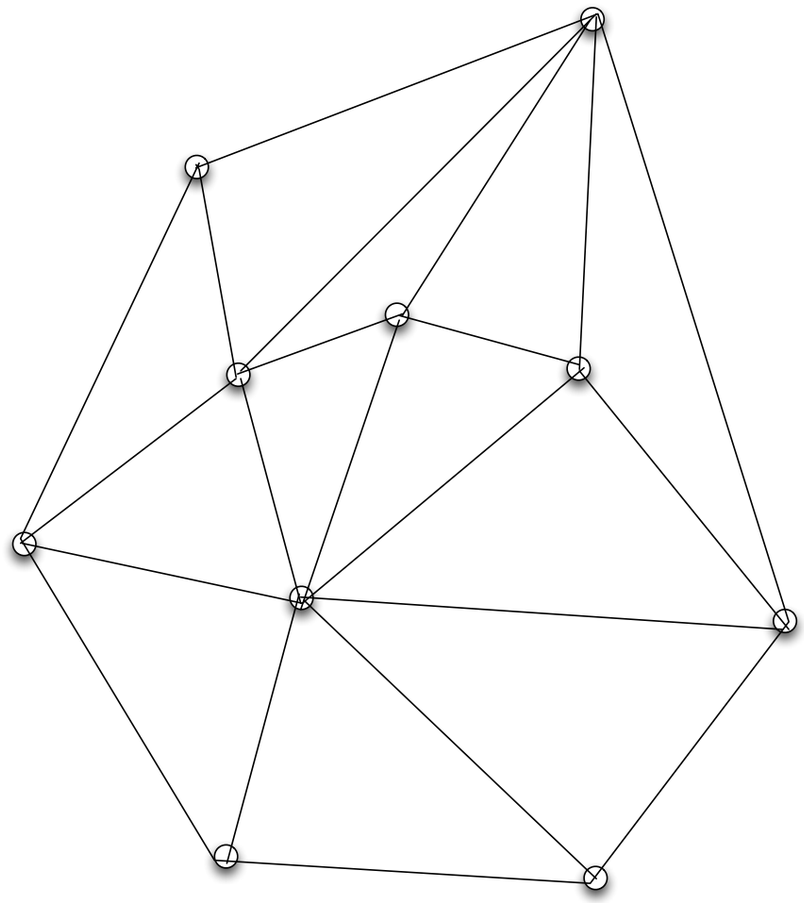
- A point set has many possible triangulations

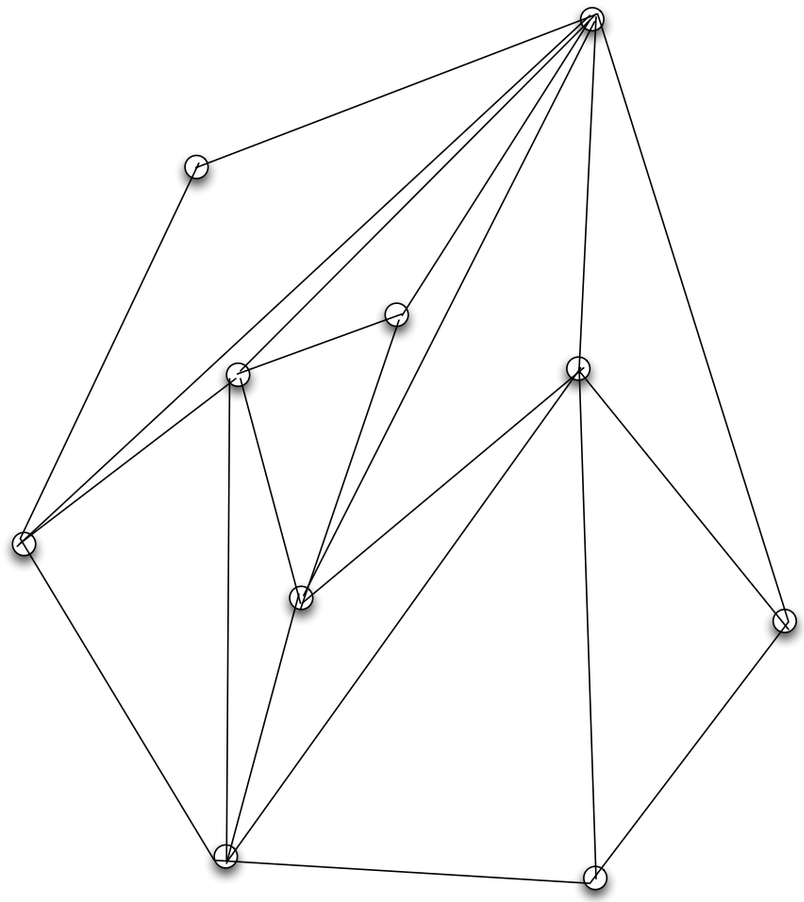




- In practice, want triangulations without small angles
  - small angles cause numerical instability with geometric predicates

Triangulating a set of points





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

Preview: The Delaunay Triangulation

- 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 and such; this causes big issues in modeling

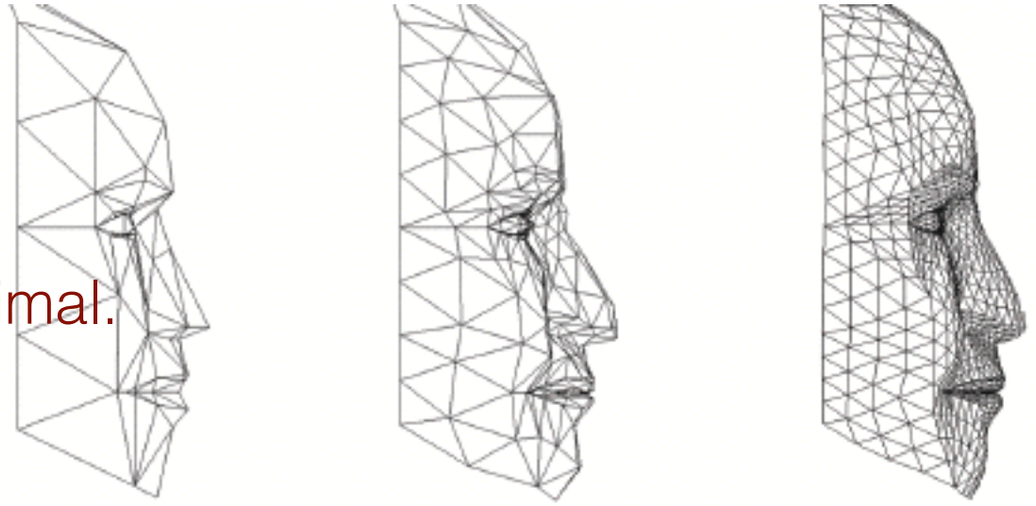
- Of all possible triangulations of a point set P, the triangulation that maximizes the minimum angle is the Delaunay triangulation

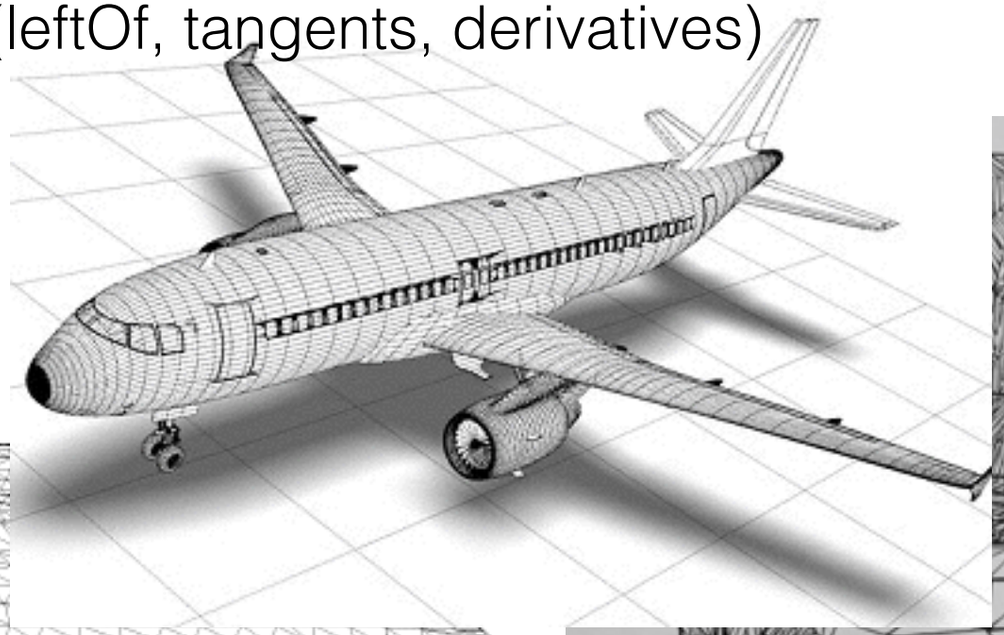
- Delaunay triangulation is the default triangulation used in practice, and has many applications and elegant properties

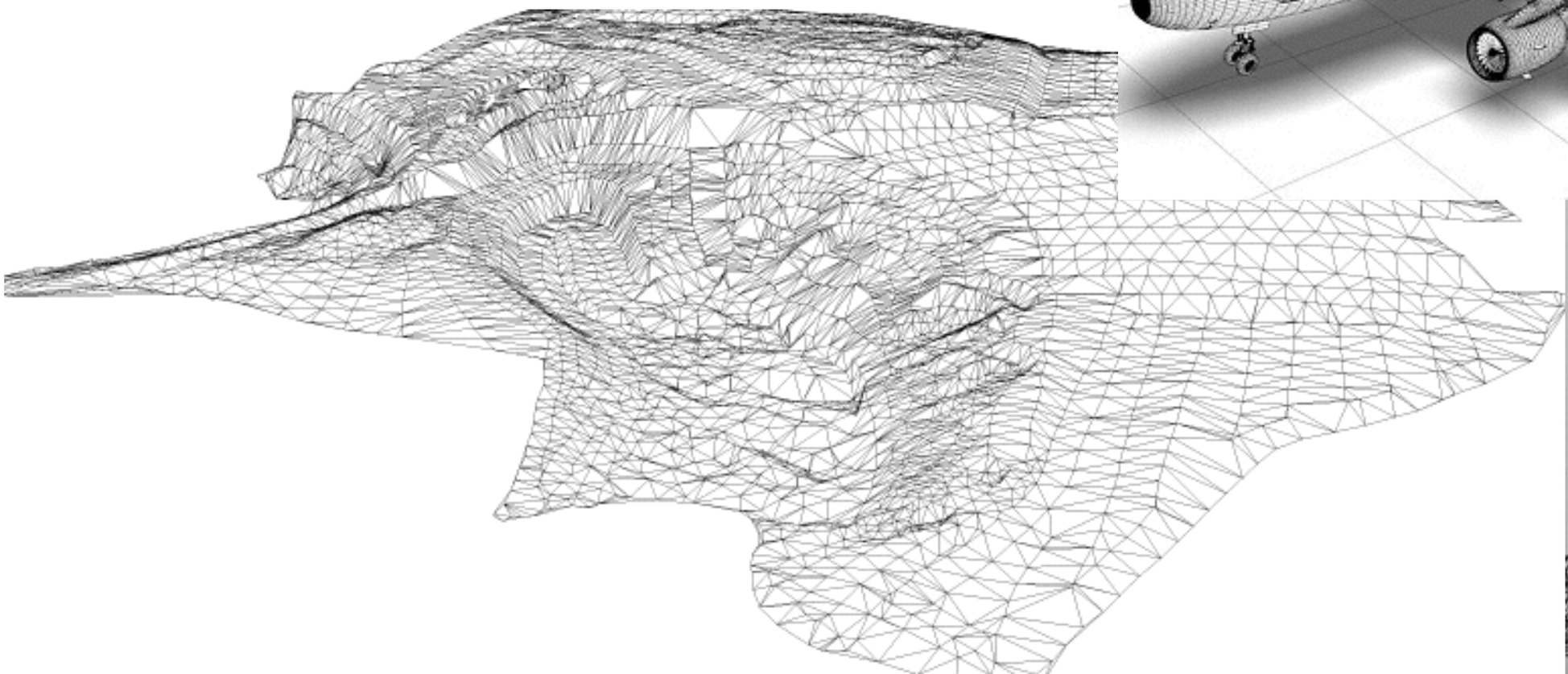
- It is defined via Voronoi diagram

DT Applications

- DT used in practice because it is angle-optimal.
  - Small angles
    - Numerical instability, Interpolation errors
    - Issues with geometric predicates (leftOf, tangents, derivatives)



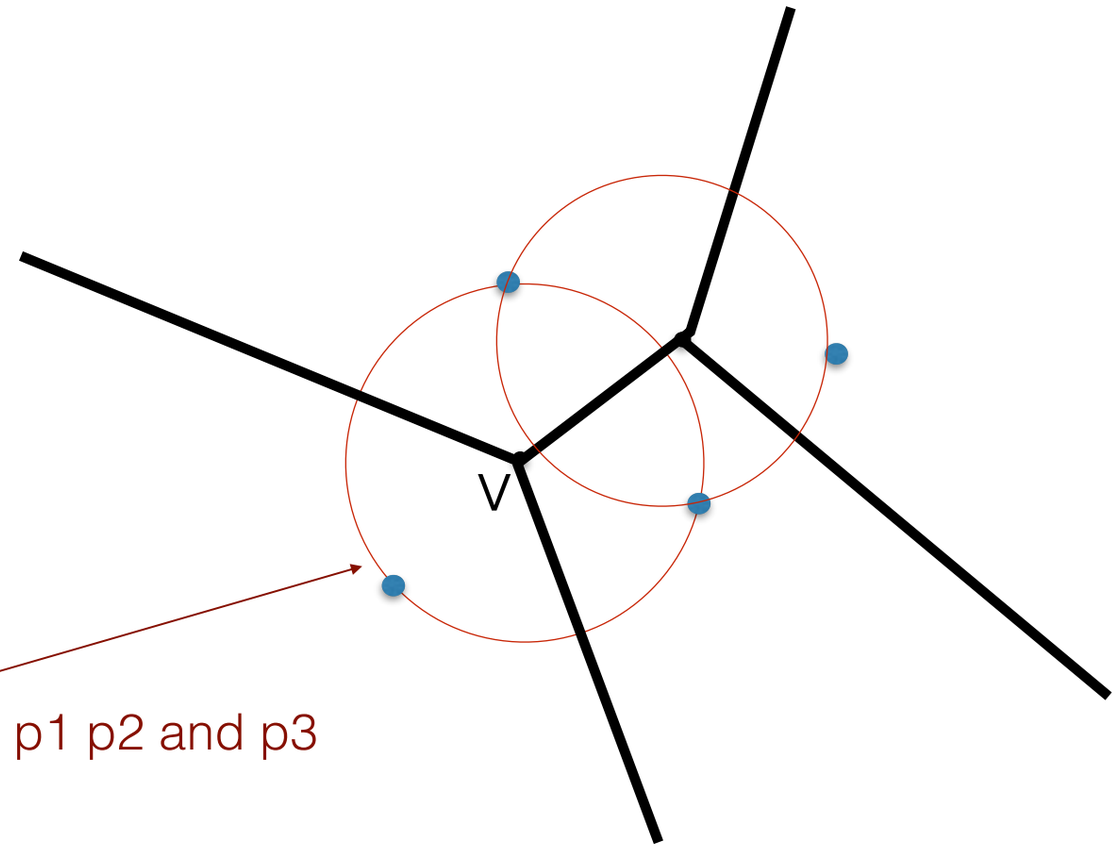




Voronoi Diagram

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 sites inside



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

The Dual

- The dual of a planar graph G is defined as follows
  - it has a vertex for each face of G
  - it has an edge between any pair of faces that are adjacent

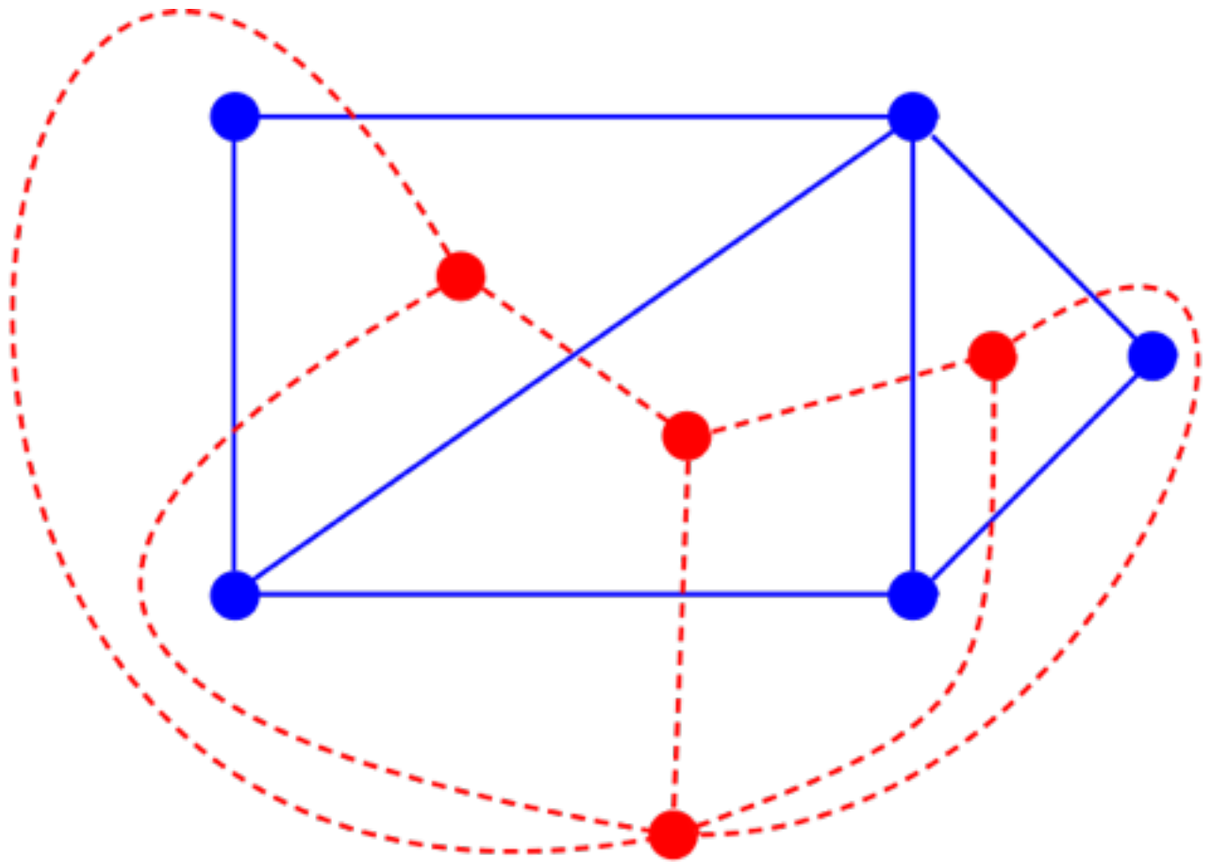


image thanks to wikipedia

The Dual

- The dual of a planar graph G is defined as follows
  - it has a vertex for each face of G
  - it has an edge between any pair of faces that are adjacent

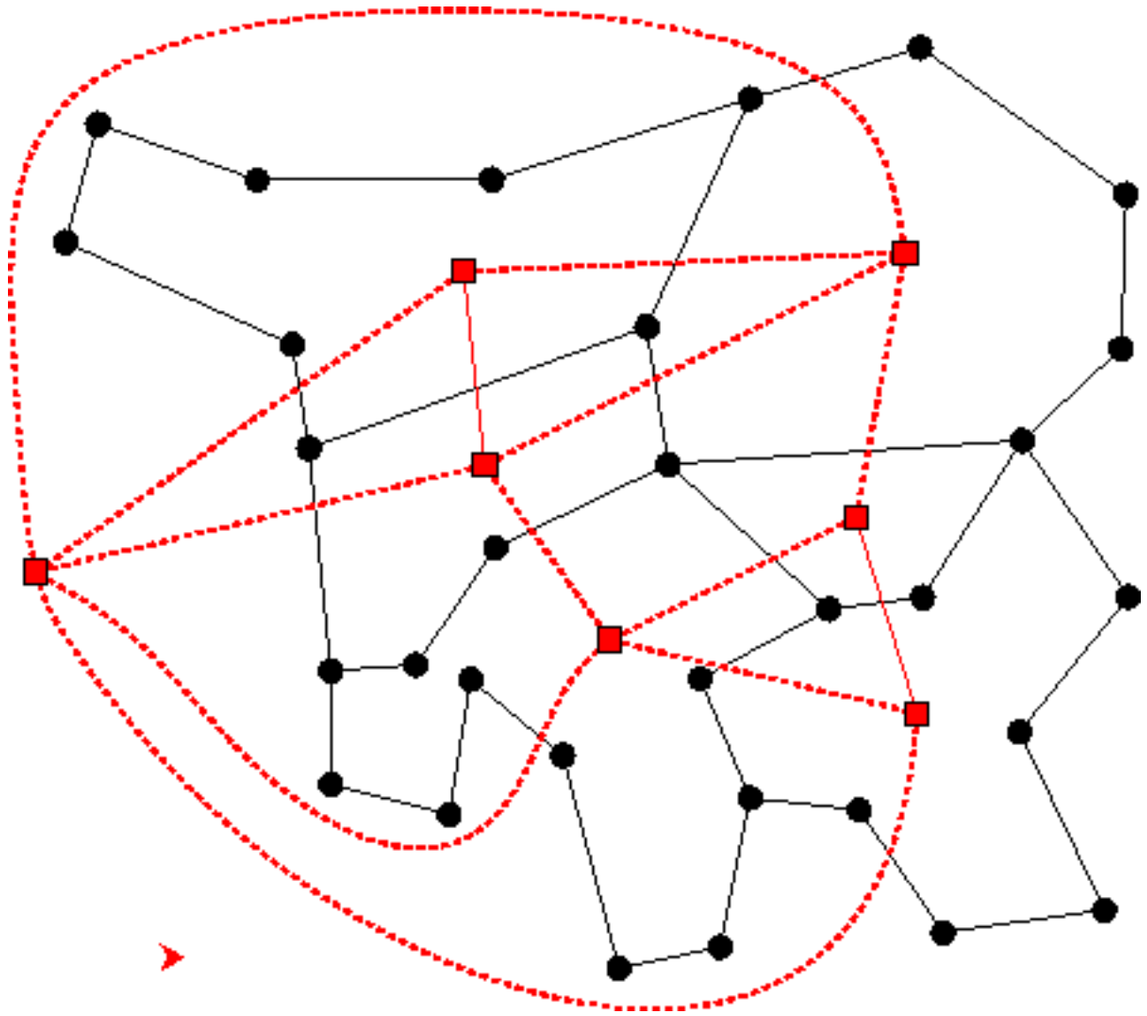


image thanks to LEDA

What is LEDA?

LEDA is the C++ **Library of Efficient Data Types and Algorithms**, which was developed at the Max-Planck-Institut für Informatik since the end of the eighties.

The Dual of Vor(P)

- Let P = {p_1, p_2,…,p_n} set of points in the plane such that no 4 are co-circular.
- Vor(P) is a planar graph
- The 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

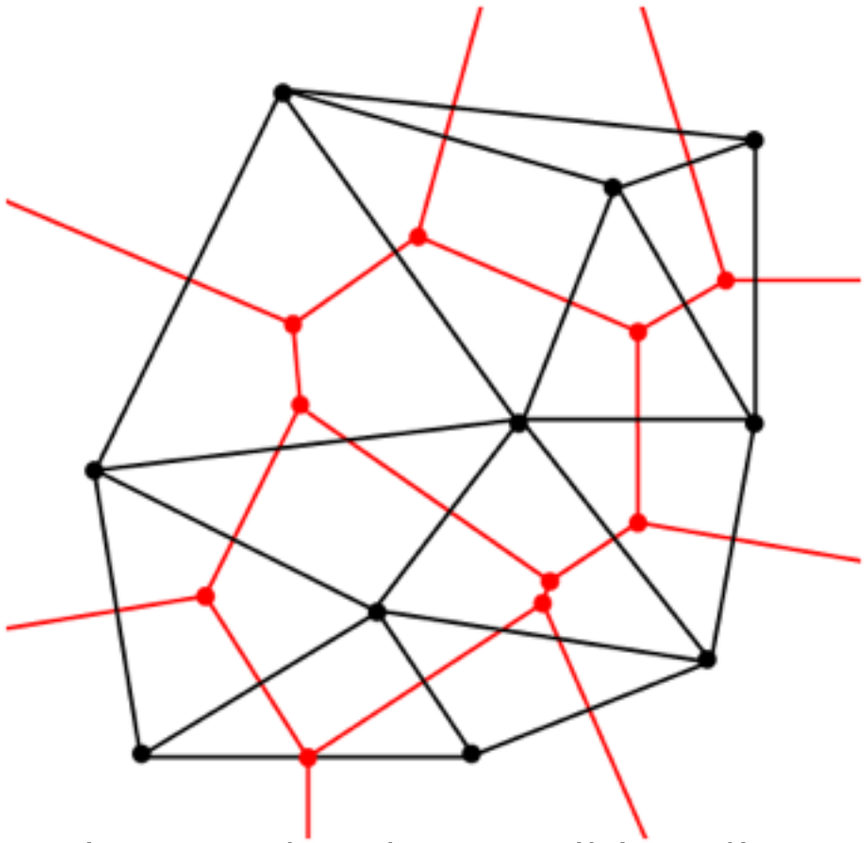
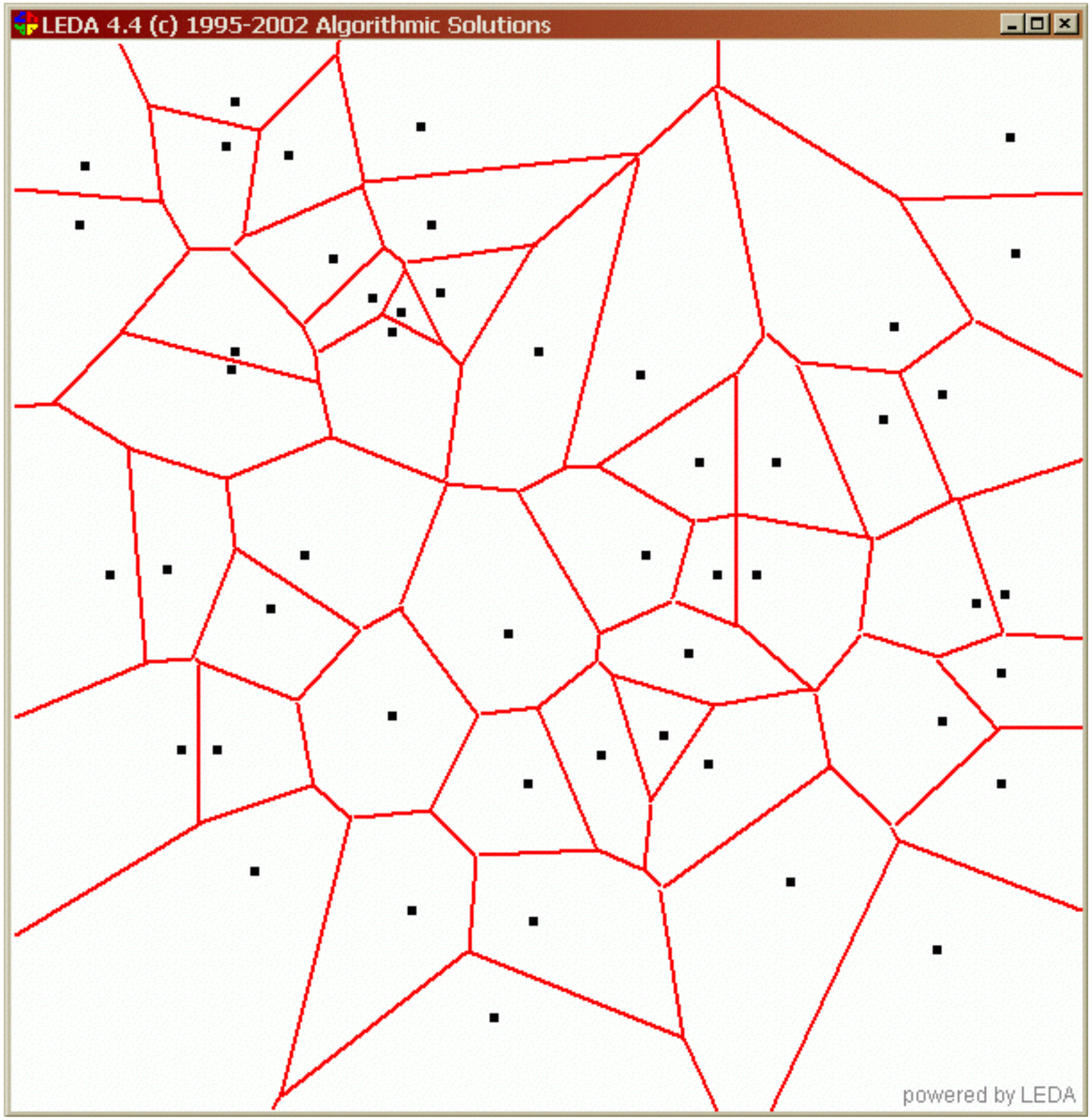
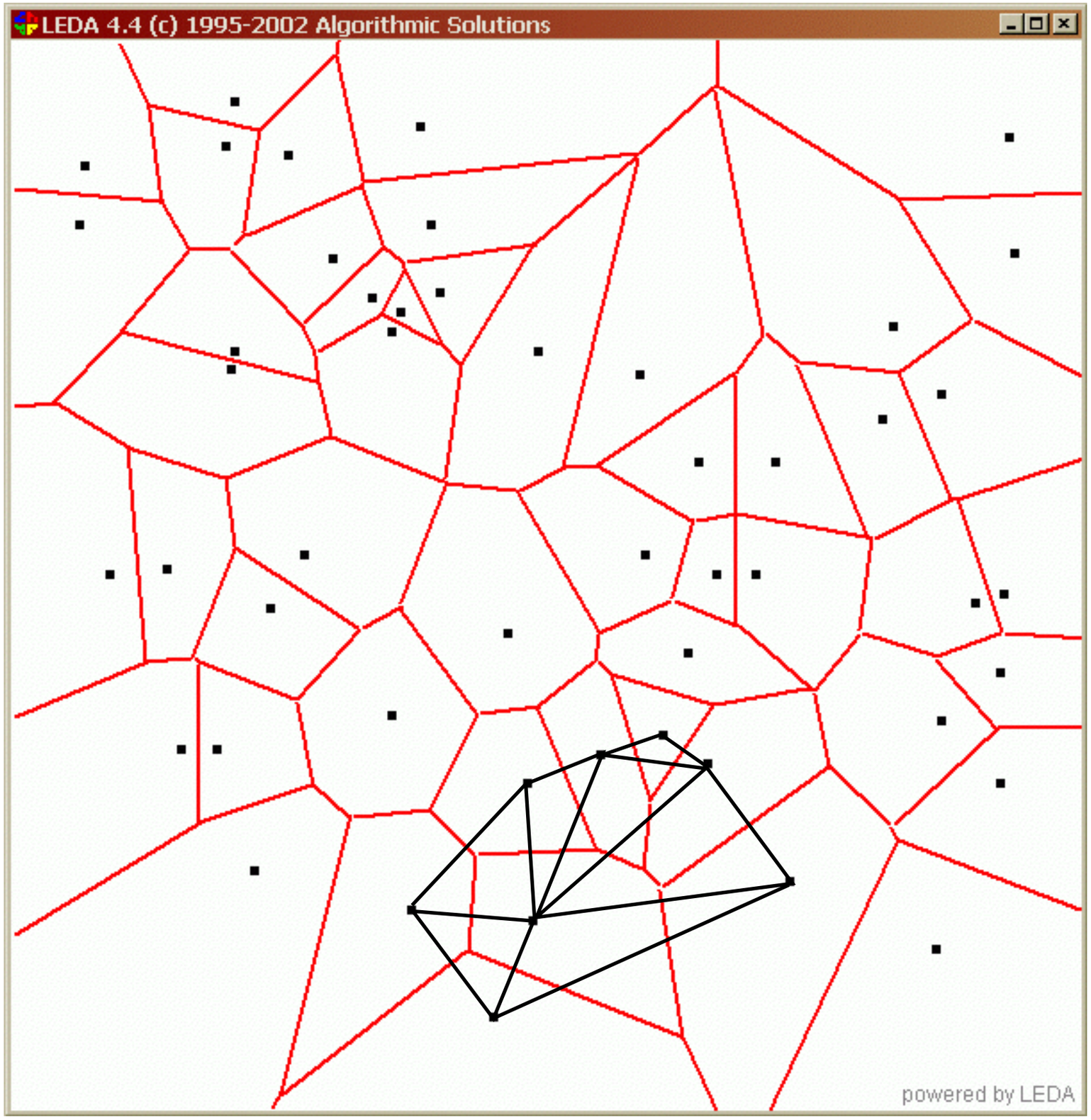


image thanks to wikipedia


LEDA 4.4 (c) 1995-2002 Algorithmic Solutions
powered by LEDA


LEDA 4.4 (c) 1995-2002 Algorithmic Solutions
powered by LEDA

The Dual of Vor(P)

- Claim: In the dual of Vor(P) each Voronoi vertex v defines a triangle.

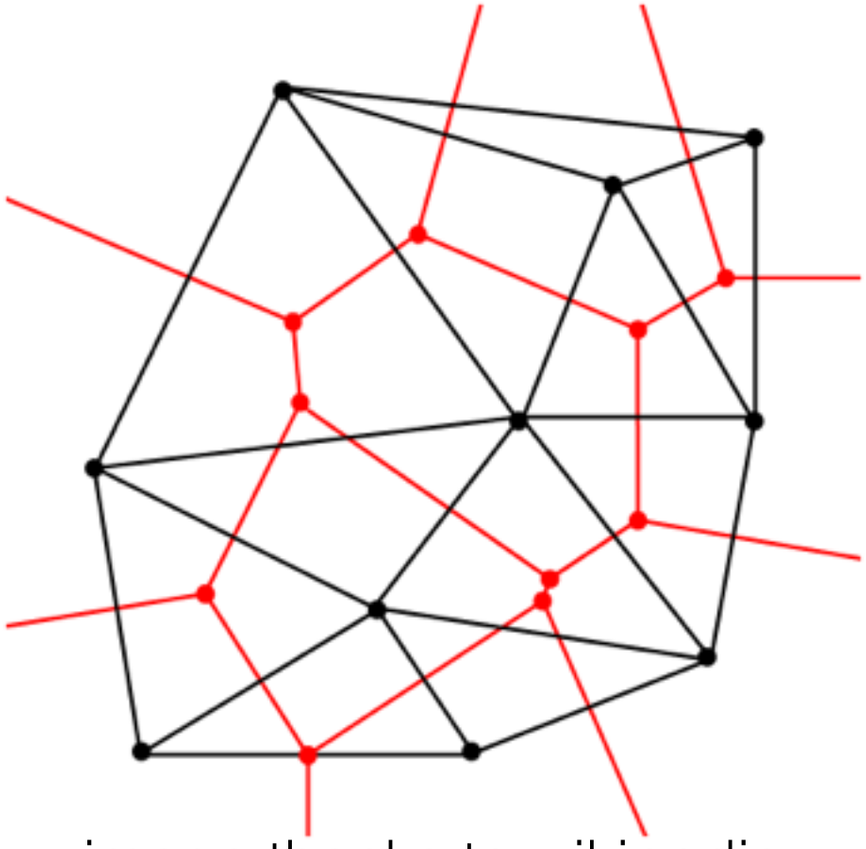
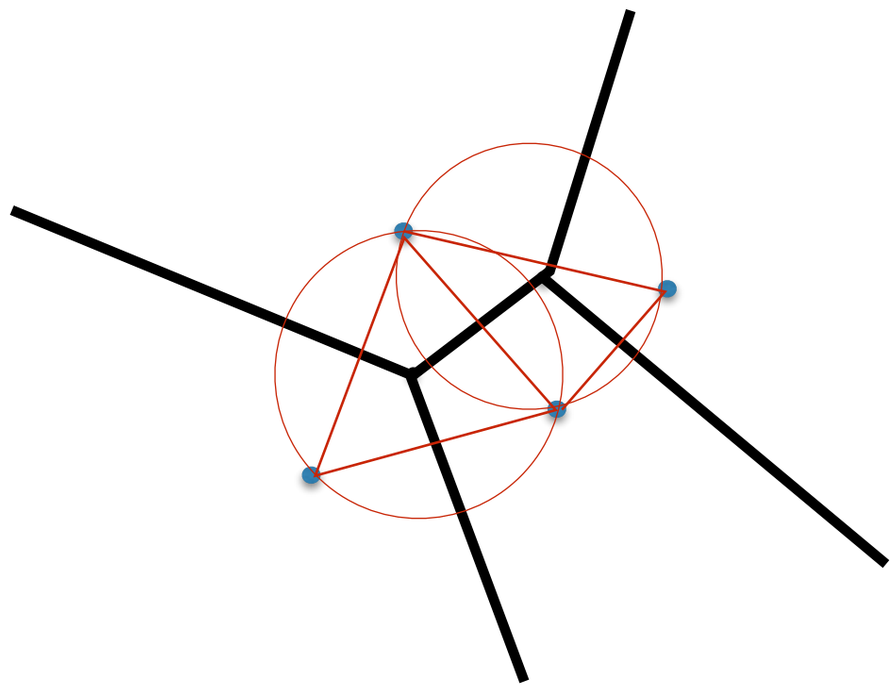


image thanks to wikipedia



Theorem [Delaunay 1934]:

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

Proof: not trivial. See p.197 in 4M book.

Delaunay Triangulation Construction

Delaunay Triangulation Construction

- Construct Vor(P) and get the dual
  - Use for e.g. Fortune's algorithm which runs in O(n lg n)
  - Size of Vor(P) is O(n)
  - The dual can be obtained in O(n) time

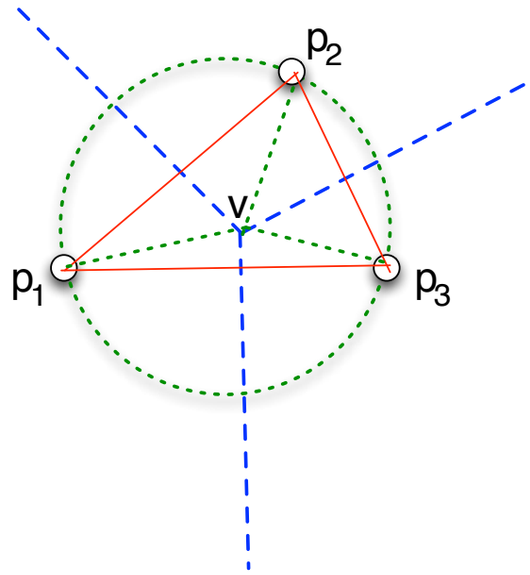
Delaunay Triangulation Construction

- Construct Vor(P) and get the dual
  - Use for e.g. Fortune's algorithm which runs in O(n lg n)
  - Size of Vor(P) is O(n)
  - The dual can be obtained in O(n) time

- We would like to build a Delaunay triangulation directly

Delaunay Triangulation

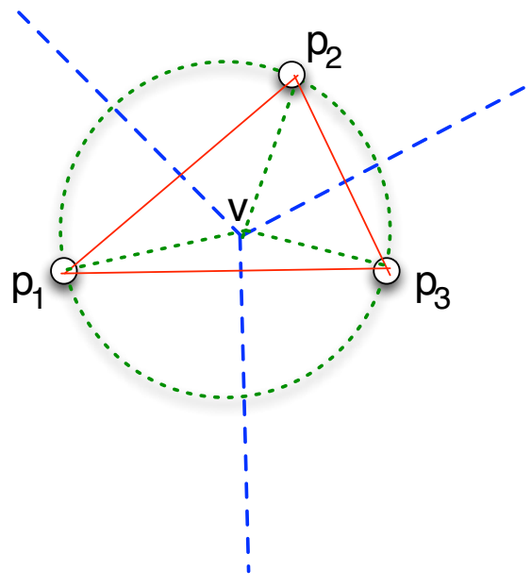
P = {p_1, p_2,…,p_n} set of points in the plane
no 4 points co-circular



Delaunay Triangulation

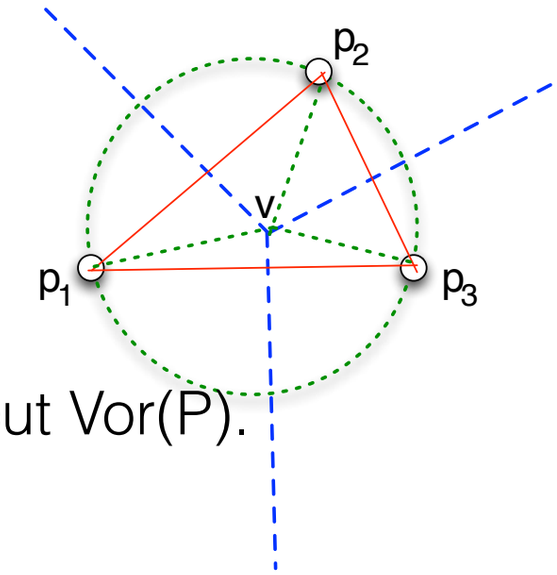
- By definition, DT(P) is the dual of Vor(P).

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



Delaunay Triangulation

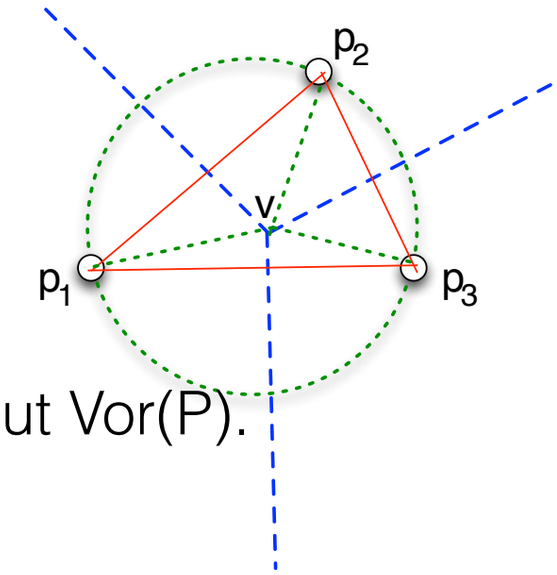
P = {p_1, p_2,…,p_n} set of points in the plane
no 4 points co-circular



- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

Delaunay Triangulation

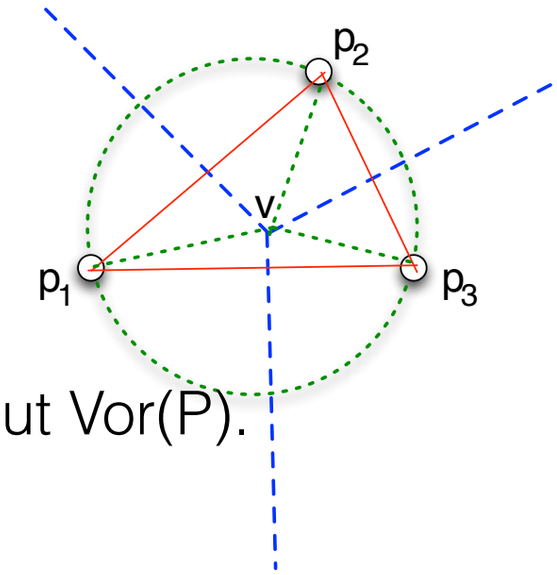
P = {p_1, p_2,…,p_n} set of points in the plane
no 4 points co-circular



- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

Delaunay Triangulation

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

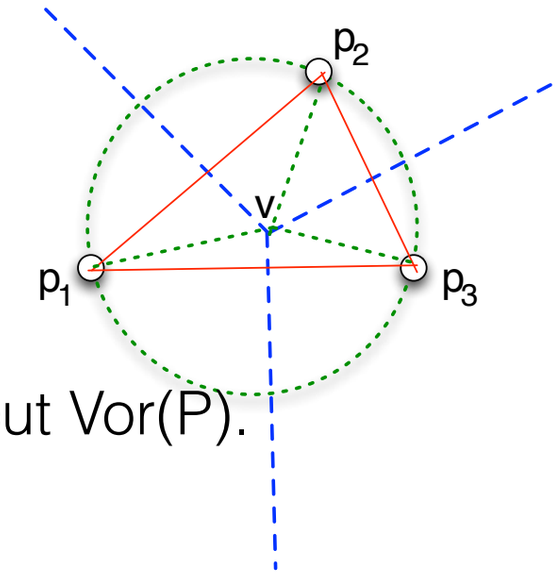


- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

- We know that vertices of Vor(P) have the **empty circumcircle property**

Delaunay Triangulation

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

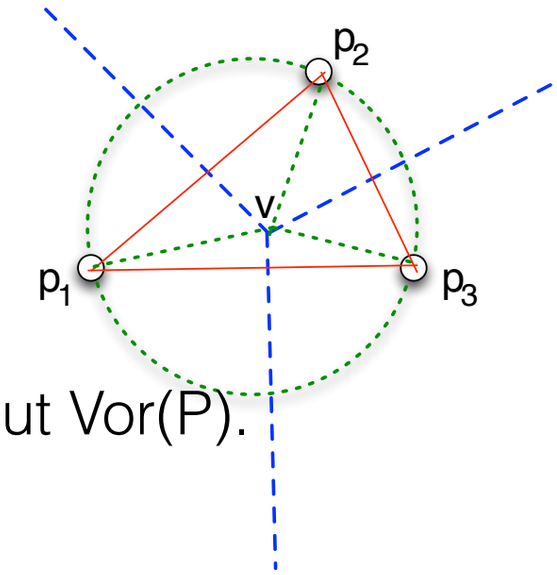


- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

- We know that vertices of Vor(P) have the **empty circumcircle property**
  - This means that if $p_1p_2p_3$ is a triangle in DT(P), the circumcircle of $p_1p_2p_3$ is empty

Delaunay Triangulation

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

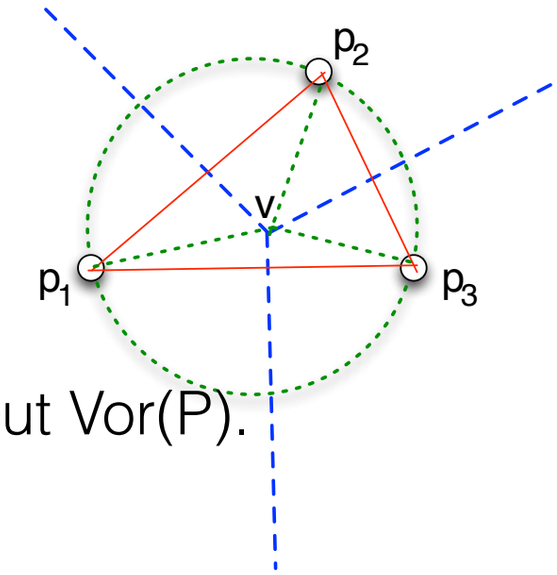


- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

- We know that vertices of Vor(P) have the **empty circumcircle property**
  - This means that if $p_1p_2p_3$ is a triangle in DT(P), the circumcircle of $p_1p_2p_3$ is empty

Delaunay Triangulation

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



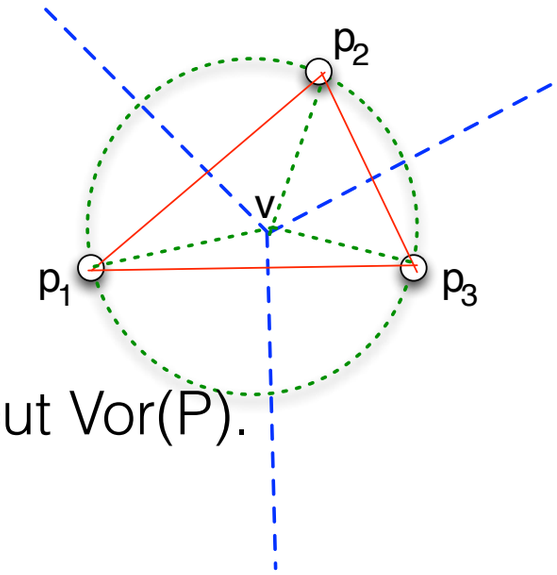
- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

- We know that vertices of Vor(P) have the **empty circumcircle property**
  - This means that if $p_1p_2p_3$ is a triangle in DT(P), the circumcircle of $p_1p_2p_3$ is empty

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

Delaunay Triangulation

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



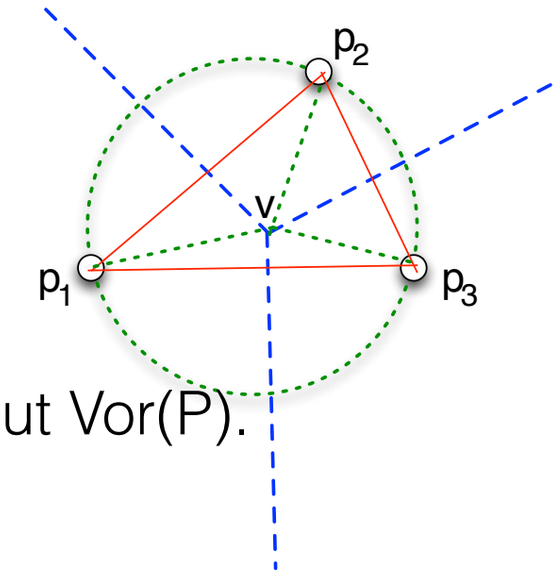
- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

- We know that vertices of Vor(P) have the **empty circumcircle property**
  - This means that if $p_1p_2p_3$ is a triangle in DT(P), the circumcircle of $p_1p_2p_3$ is empty

- It was shown that this is true the other way around
  - If all triangles in a triangulation have their circumcircles empty, then the triangulation is the DT

Delaunay Triangulation

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



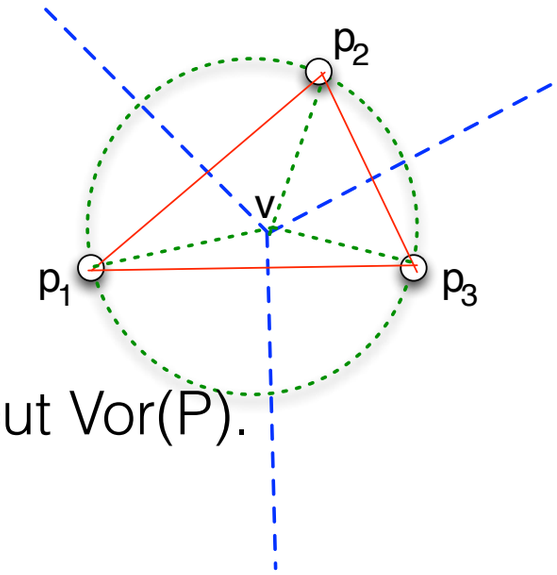
- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

- We know that vertices of Vor(P) have the **empty circumcircle property**
  - This means that if $p_1p_2p_3$ is a triangle in DT(P), the circumcircle of $p_1p_2p_3$ is empty

- It was shown that this is true the other way around
  - If all triangles in a triangulation have their circumcircles empty, then the triangulation is the DT

Delaunay Triangulation

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



- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

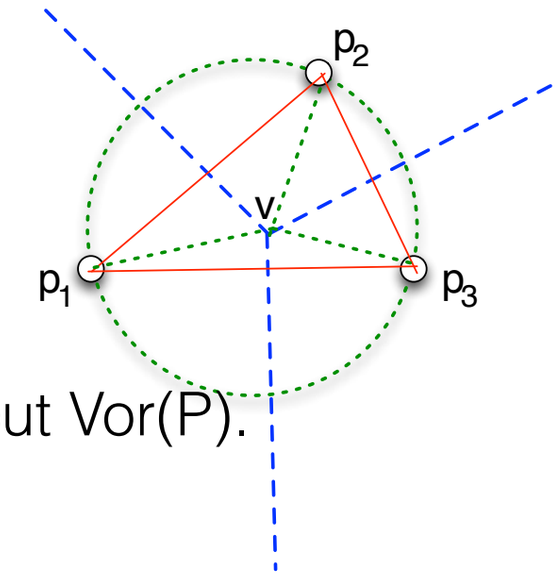
- We know that vertices of Vor(P) have the **empty circumcircle property**
  - This means that if $p_1p_2p_3$ is a triangle in DT(P), the circumcircle of $p_1p_2p_3$ is empty

- It was shown that this is true the other way around
  - If all triangles in a triangulation have their circumcircles empty, then the triangulation is the DT

Theorem: A triangulation of P is the Delaunay triangulation if and only if all triangles of T(P) have the empty-circle property.

Delaunay Triangulation

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



- By definition, DT(P) is the dual of Vor(P).
- Would be nice to have a direct way to characterize DT(P) without Vor(P).

- We know that vertices of Vor(P) have the **empty circumcircle property**
  - This means that if $p_1p_2p_3$ is a triangle in DT(P), the circumcircle of $p_1p_2p_3$ is empty

- It was shown that this is true the other way around
  - If all triangles in a triangulation have their circumcircles empty, then the triangulation is the DT

Theorem: A triangulation of P is the Delaunay triangulation if and only if all triangles of T(P) have the empty-circle property.

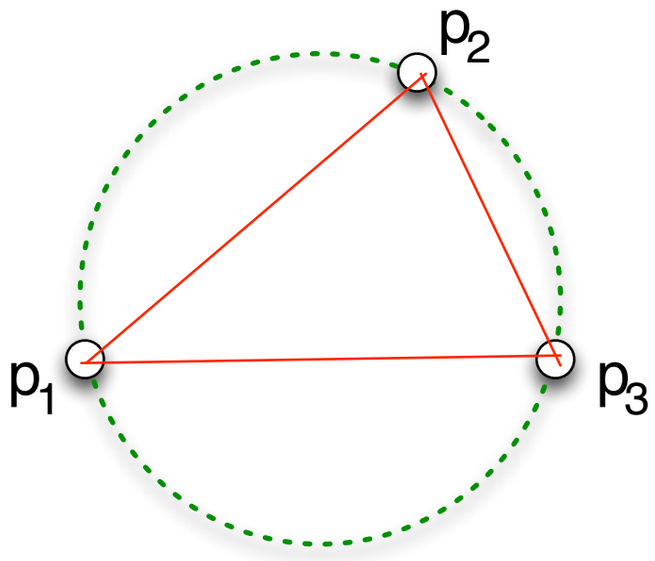
Delaunay Triangulation

This gives us a way to build it directly

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

Constructing the Delaunay Triangulation (DT)

How to build a triangulation such that **all triangles have the empty-circumcircle property**?



circumcircle of $p_1p_2p_3$ is empty (of other sites)

Idea: Start with an arbitrary triangulation, and flip edges until all triangles have the empty-circle property.

Constructing DT

Algorithm idea:

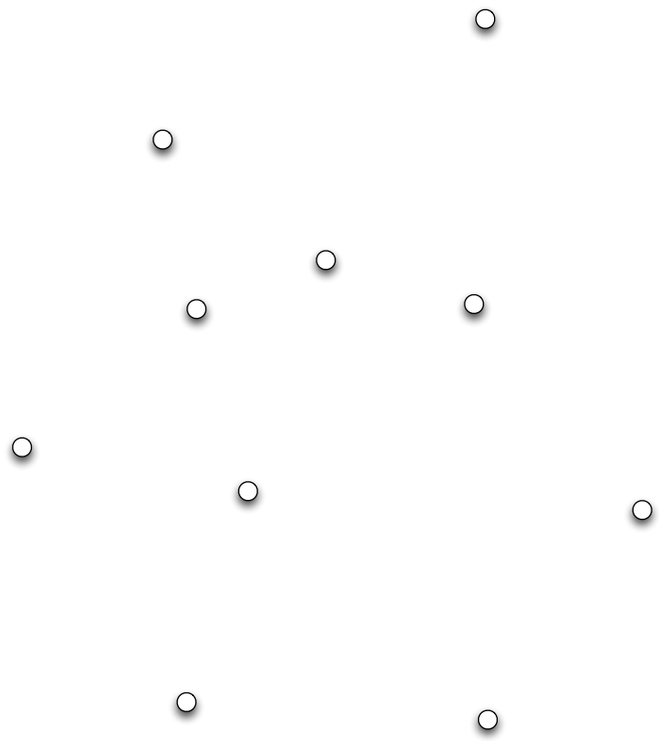
1. Compute an arbitrary triangulation T(P)
2. Flip edges in T until all triangles have the empty circumcircle property

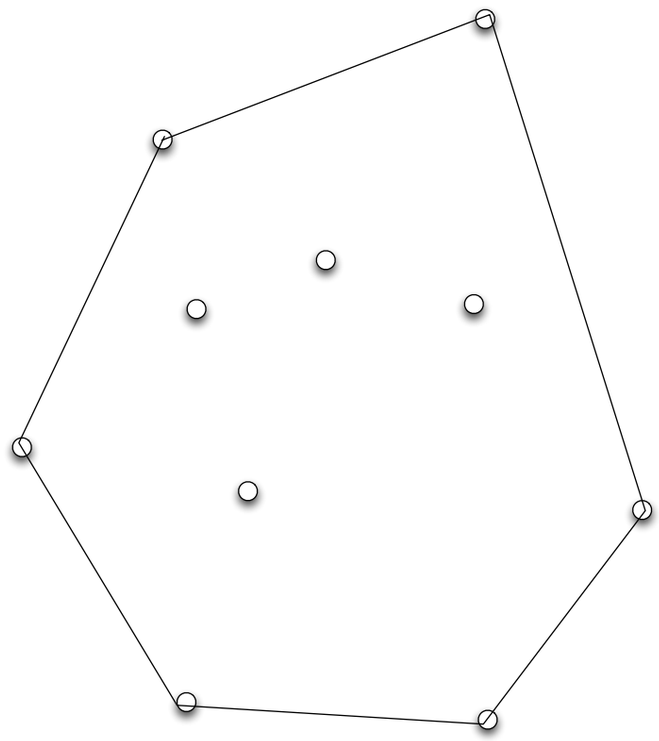
By theorem, the resulting triangulation is the Delaunay triangulation.

Constructing DT

- First, how to build *any* triangulation?
  - Input: a set of n points, no 4 co-circular
  - Output: a partition of CH(P) into triangles

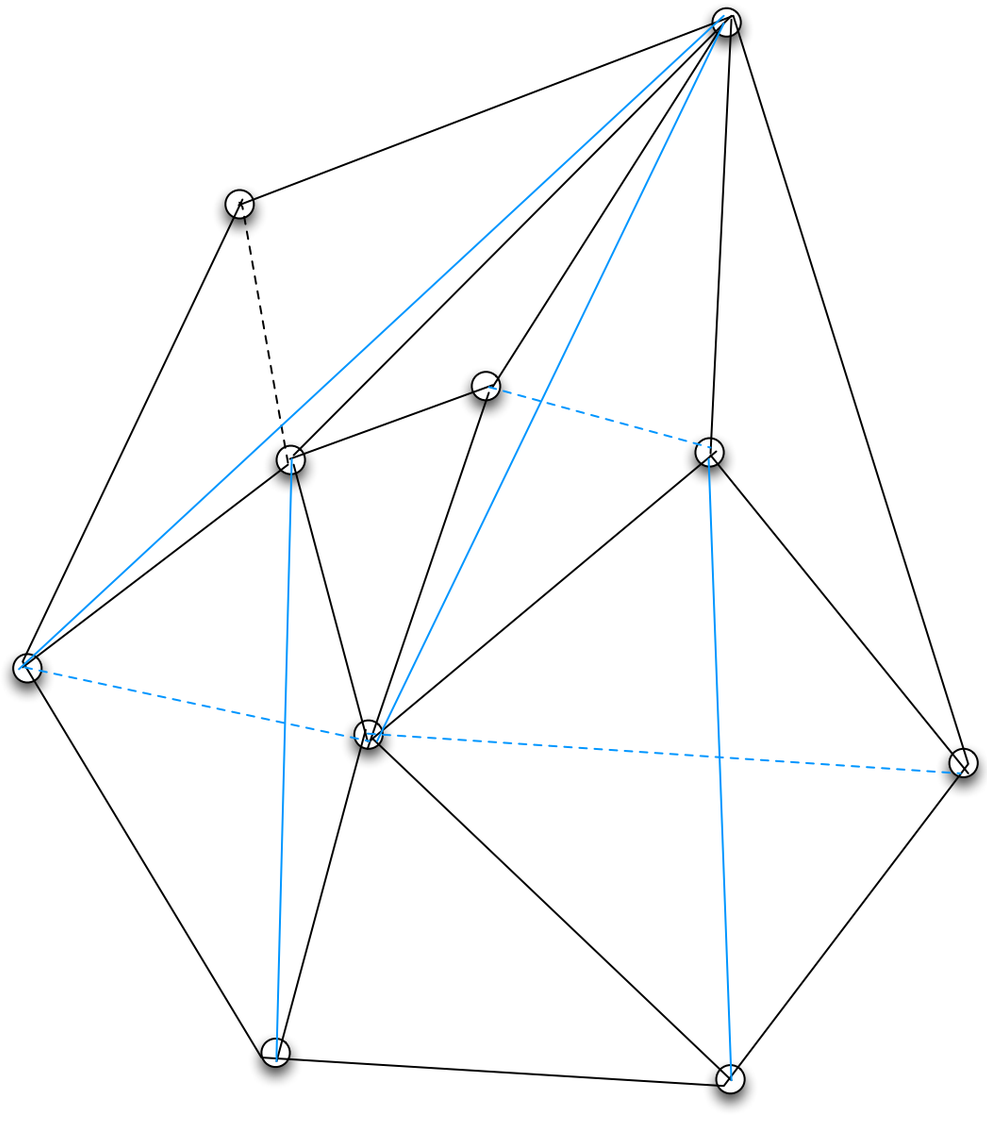
- Come up with an algorithm to compute a triangulation of P
  - Possible ideas: incremental, plane sweep, divide-and-conquer, …





Flipping edges

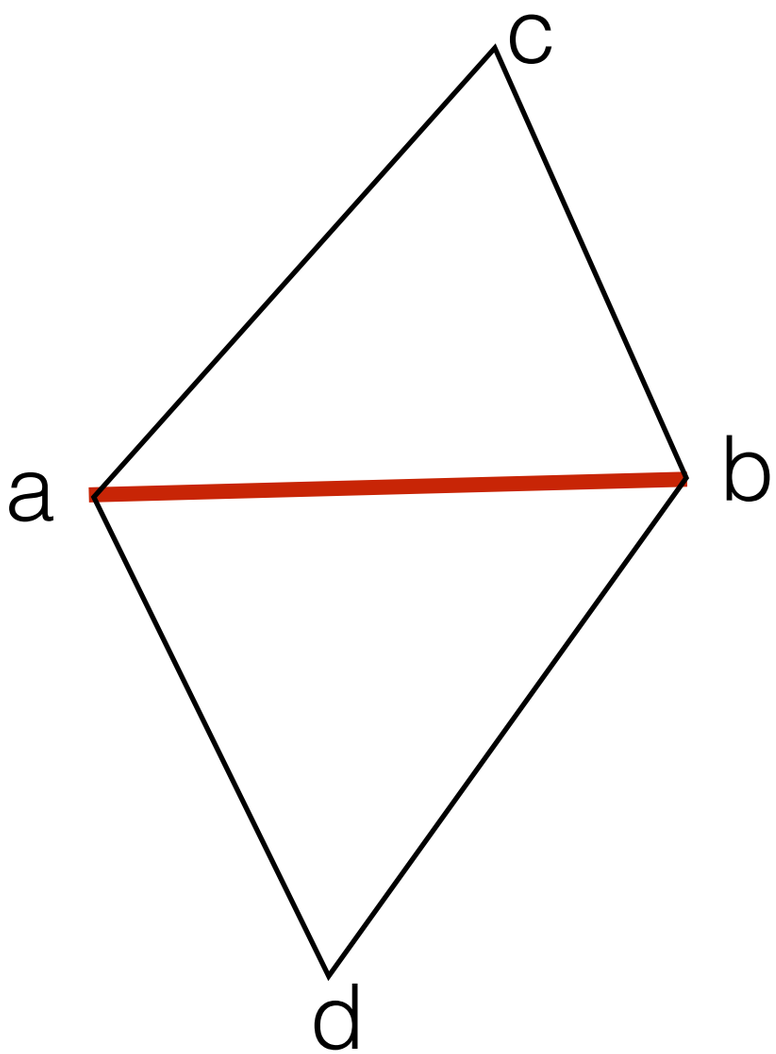
Given a triangulation of P, we can get a different triangulation by flipping an edge.



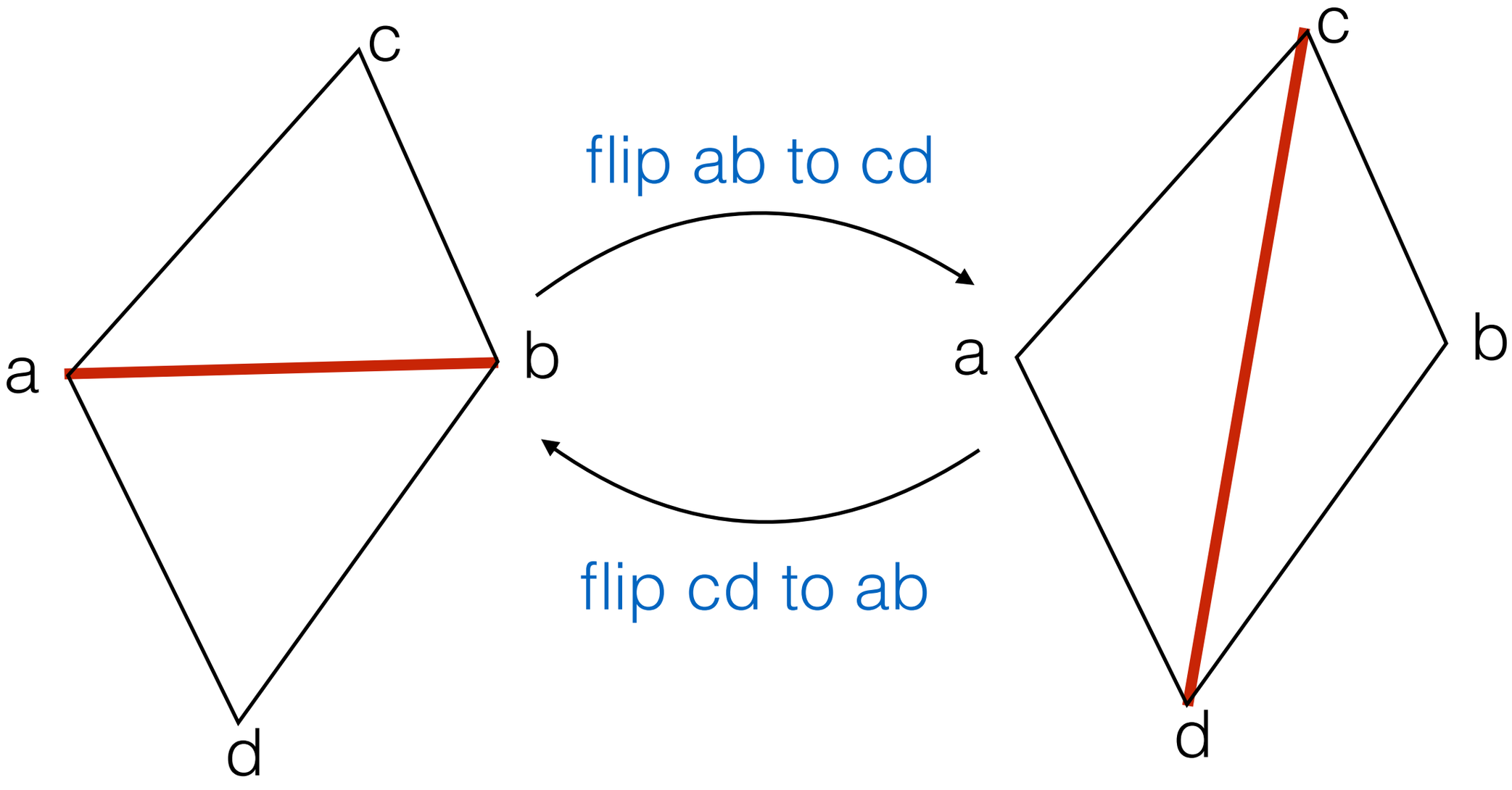
We'll target triangles whose "empty circumcircle property" is violated.

Ever edge has two adjacent triangles.

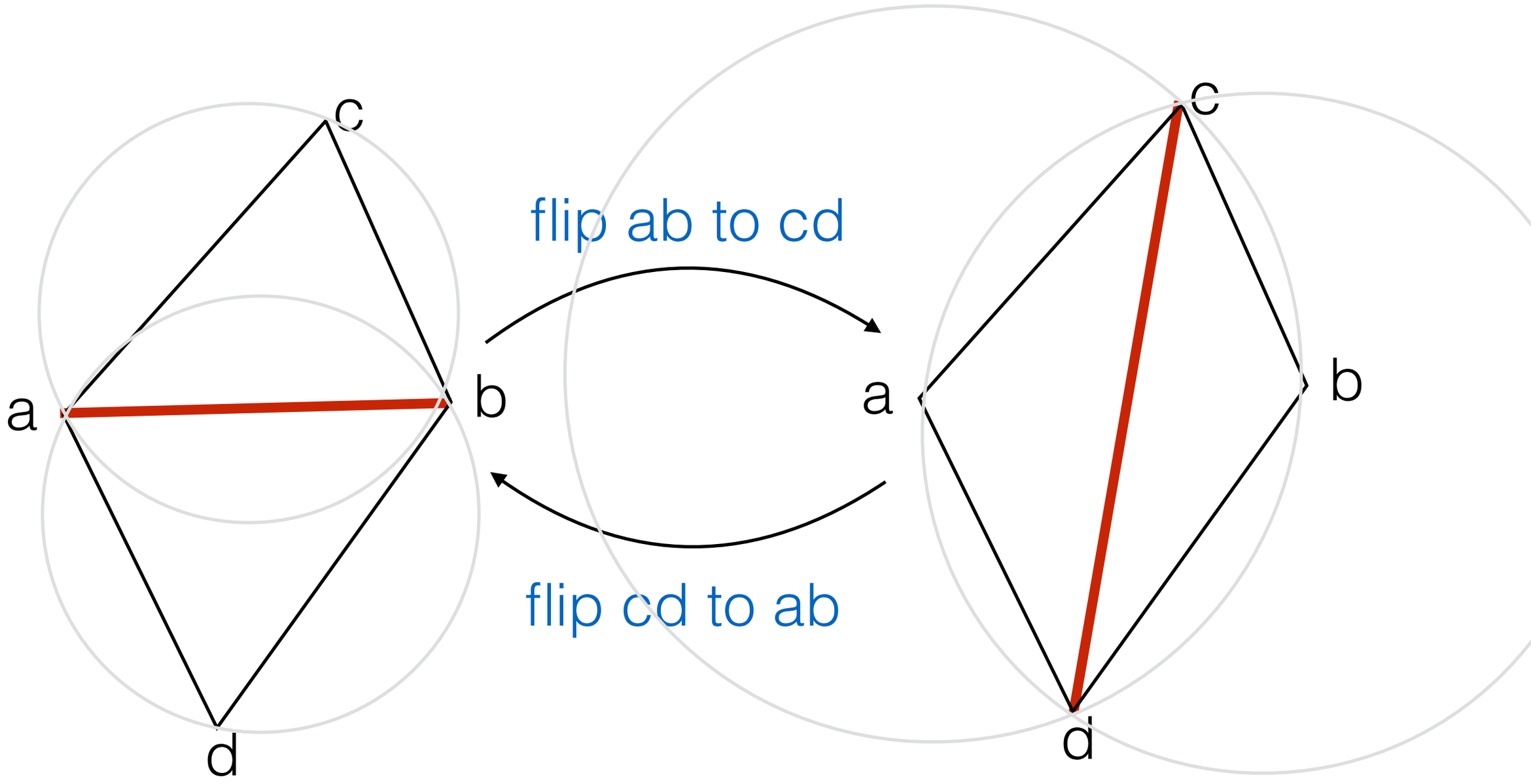
Ever edge has two adjacent triangles.



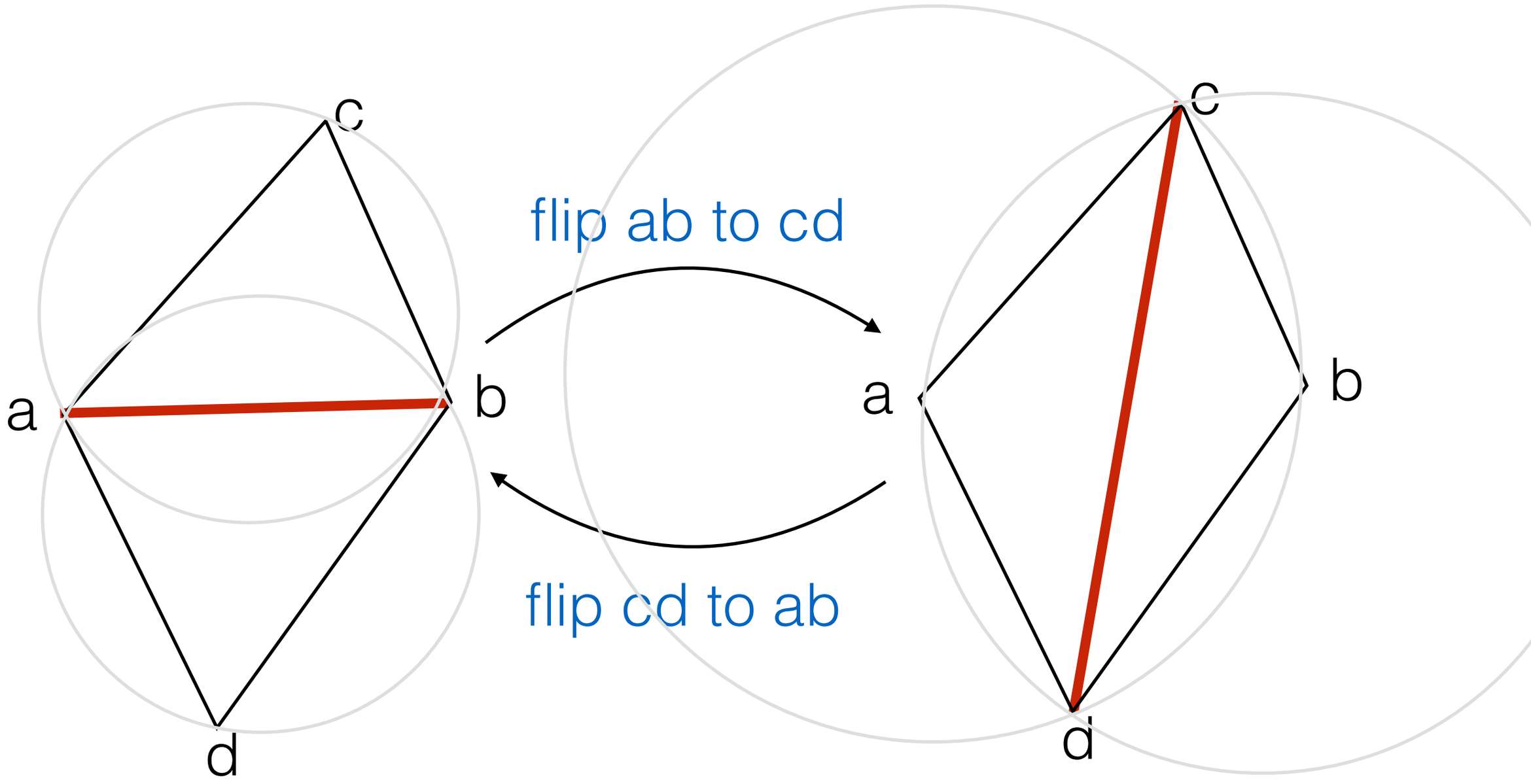
Flipping an edge



Our goal is to obtain triangles with empty-circumcircles.



Looking at circumcircles:

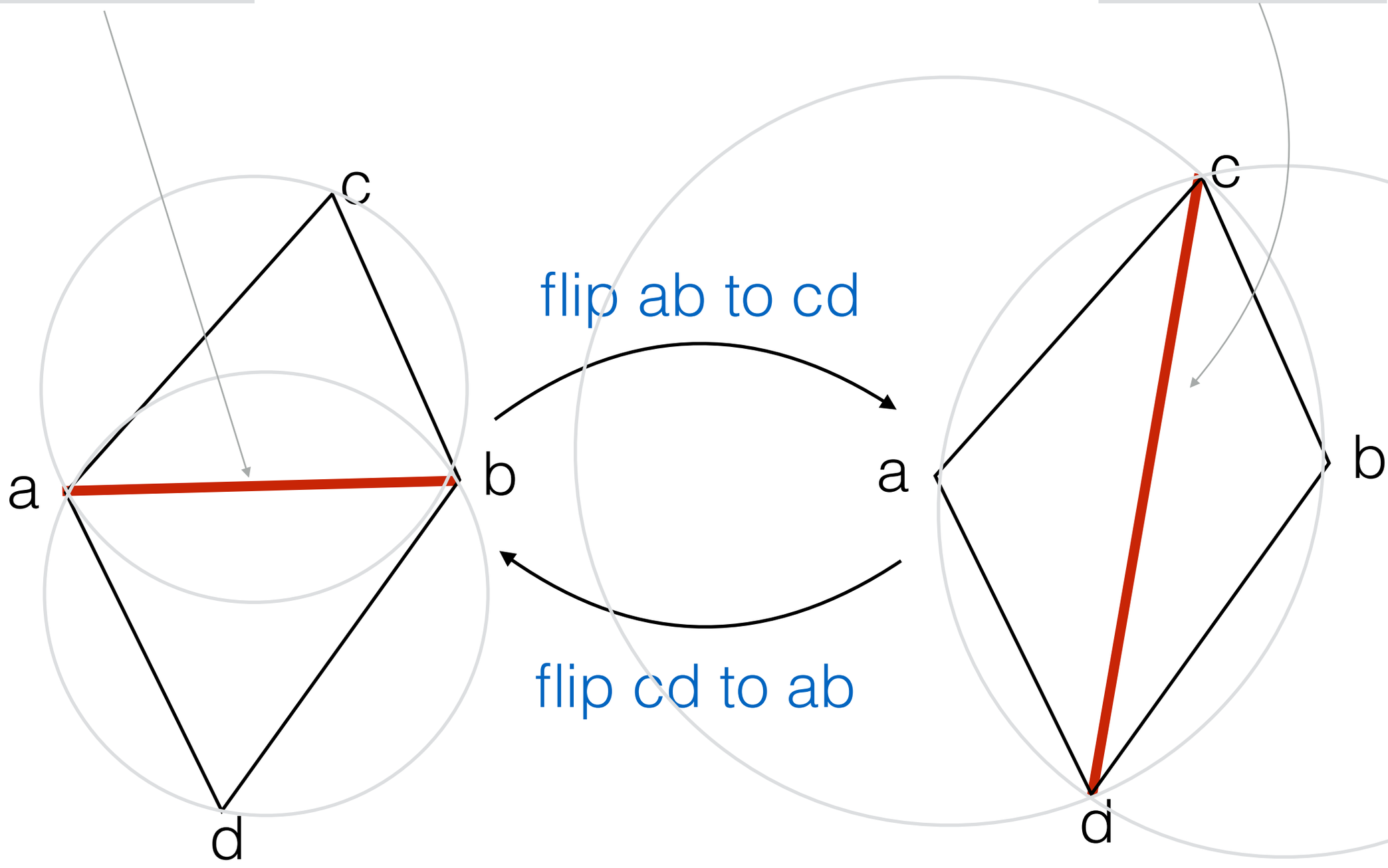


c is outside C(abd)
d is outside C(abc)

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

legal edge

illegal edge

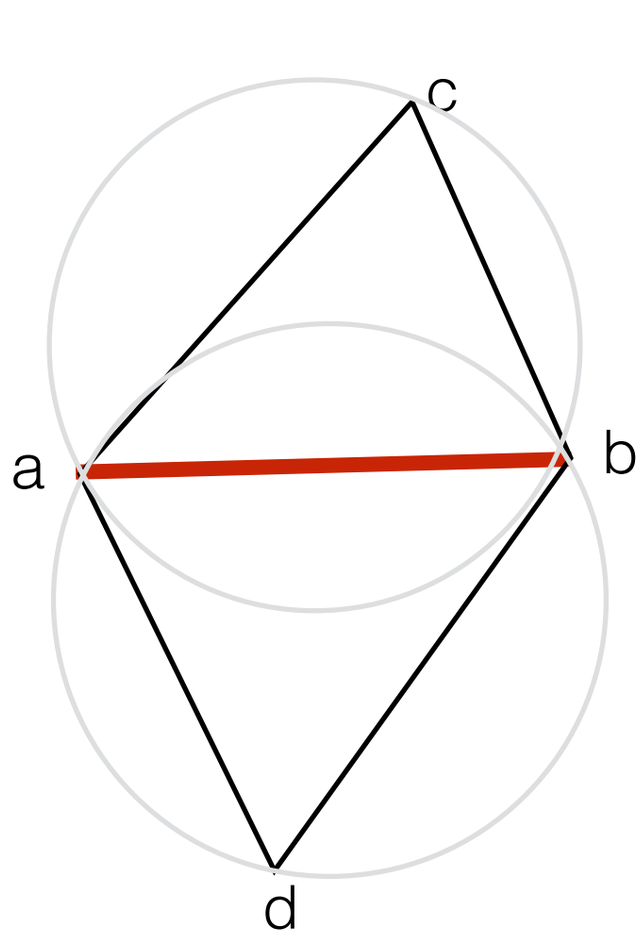


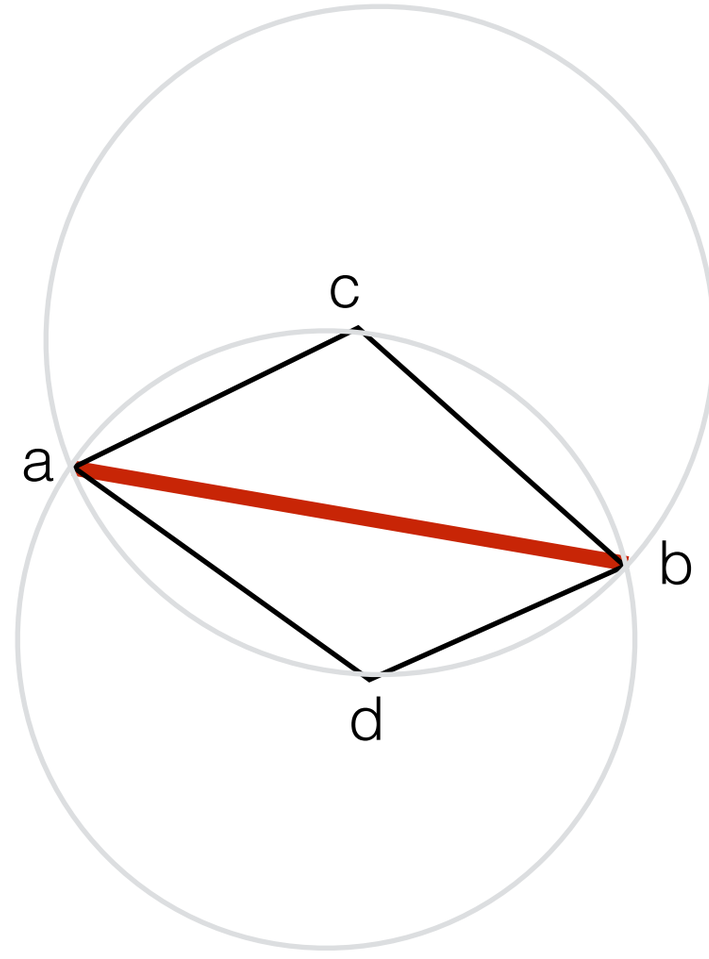
c is outside C(abd)
d is outside C(abc)

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

Legal and Illegal Edges

- An edge ab is **illegal** if its adjacent triangles violate the empty-circumcircle property
  - d is inside C(abc), and c is inside C(abd)
- Otherwise, edge ab is **legal**
  - d on or outside C(abc), and c on or outside C(abd)

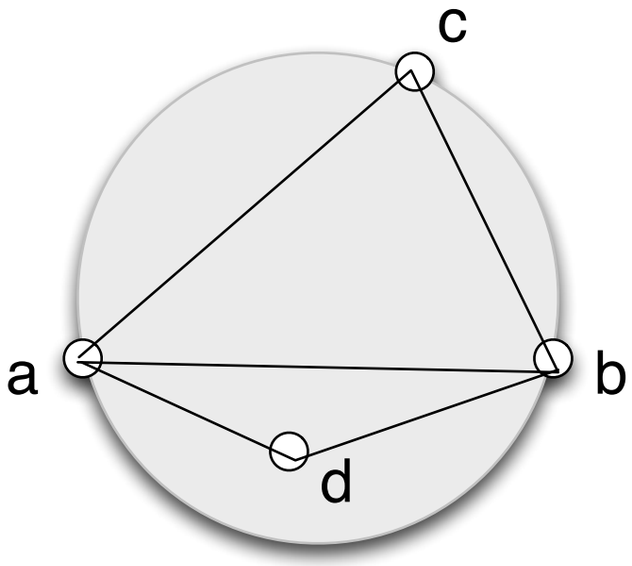




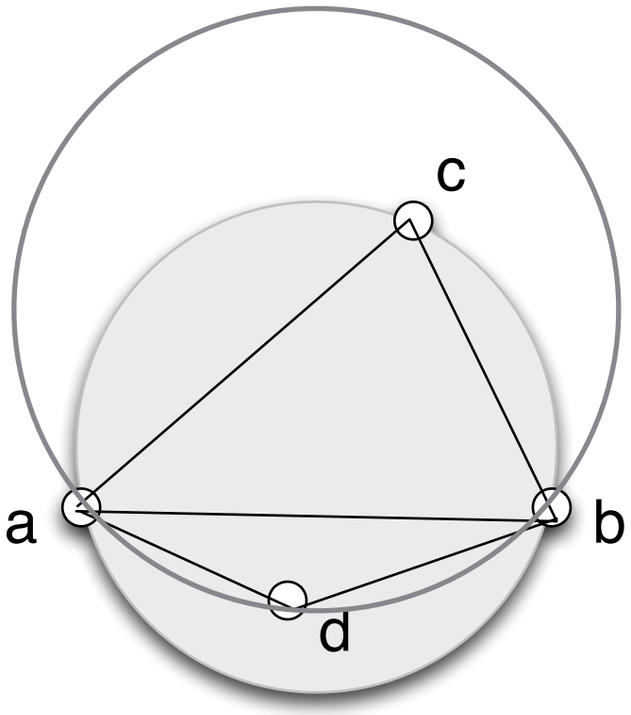
Legal and Illegal Edges

Claim: If d is inside circumcircle of abc, then c is inside circumcircle of abd. So it does not matter which one we check.

Proof: Exercise. Use Thales theorem.



<==>

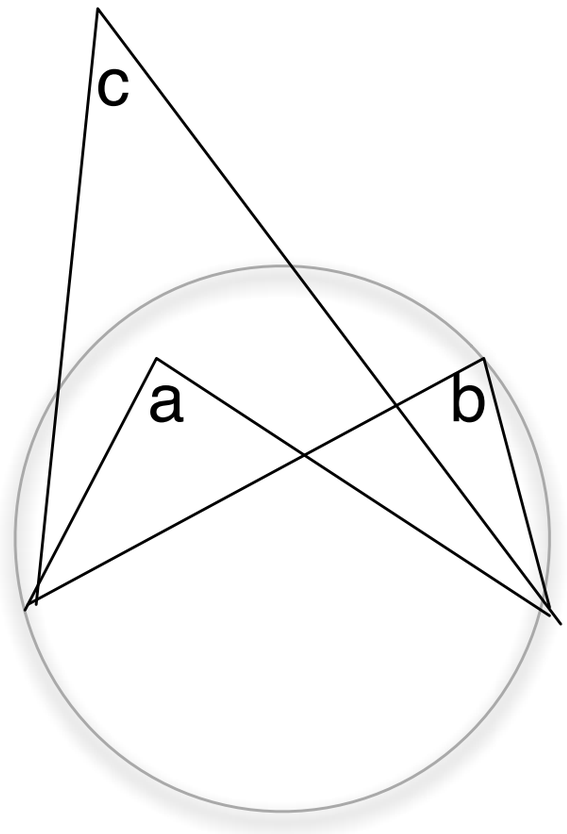


d inside circumcircle of abc

c inside circumcircle of abd

Thales Theorem

- Consider 3 angles with the same endpoints, and a circle through the endpoints
- Then a > b > c



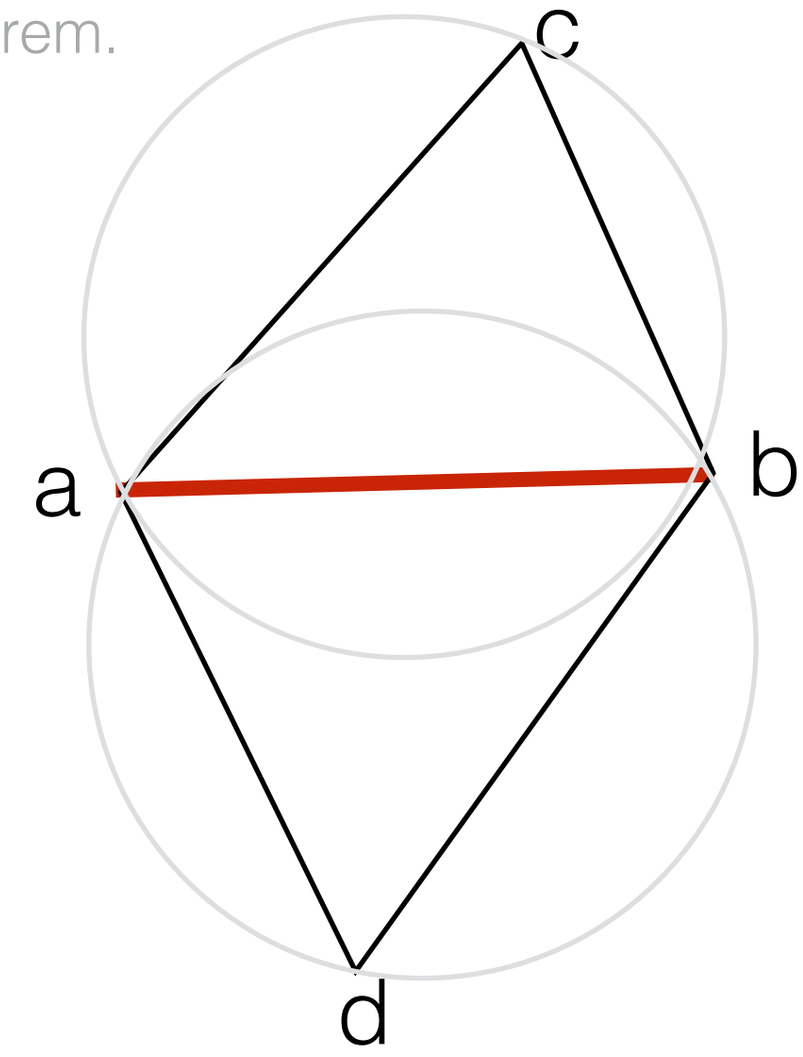
a > b > c

- 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

Legal and Illegal Edges

Claim: Similarly, if d is outside circumcircle of abc, then c is outside circumcircle of abd. So it does not matter which one we check.

Proof: Exercise. Use Thales theorem.

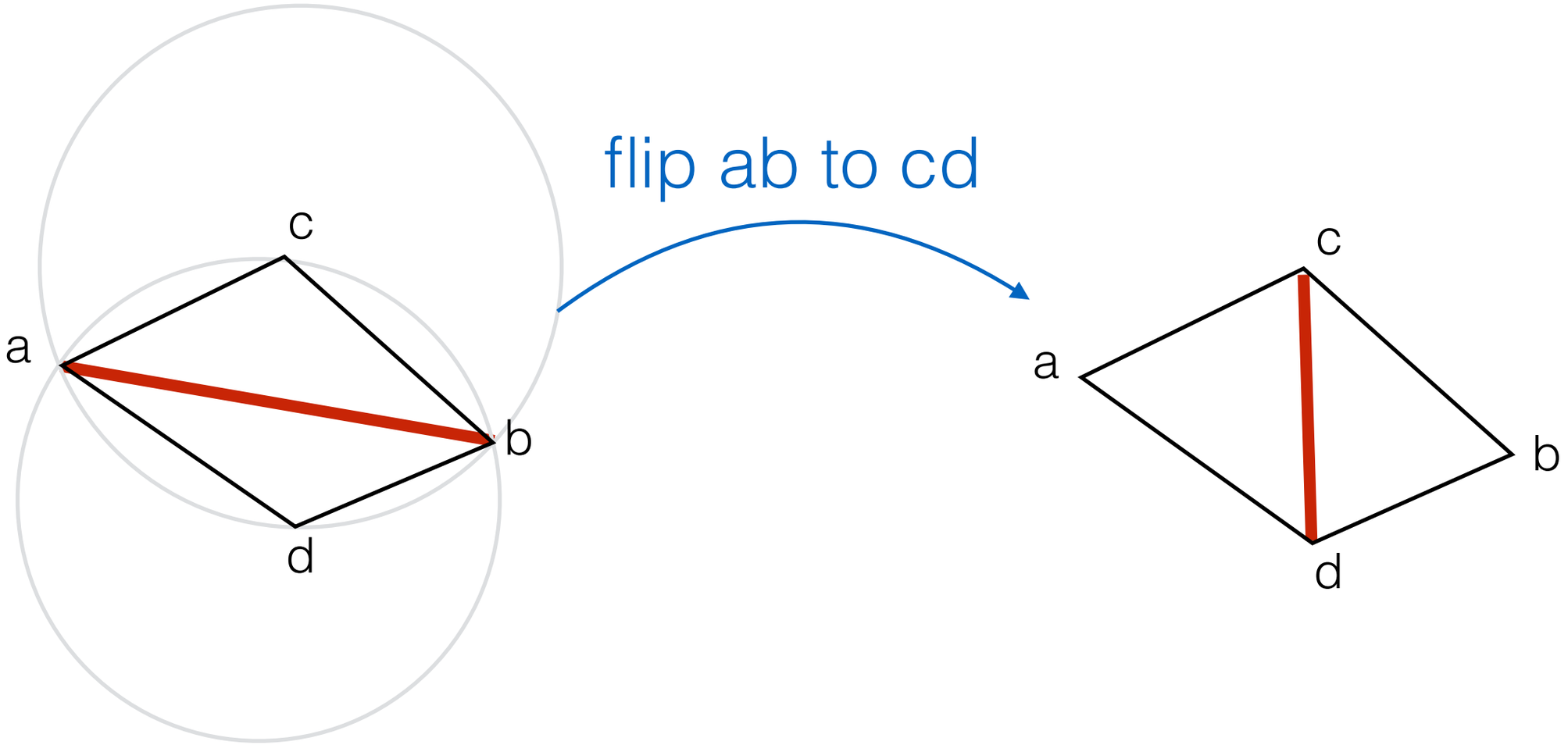


Legal and Illegal Edges

- If d is inside circumcircle of abc, then c is inside circumcircle of abd.
- Similarly, if d is outside circumcircle of abc, then c is outside circumcircle of abd.
- This means that, to check if edge ab is legal, pick one of {c, d} and check wrt the circumcircle of {a,b and the other one of {c,d}}.

Flipping edges

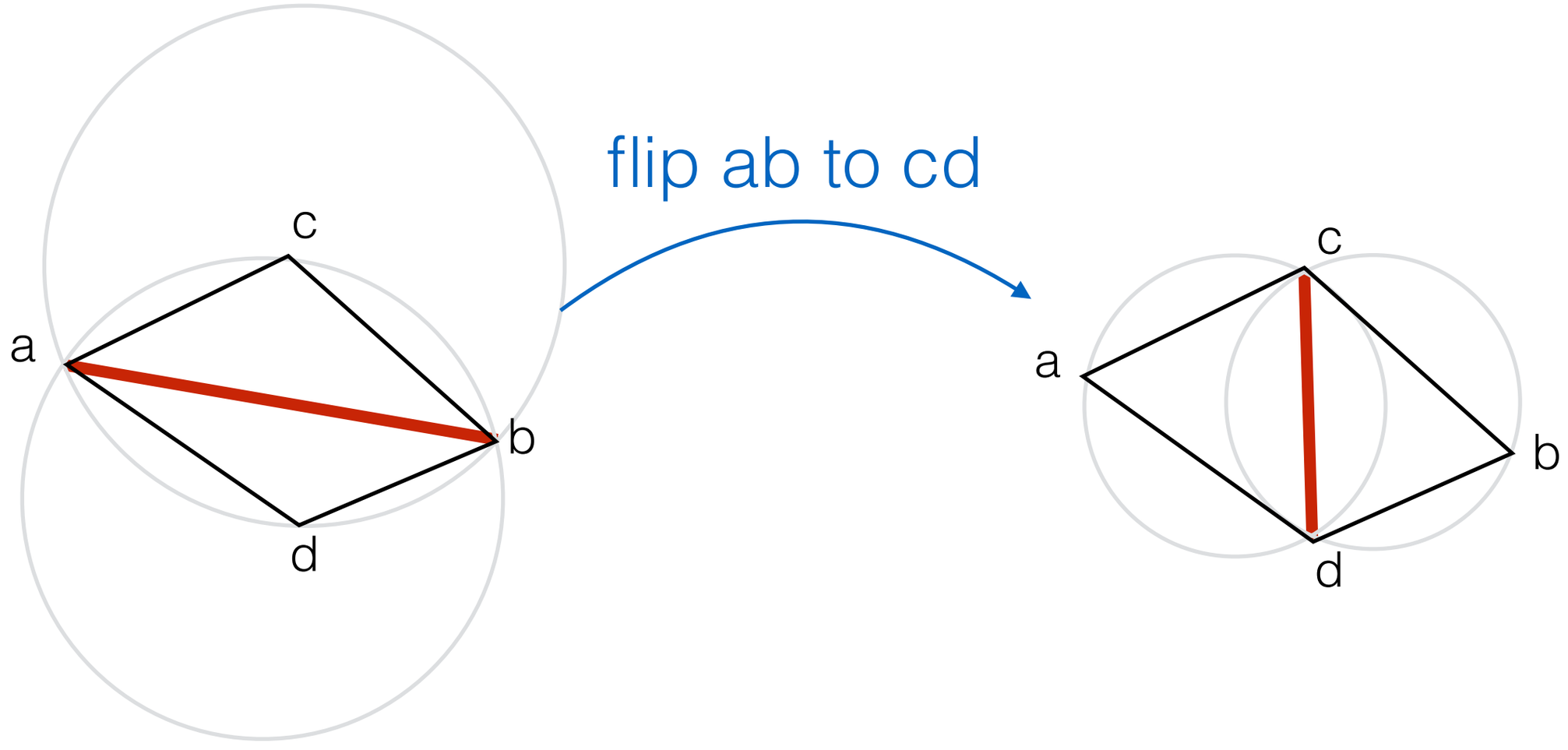
Consider an illegal edge, and flip it.



illegal edge

Flipping edges

Claim: If ab is illegal, then cd is legal.



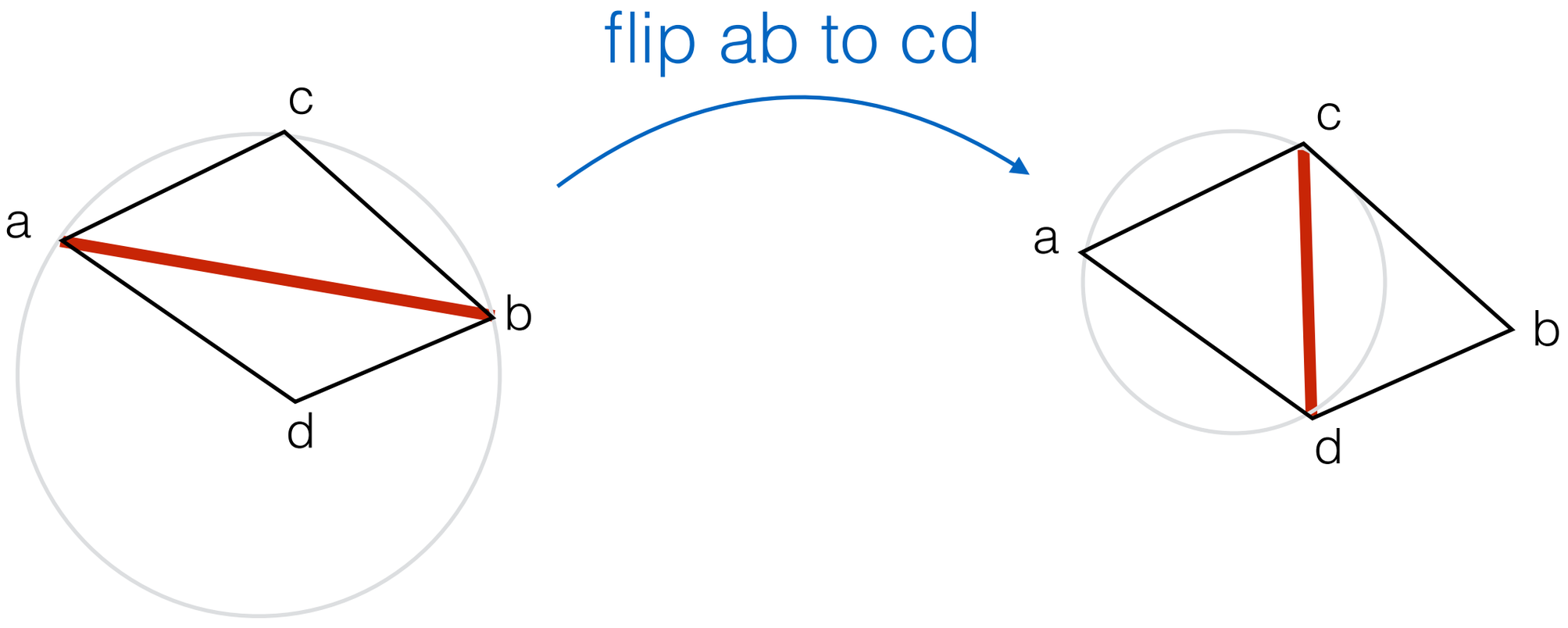
illegal edge

legal edge

Flipping edges

Claim: If ab is illegal, then cd is legal.

Proof: Exercise. Show that if d inside C(abc) then b outside C(abd) using Thales theorem.



illegal edge

legal edge

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 theorem this is the DT.

Constructing DT via edge flipping

Algorithm EdgeFlipDelaunay(P)

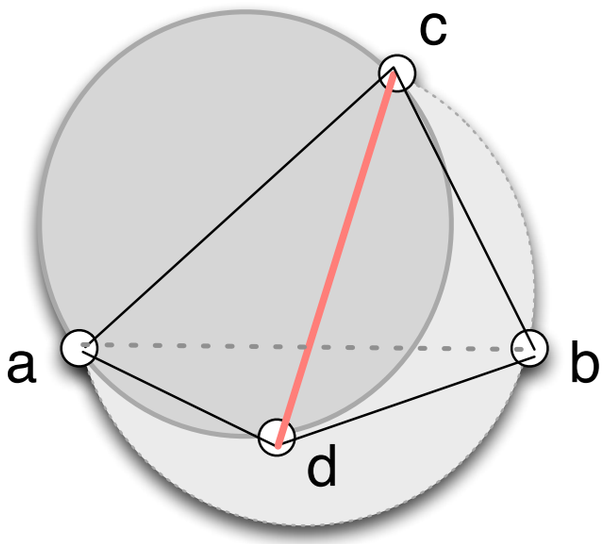
- construct an arbitrary triangulation T
- mark all edges in T and put them in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S and un-mark it
  - if ab not legal:
    - flip edge and update T

      //insert all new edges into S, unless they are marked

    - for each new edge ac, ad, bc, bc: if not marked, push it on S and mark it



Constructing DT via edge flipping

Algorithm EdgeFlipDelaunay(P)

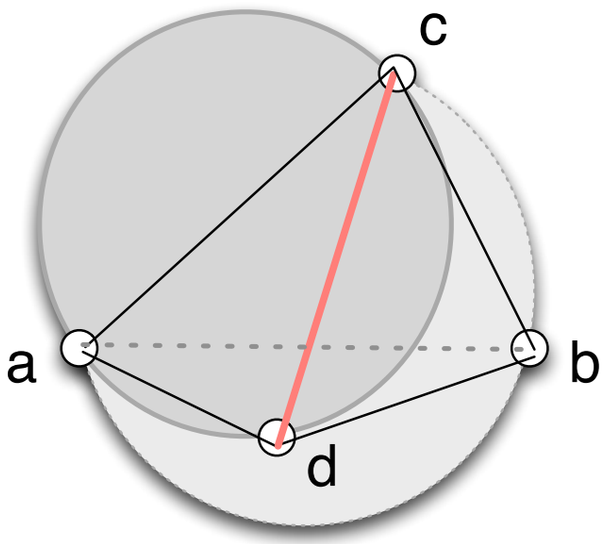
- construct an arbitrary triangulation T
- mark all edges in T and put them in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S and un-mark it
  - if ab not legal:
    - flip edge and update T

      //insert all new edges into S, unless they are marked

    - for each new edge ac, ad, bc, bc: if not marked, push it on S and mark it



Questions:

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

Constructing DT via edge flipping

Algorithm EdgeFlipDelaunay(P)

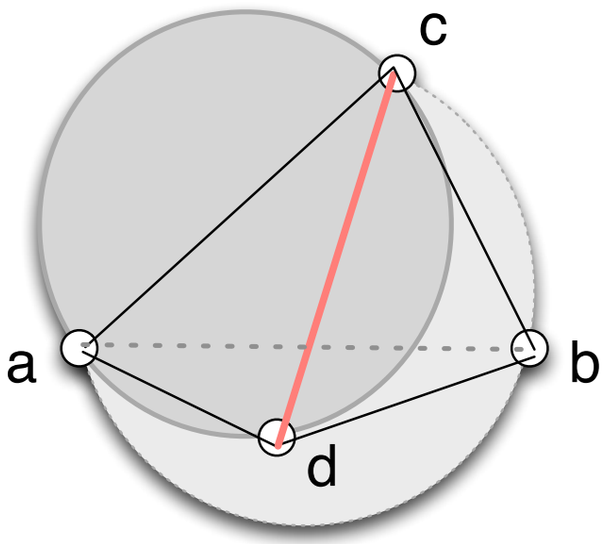
- construct an arbitrary triangulation T
- mark all edges in T and put them in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S and un-mark it
  - if ab not legal:
    - flip edge and update T

      //insert all new edges into S, unless they are marked

    - for each new edge ac, ad, bc, bc: if not marked, push it on S and mark it



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.

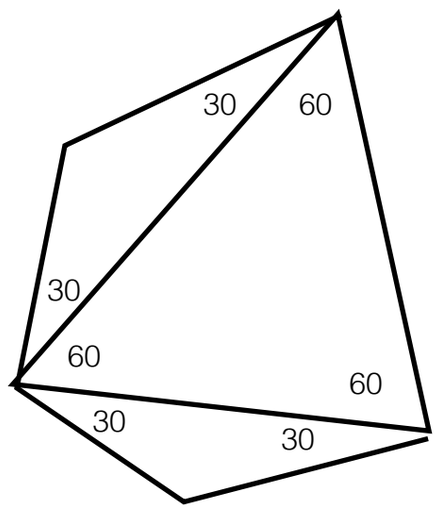
The Angle-vector

- 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
- $A(T) = [a_1 \leqq a_2 \leqq \ldots a \leqq a_{3m}]$ <———————- the angle vector of T

The Angle Vector

- 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
- $A(T) = [a_1 \leqq a_2 \leqq \ldots a \leqq a_{3m}]$ <———————- the angle vector of T

Example

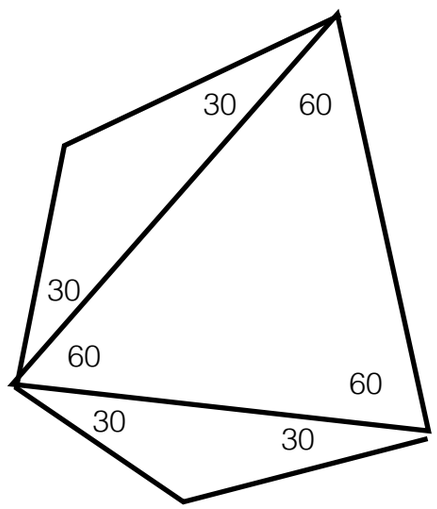


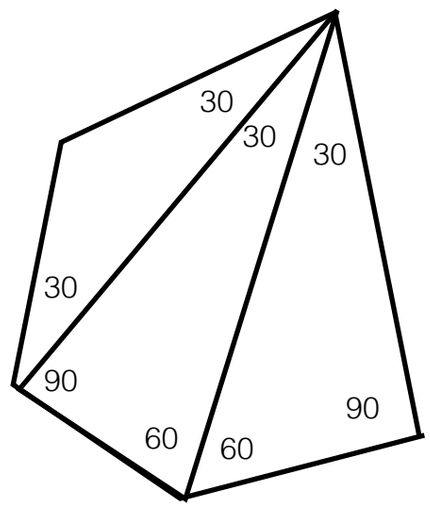
A(T) = [30, 30, 30, 30, 60, 60, 60, 120, 120]

The Angle Vector

- 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
- $A(T) = [a_1 \leqq a_2 \leqq \ldots a \leqq {}_{3m}]$ <———————- the angle vector of T

Example





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

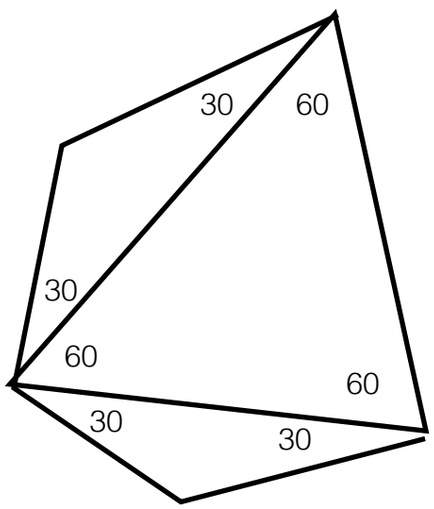
A(T) = [30, 30, 30, 30, 60, 60, 90, 90, 120]

The Angle Vector

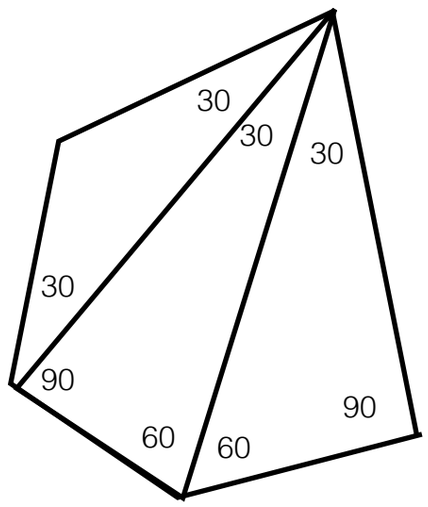
- We compare the angle vectors of triangulations, lexicographically

- We say that A(T) is **lexicographically** smaller than A(T)
    - that is, there exists i, 1 ≦ i ≦ 3m such that
    - $a_j = a_j'$, for all $j < i$
    - $a_i < a_i'$

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]

The 6 smallest angles in T and T' are the same, but the 7th smallest angle is better (larger) in T' than in T.

Comparing Angle Vectors

Comparing Angle Vectors

- We consider all possible triangulations of P

Comparing Angle Vectors

- We consider all possible triangulations of P
- We compare their angle vectors lexicographically

Comparing Angle Vectors

- We consider all possible triangulations of P
- We compare their angle vectors lexicographically
  - This means that the largest triangulation is the one that has the largest “first” angle, i.e. the largest minimum angle

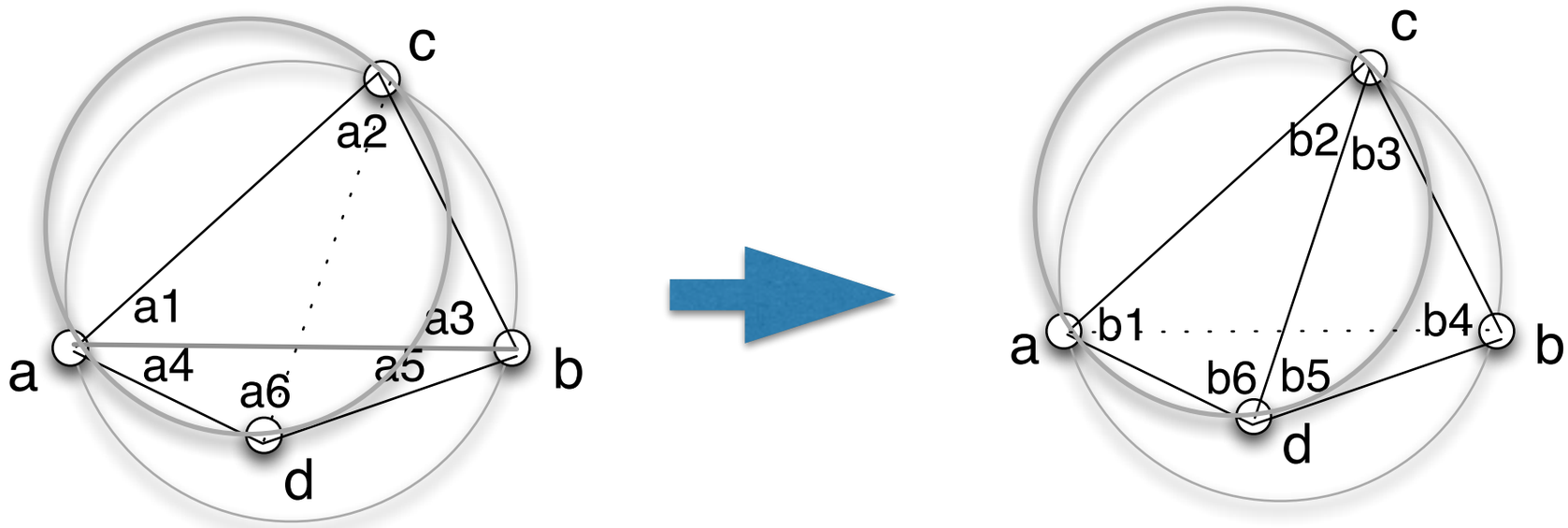
Comparing Angle Vectors

- We consider all possible triangulations of P
- We compare their angle vectors lexicographically
  - This means that the largest triangulation is the one that has the largest “first” angle, i.e. the largest minimum angle
  - If there is more than one triangulation with the same smallest angle, then the largest one will be the one with the largest second-smallest angle

Comparing Angle Vectors

- We consider all possible triangulations of P
- We compare their angle vectors lexicographically
  - This means that the largest triangulation is the one that has the largest “first” angle, i.e. the largest minimum angle
  - If there is more than one triangulation with the same smallest angle, then the largest one will be the one with the largest second-smallest angle
  - And so on

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

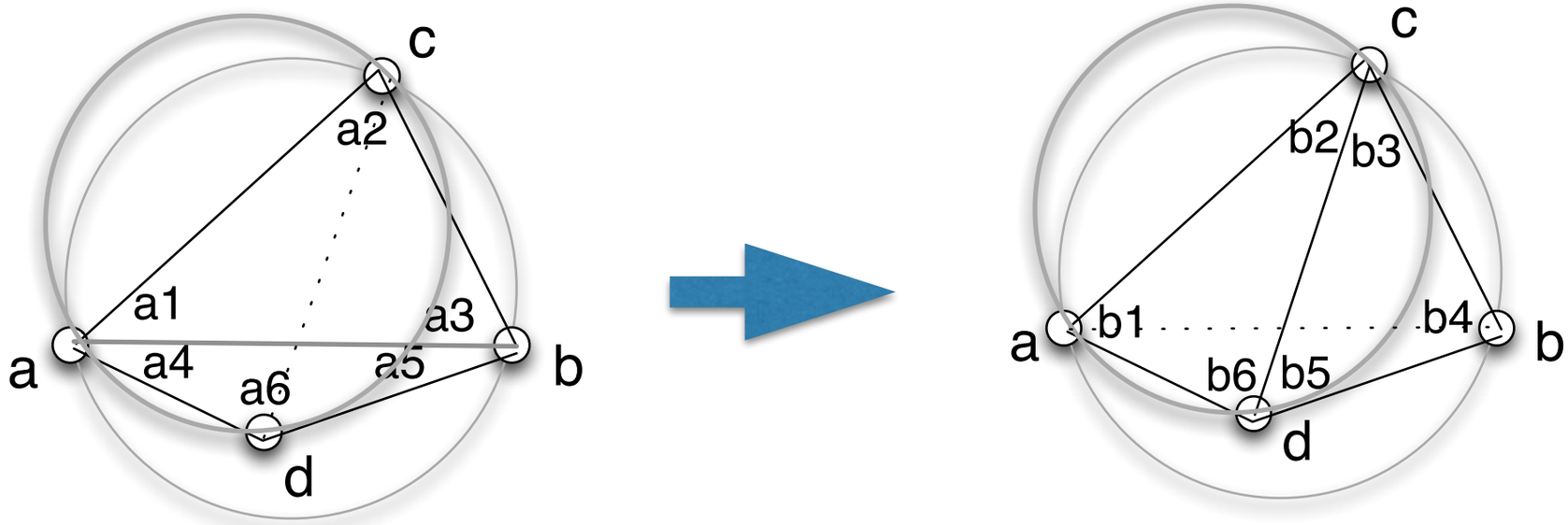


before : A(T) = [a1, a2, a3, a4, a5, a6]

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

- We look only at the angles inside the two triangles
- We can ignore the rest of the angle vector because it does not change after the flip

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]

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

Proof:

b1 = a1+a4 > a1

b2 > a5 (b outside C(acd))

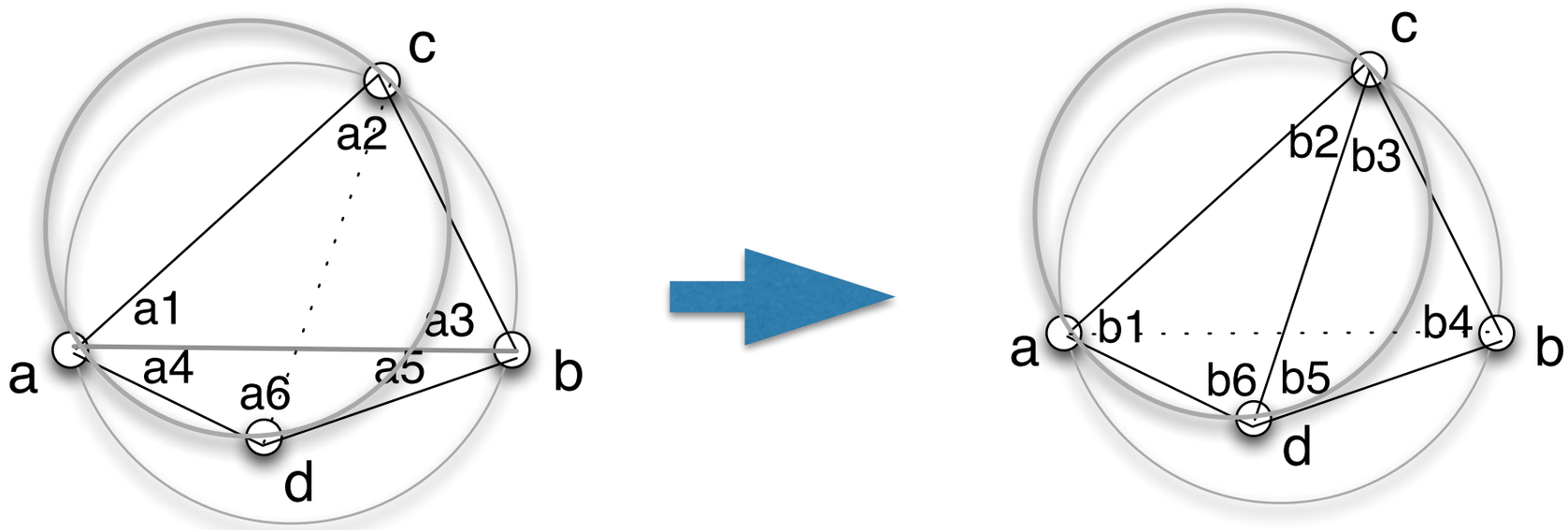
b3 > … similar

b4 = a3 + a5 > a3

b5 > … similar

b6 > a3 (b outside C(acd))

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]

Claim: 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')

DT and angles

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

Proof: DT has only legal edges. Any other flip would cause the angle vector to decrease.

DT and angles

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

- What this means

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

- What this means
    - DT is the triangulation with the largest minimum angle

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

- What this means
  - DT is the triangulation with the largest minimum angle
  - If another triangulation with the same min angle exists, then DT will have a larger 2nd smallest angle.

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

- What this means
  - DT is the triangulation with the largest minimum angle
  - If another triangulation with the same min angle exists, then DT will have a larger 2nd smallest angle.
  - If another triangulation exists that has the same two smallest angles, then DT will have a larger 3rd smallest angle.

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

- What this means
  - DT is the triangulation with the largest minimum angle
  - If another triangulation with the same min angle exists, then DT will have a larger 2nd smallest angle.
  - If another triangulation exists that has the same two smallest angles, then DT will have a larger 3rd smallest angle.
  - and so on.

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

- What this means
  - DT is the triangulation with the largest minimum angle
  - If another triangulation with the same min angle exists, then DT will have a larger 2nd smallest angle.
  - If another triangulation exists that has the same two smallest angles, then DT will have a larger 3rd smallest angle.
  - and so on.
- When P is so that no 4 are co-circular, there exists a unique legal triangulation, which is the unique Delaunay triangulation. This triangulation maximizes the minimum angle.

DT and angles

- Thus flipping an illegal edge to a legal edge increases the angle vector
- And symmetrically: flipping a legal edge to an illegal edge decreases the angle vector.

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

Proof: DT has only legal edges. Any other flip would cause the angle vector to decrease.

- What this means
  - DT is the triangulation with the largest minimum angle
  - If another triangulation with the same min angle exists, then DT will have a larger 2nd smallest angle.
  - If another triangulation exists that has the same two smallest angles, then DT will have a larger 3rd smallest angle.
  - and so on.
- When P is so that no 4 are co-circular, there exists a unique legal triangulation, which is the unique Delaunay triangulation. This triangulation maximizes the minimum angle.

Does this terminate?

Algorithm EdgeFlipDelaunay(P)

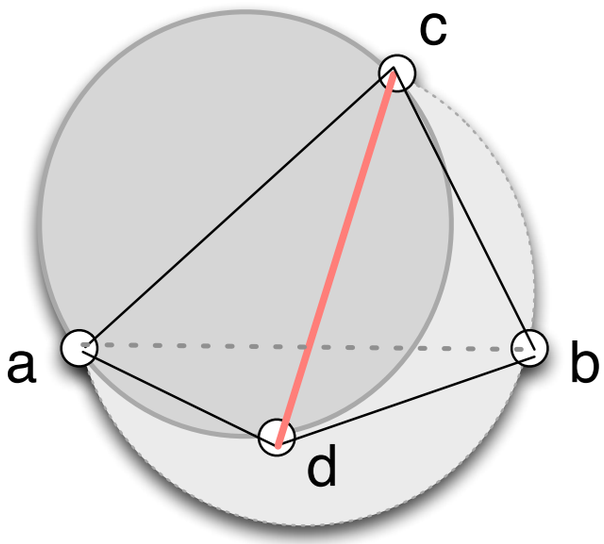
- construct an arbitrary triangulation T
- mark all edges in T and put them in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S and un-mark it
  - if ab not legal:
    - flip edge and update T

      //insert all new edges into S, unless they are marked

    - for each new edge ac, ad, bc, bc: if not marked, push it on S and mark it



We'll argue based on A(T)

Does this terminate?

Algorithm EdgeFlipDelaunay(P)

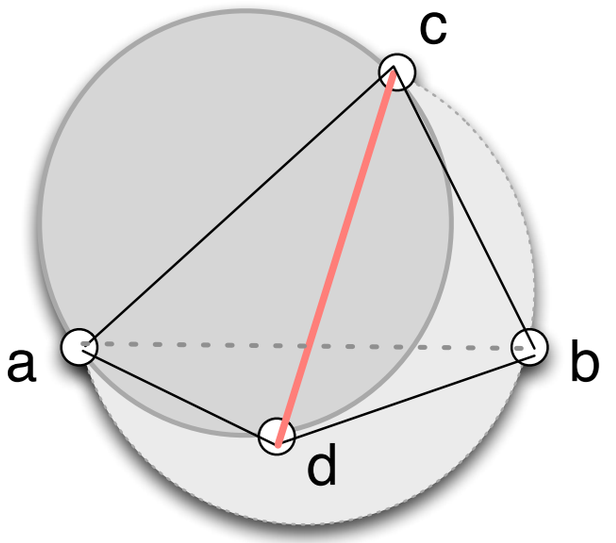
- construct an arbitrary triangulation T
- mark all edges in T and put them in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S and un-mark it
  - if ab not legal:
    - flip edge and update T

    //insert all new edges into S, unless they are marked

    - for each new edge ac, ad, bc, bc: if not marked, push it on S and mark it



We'll argue based on A(T)

- Every flip increases the angle vector.
- There is a finite number of different triangulations.

Running time?

Algorithm EdgeFlipDelaunay(P)

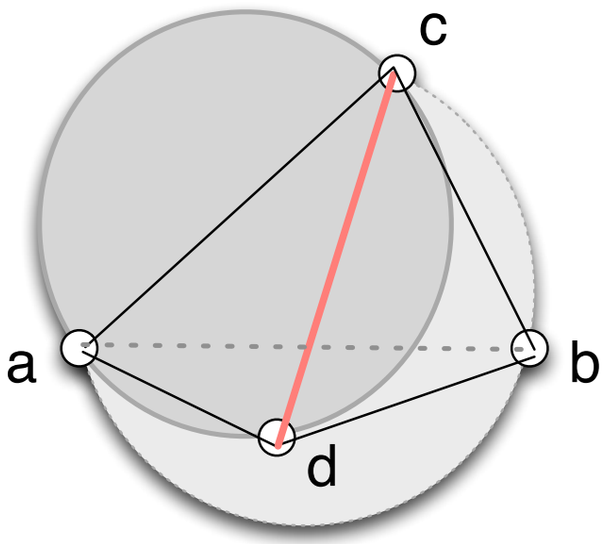
- construct an arbitrary triangulation T
- mark all edges in T and put them in S

  //S holds all edges that are potentially illegal

- while S not empty
  - pop edge ab from S and un-mark it
  - if ab not legal:
    - flip edge and update T

    //insert all new edges into S, unless they are marked

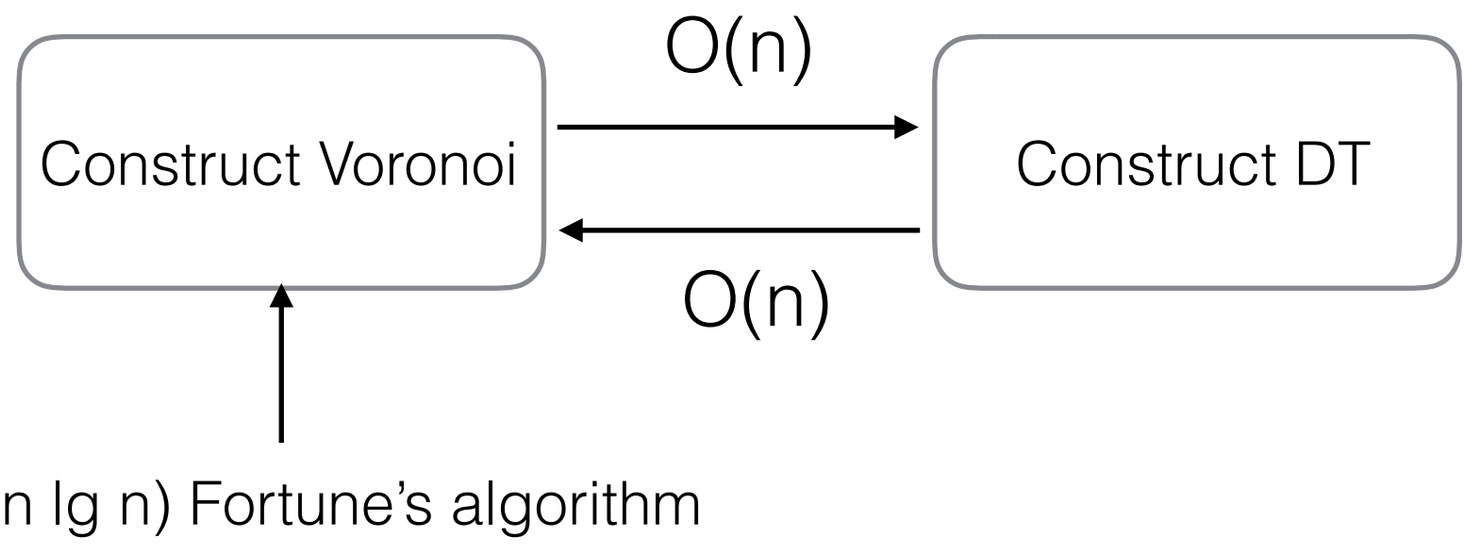
    - for each new edge ac, ad, bc, bc: if not marked, push it on S and mark it



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

Constructing DT

- O(n lg n) via Voronoi



- $O(n^2)$ via ege flipping

- O(n lg n) expected via Randomized Incremental Construction (RIC)

DT via Randomized Incremental Construction

Computing DT via RIC

$P = \{p_1, p_2, \ldots, p_n\}$ set of points in the plane
no 4 points co-circular

RIC (DT Randomized Incremental Construction)

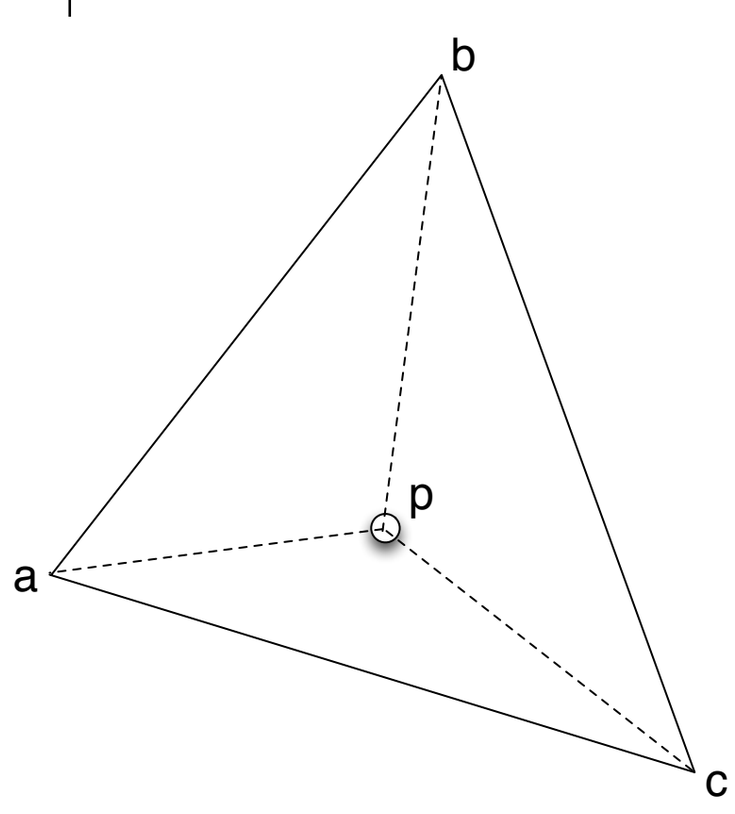
- Let $(p_1, p_2, p_3, \ldots p_n)$ be a random permutation of P

  //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



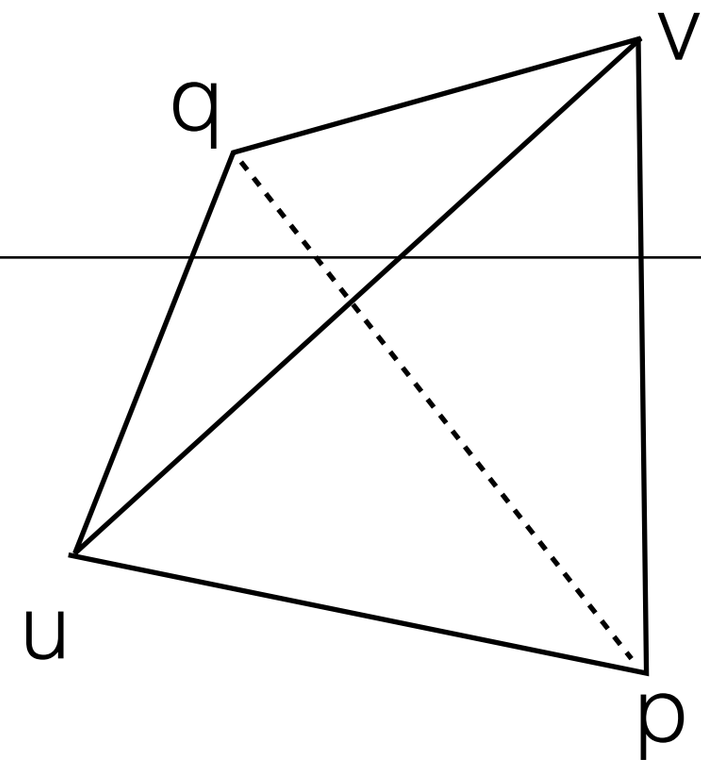
P = {p_1, p_2,…,p_n} set of points in the plane
no 4 points co-circular

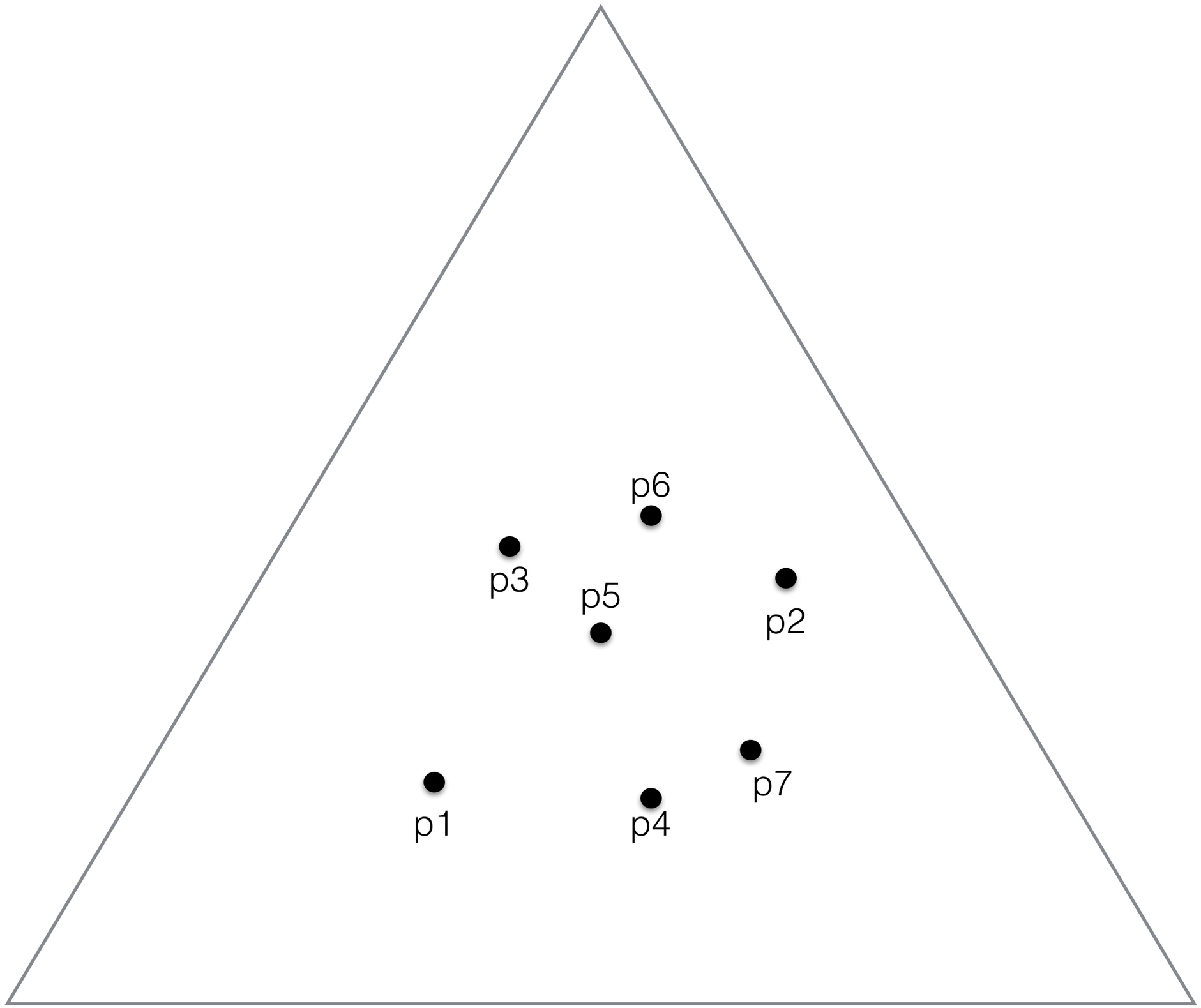
Computing DT: RIC

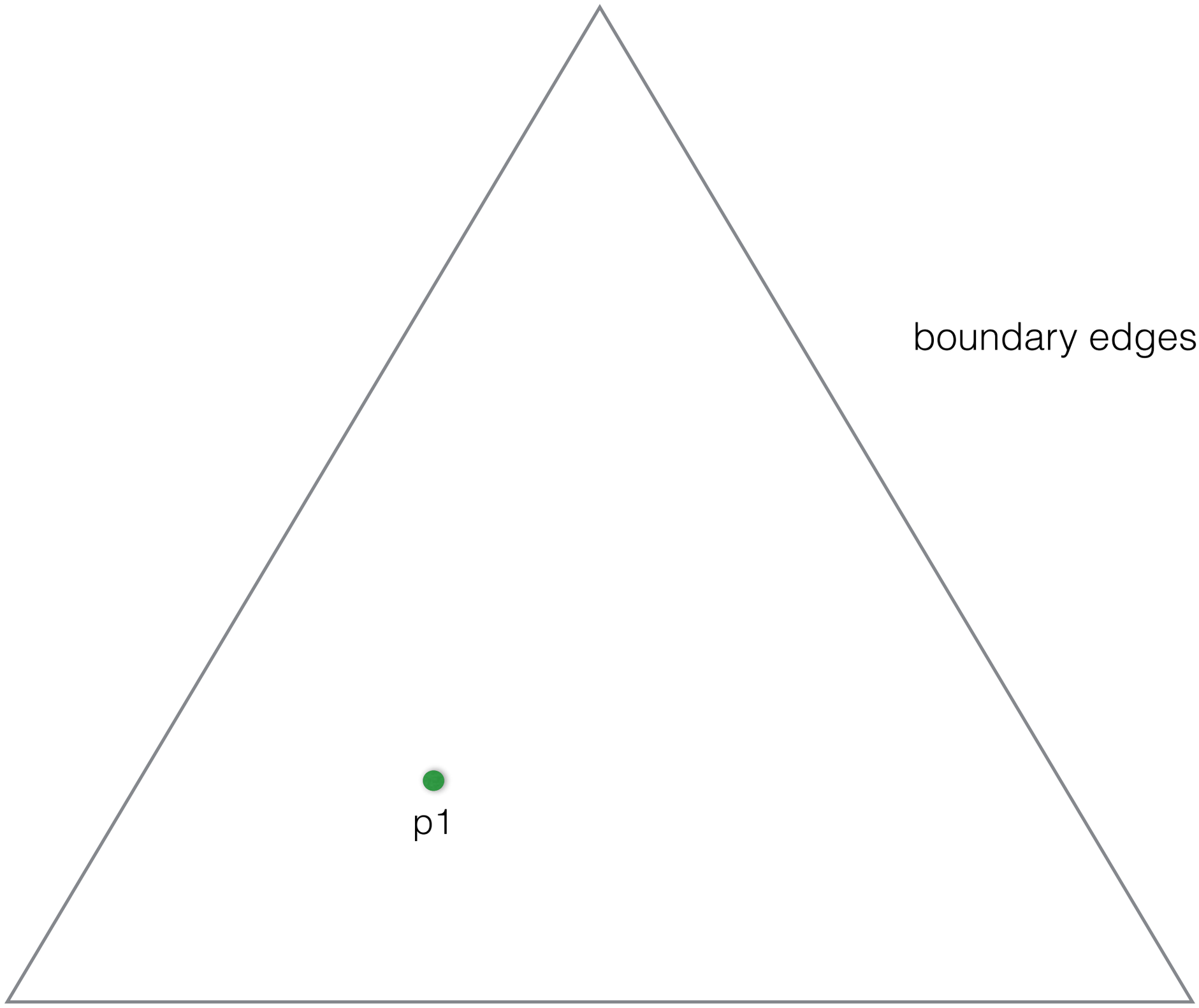
LegalizeEdge(p, uv, T)

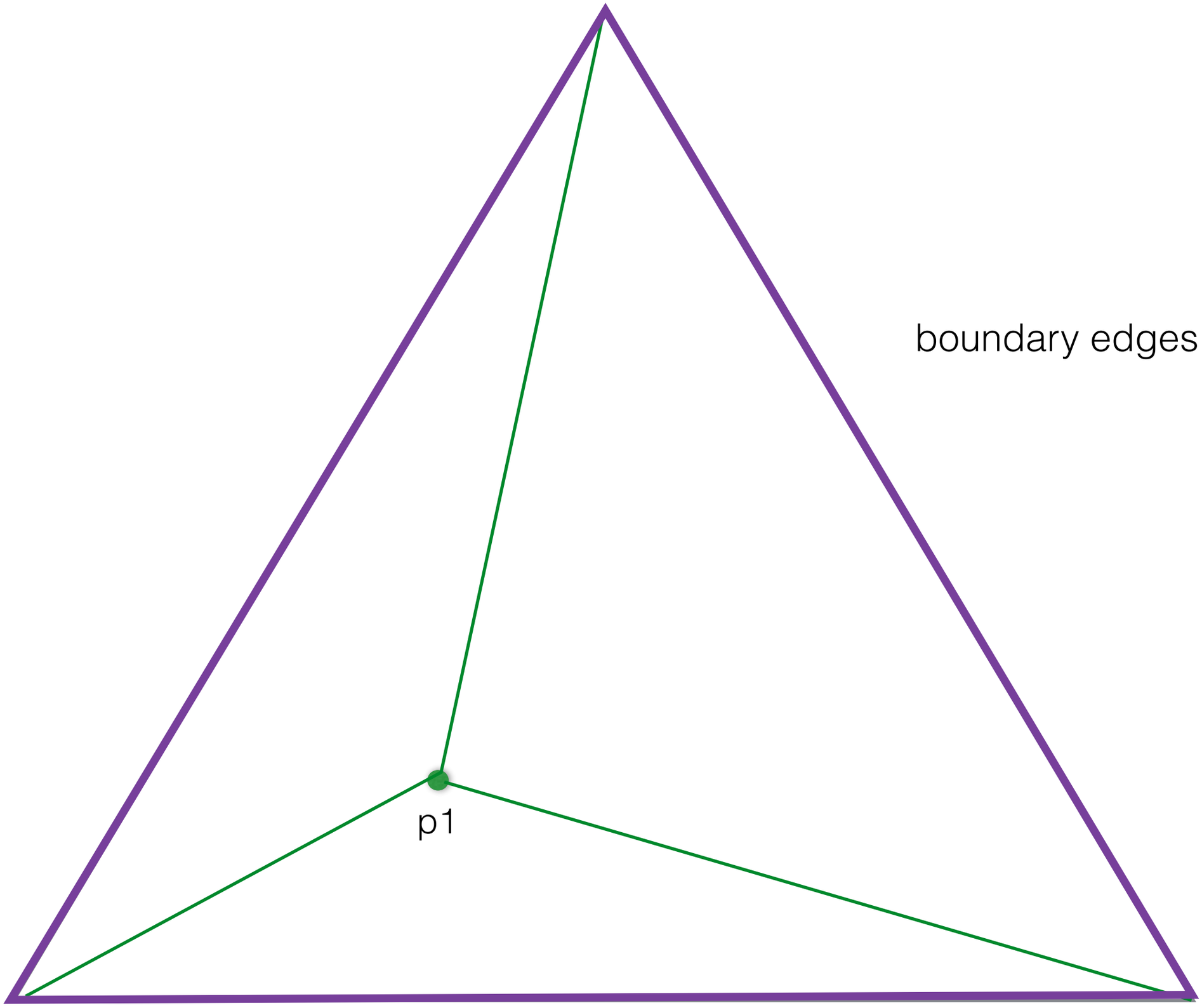
//point p was inserted, and puv is a new triangle; edge uv may need to be flipped

- 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)

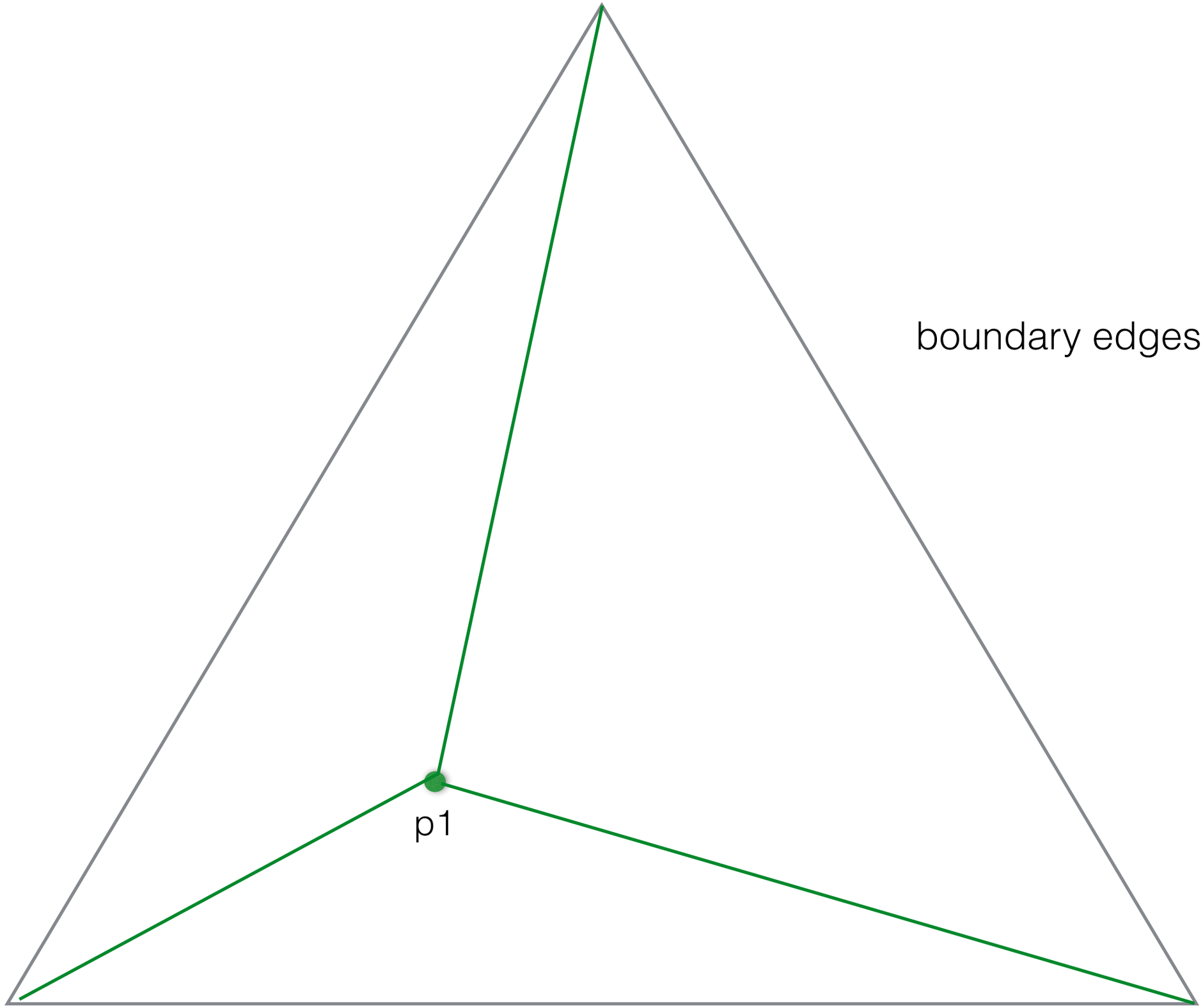



p6
p3
p5
p2
p7
p1
p4

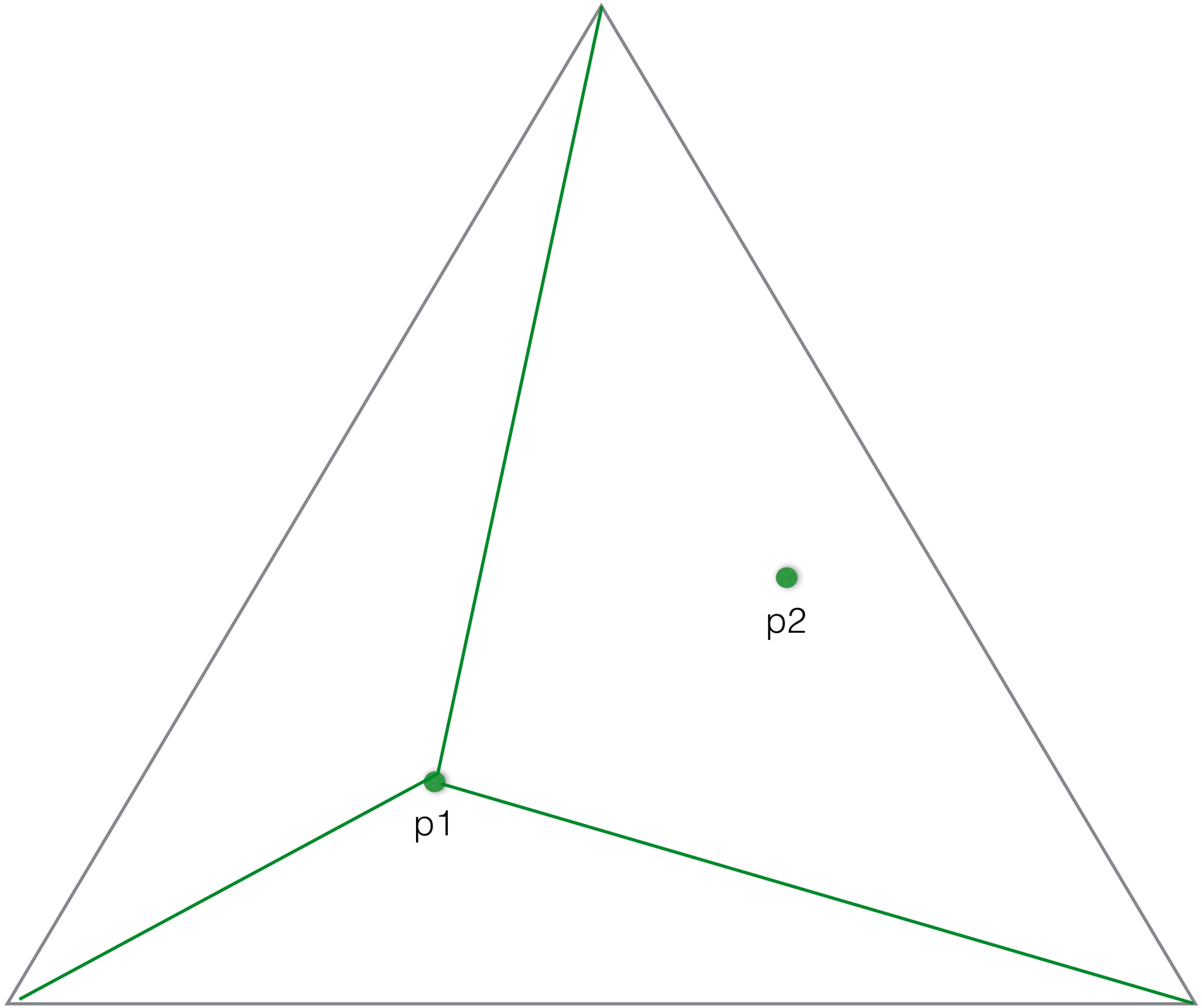

boundary edges are legal
p1

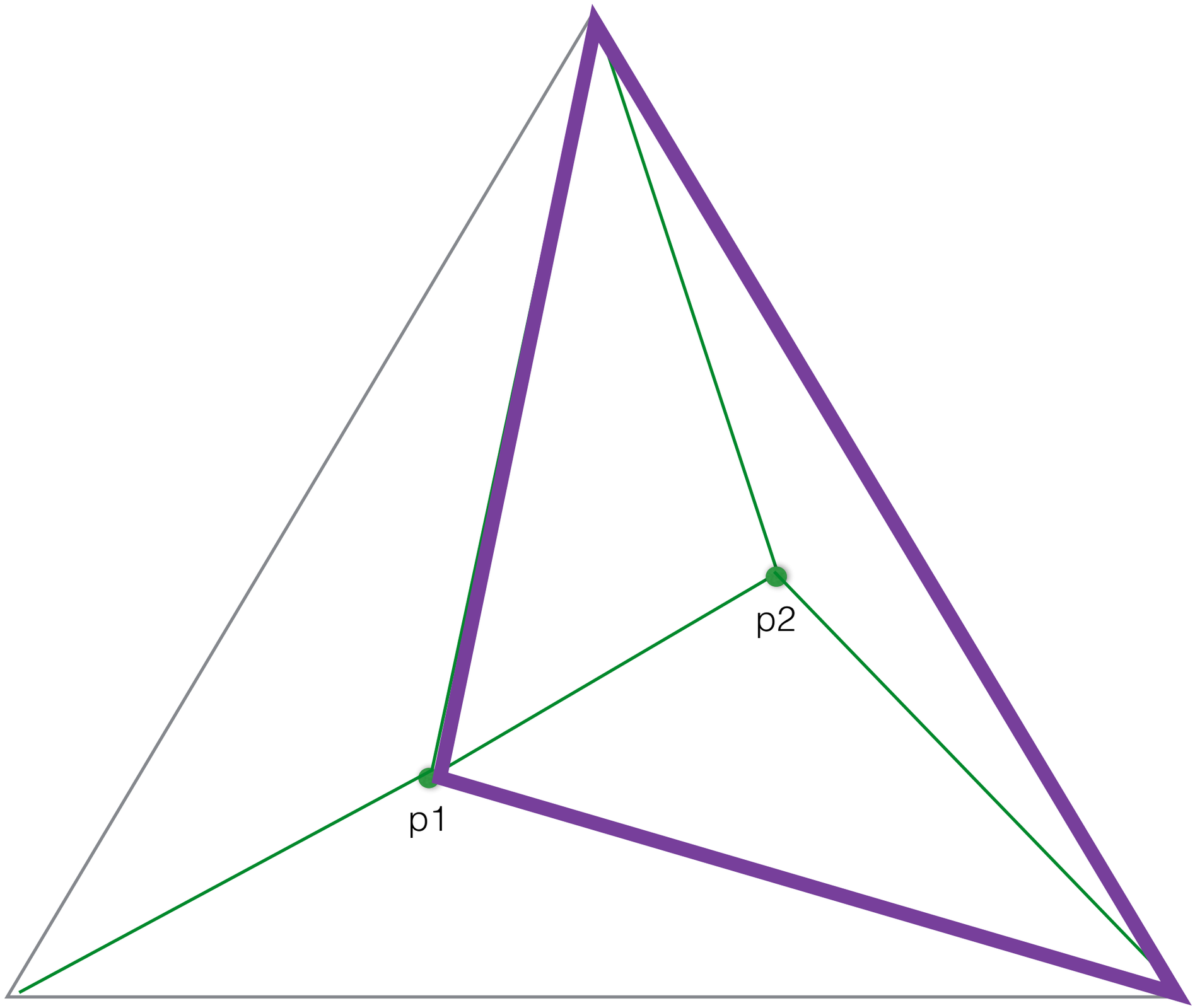


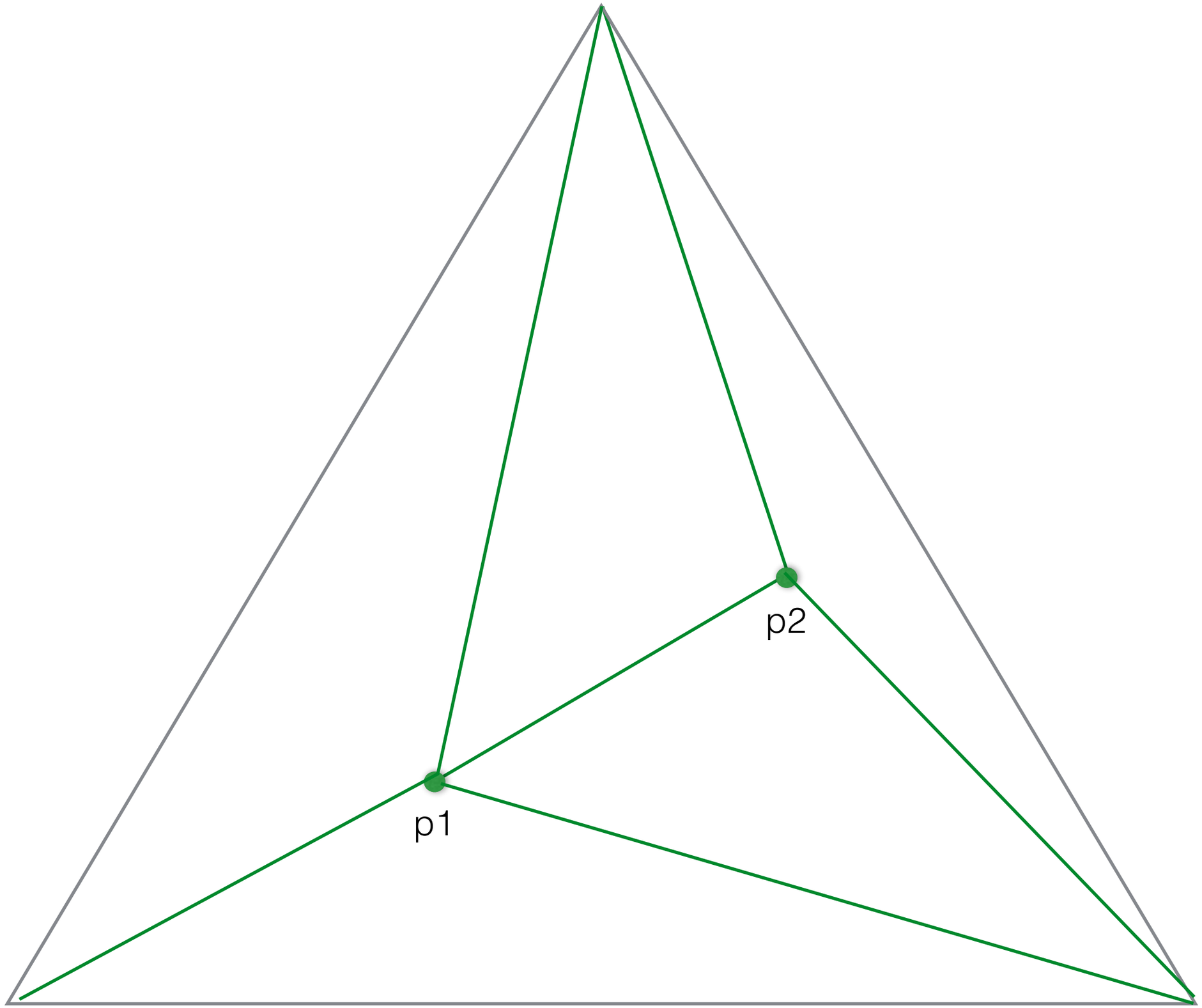
boundary edges are legal

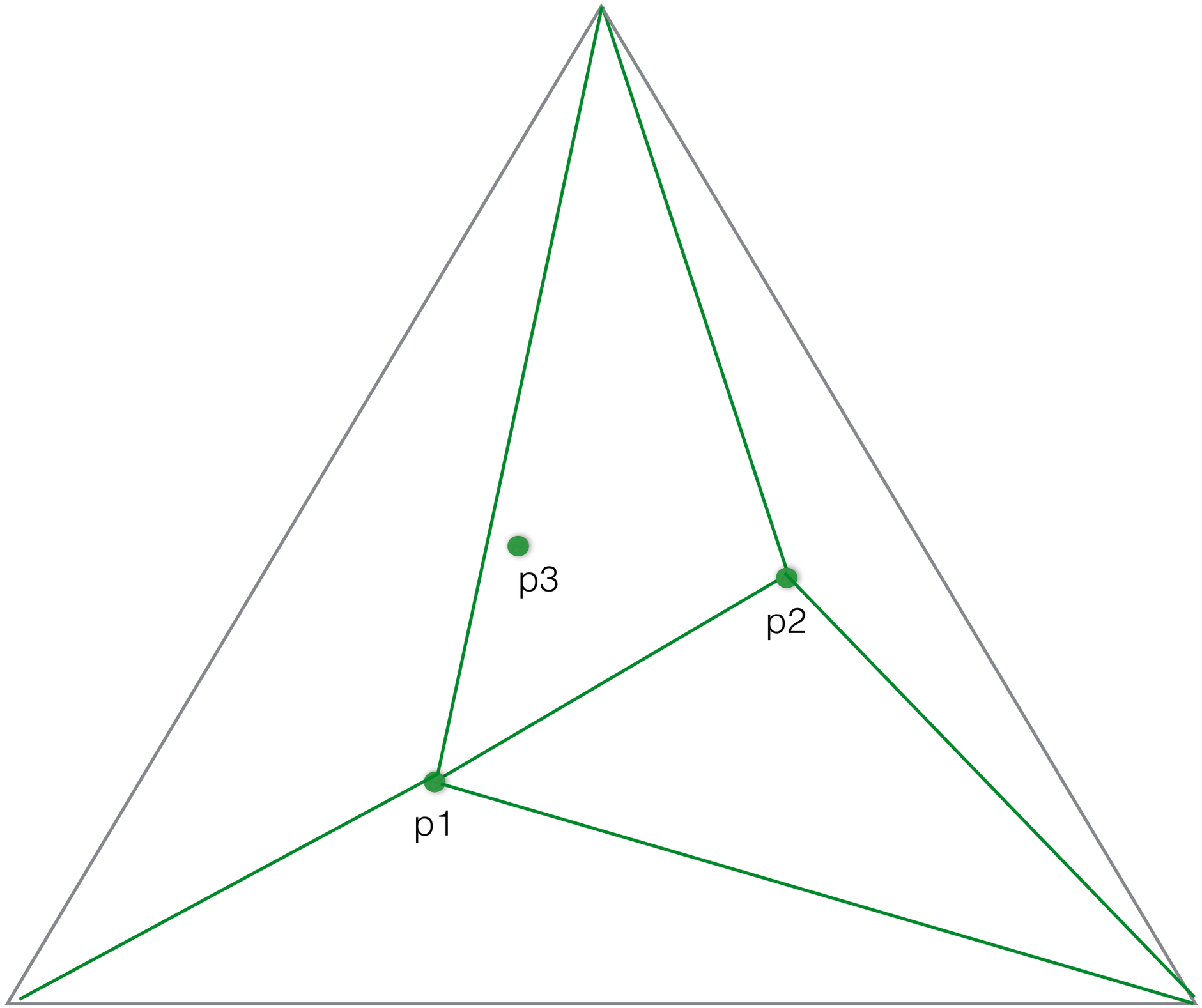


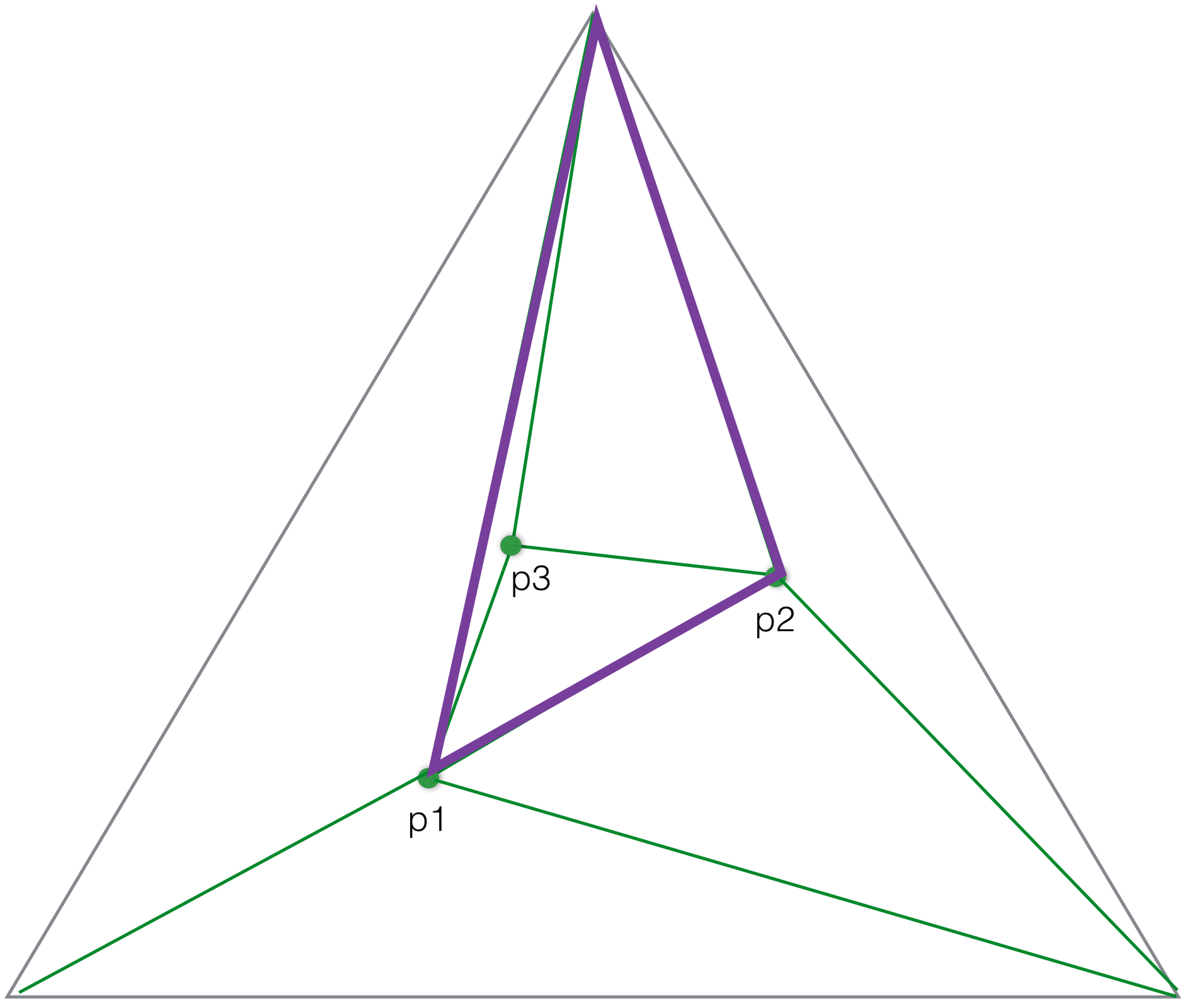
boundary edges are legal

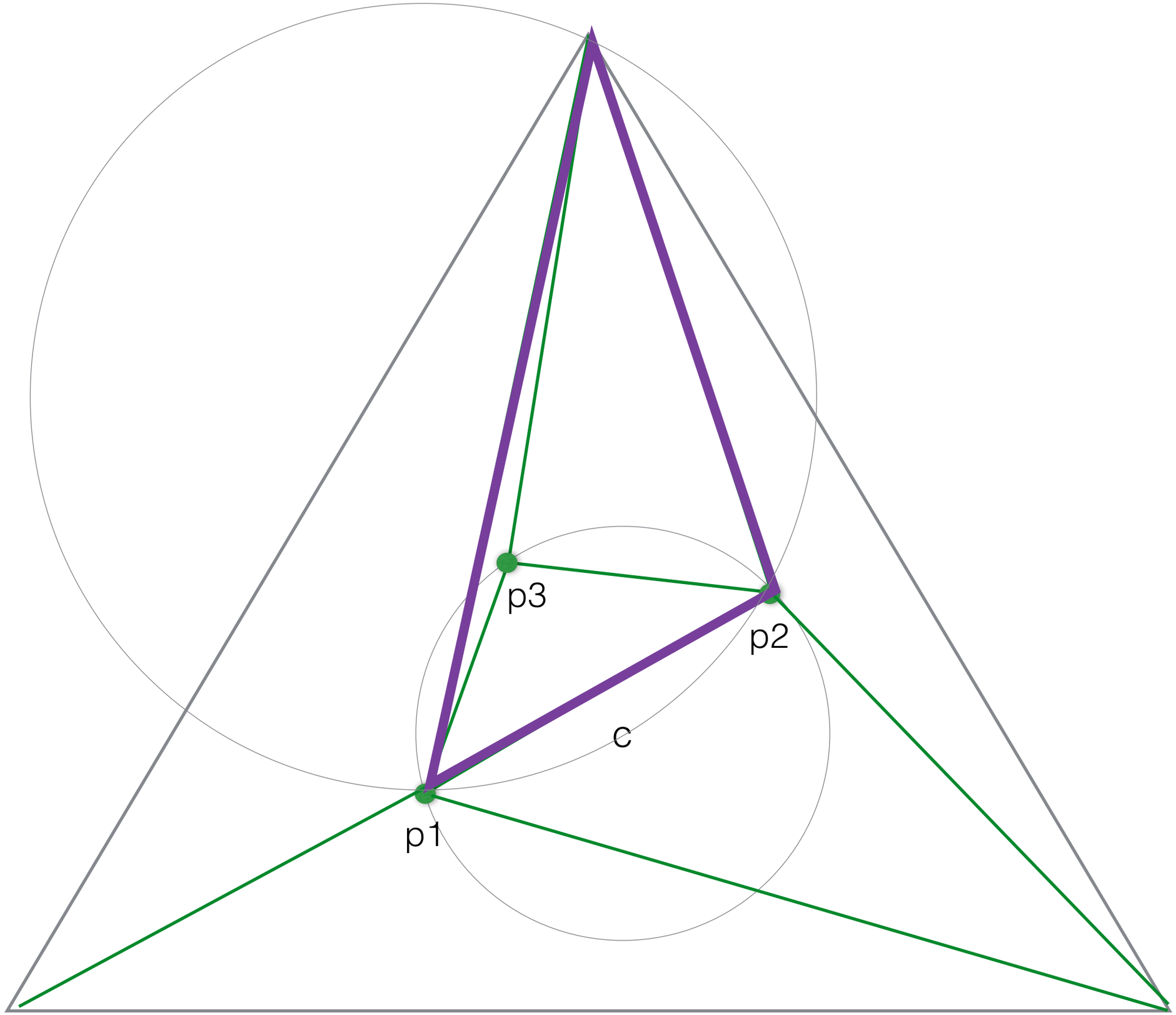

p2
p1

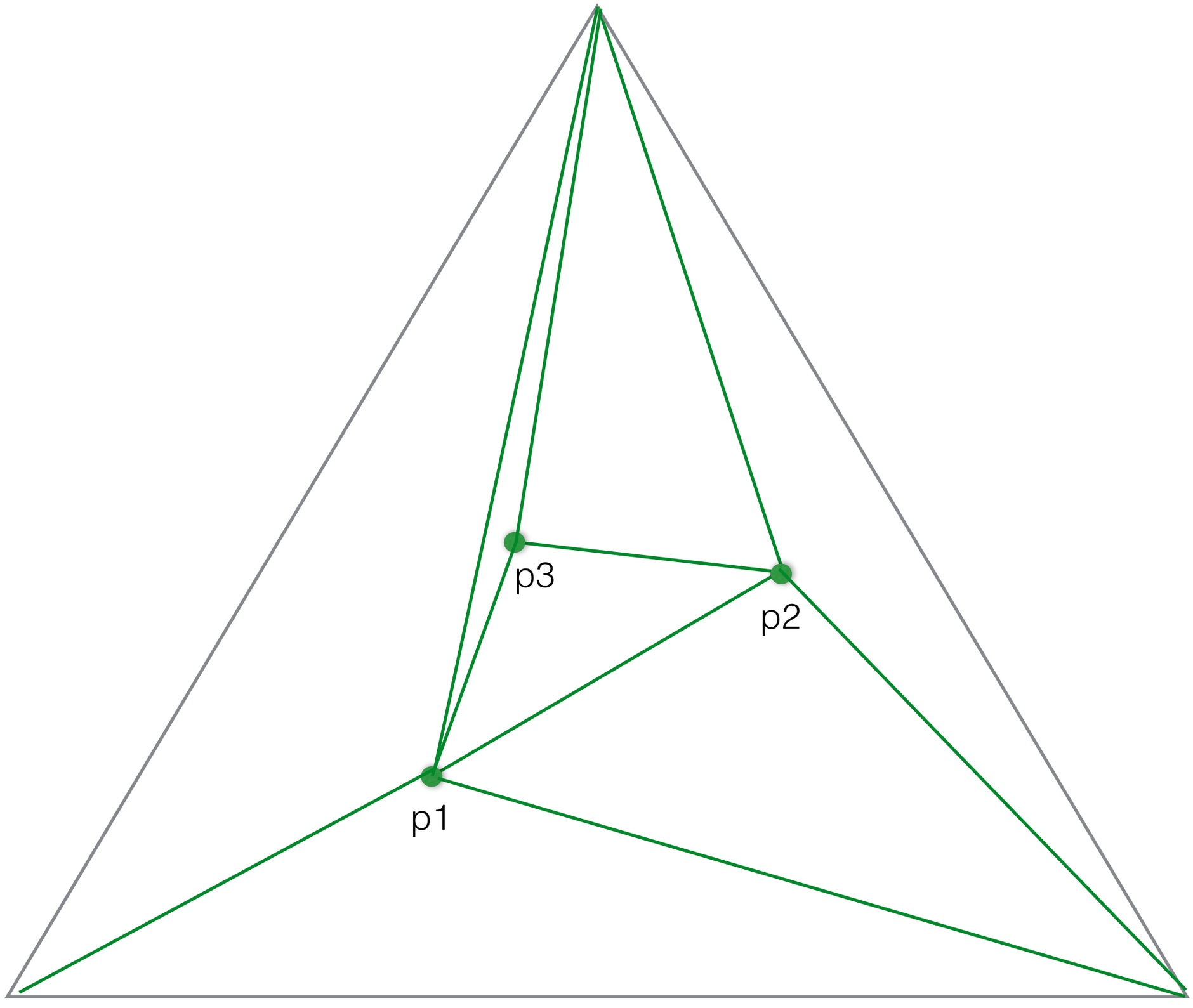

p2
p1

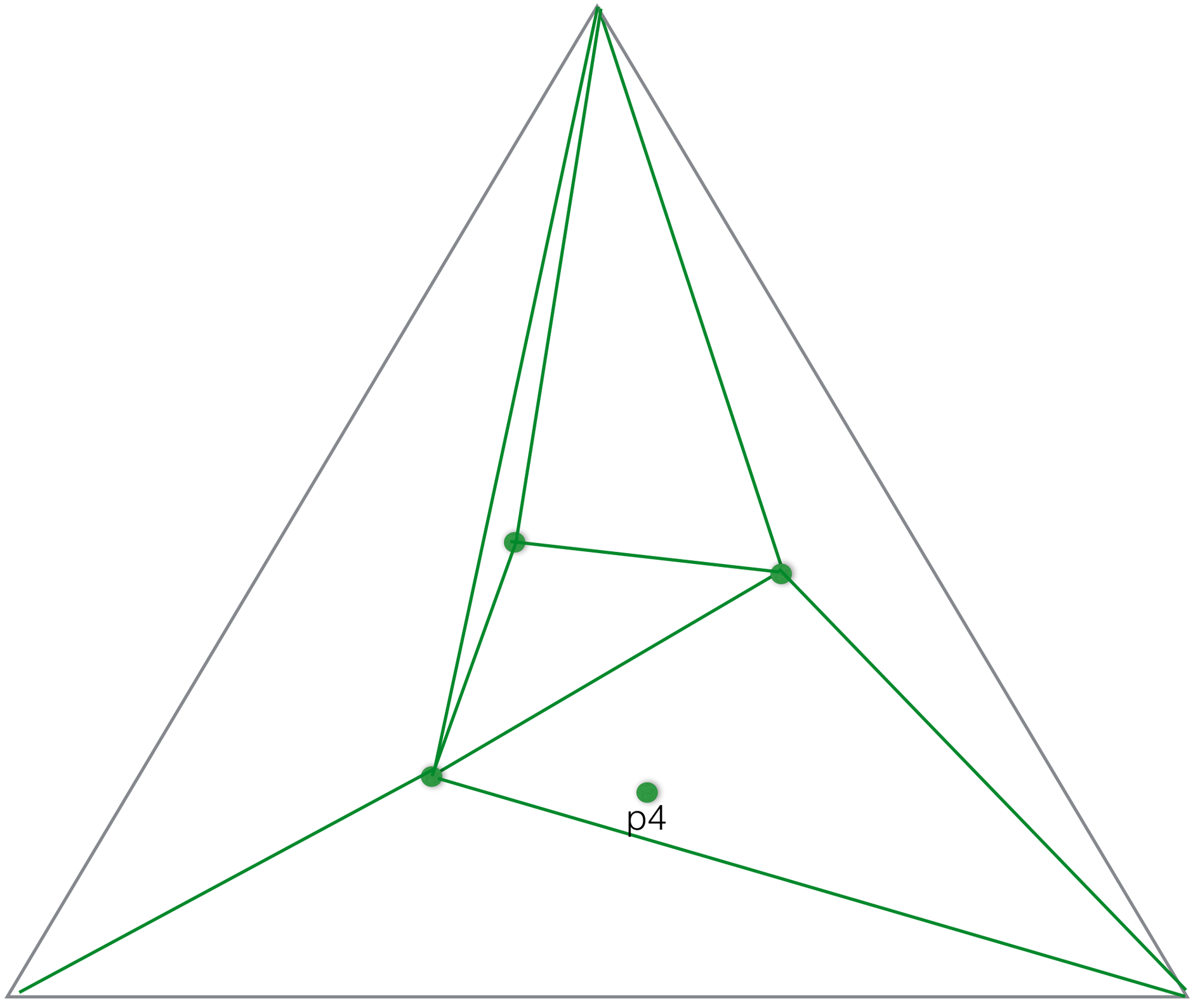

p2
p1

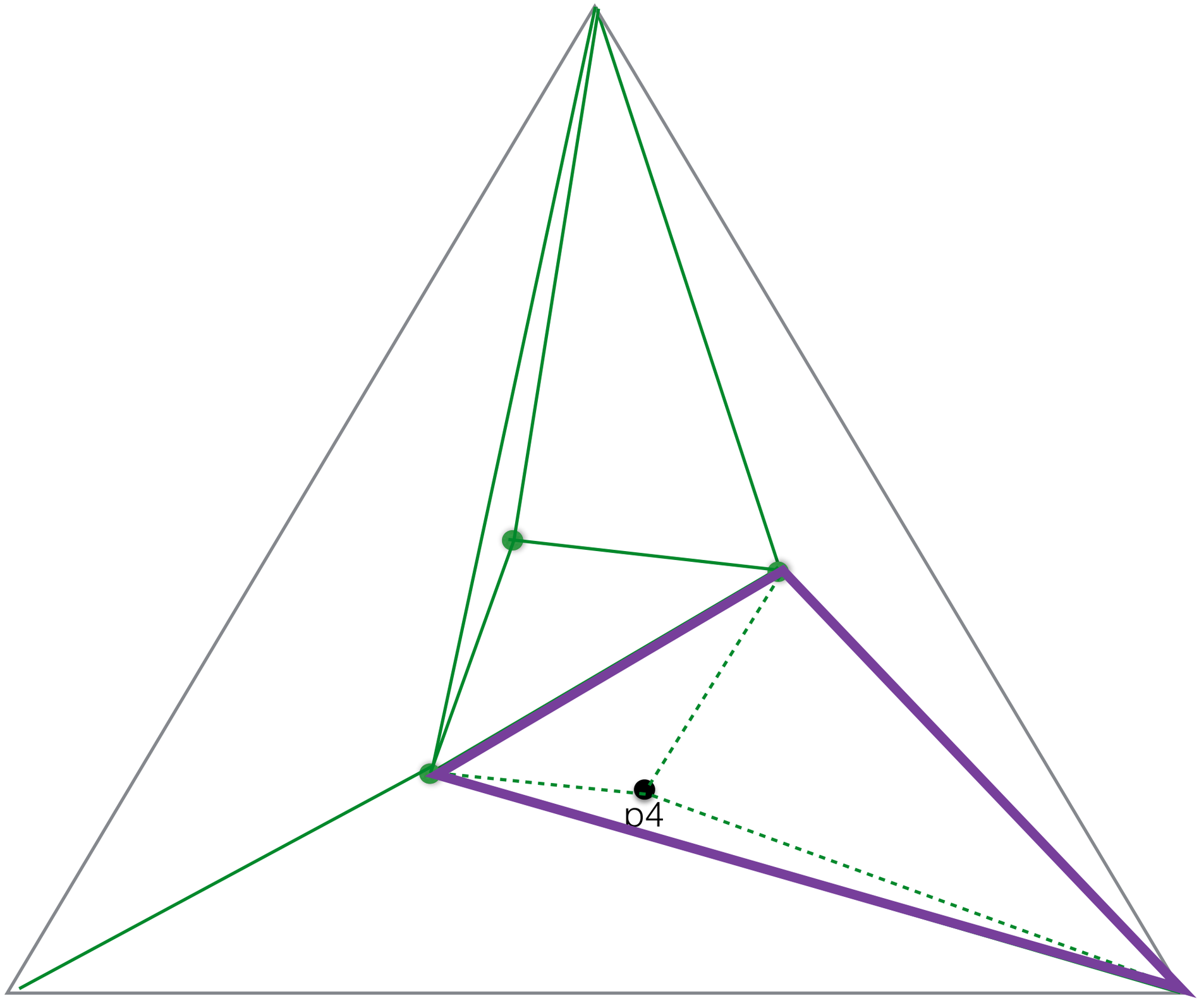

p3
p2
p1

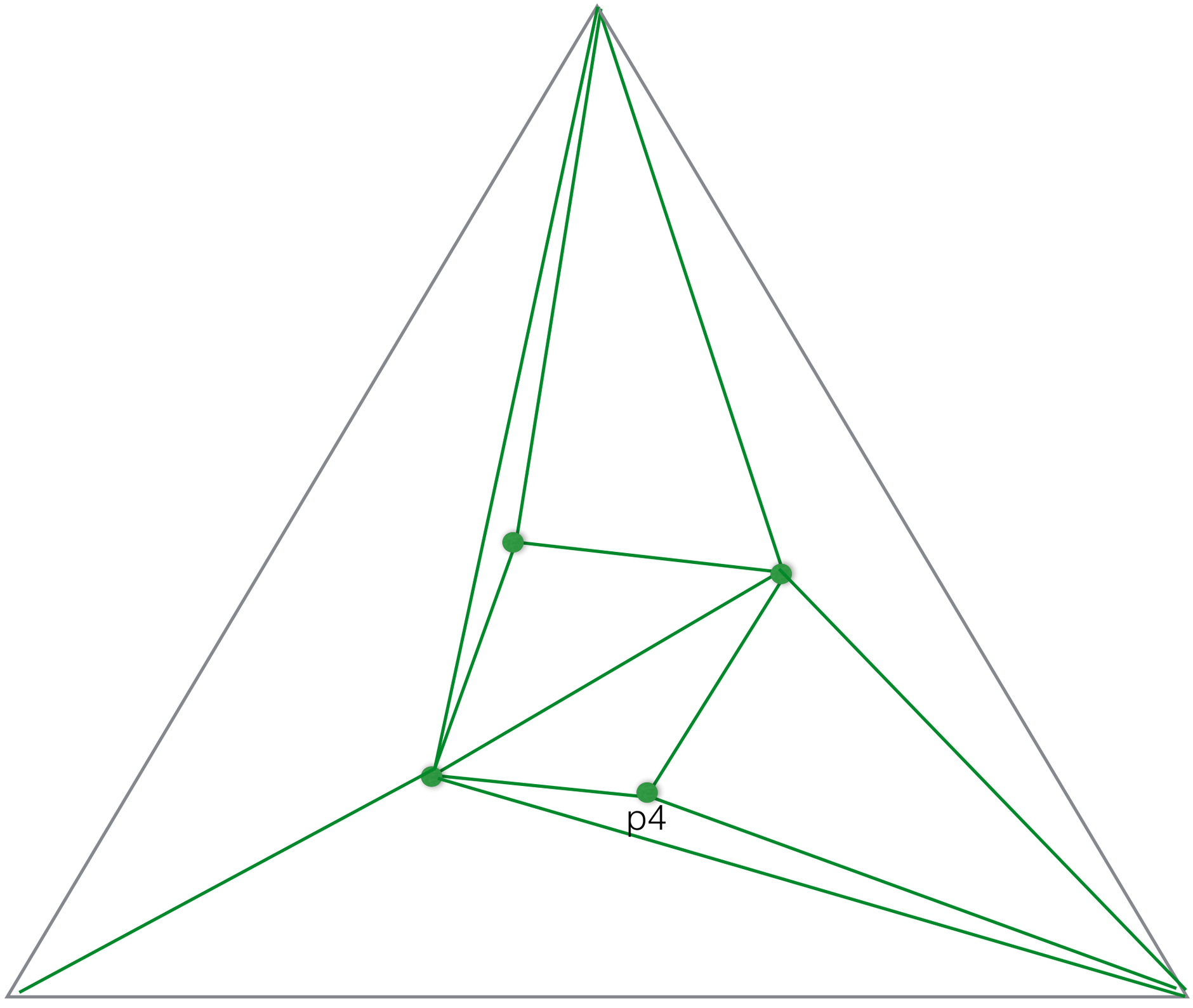

p3
p2
p1

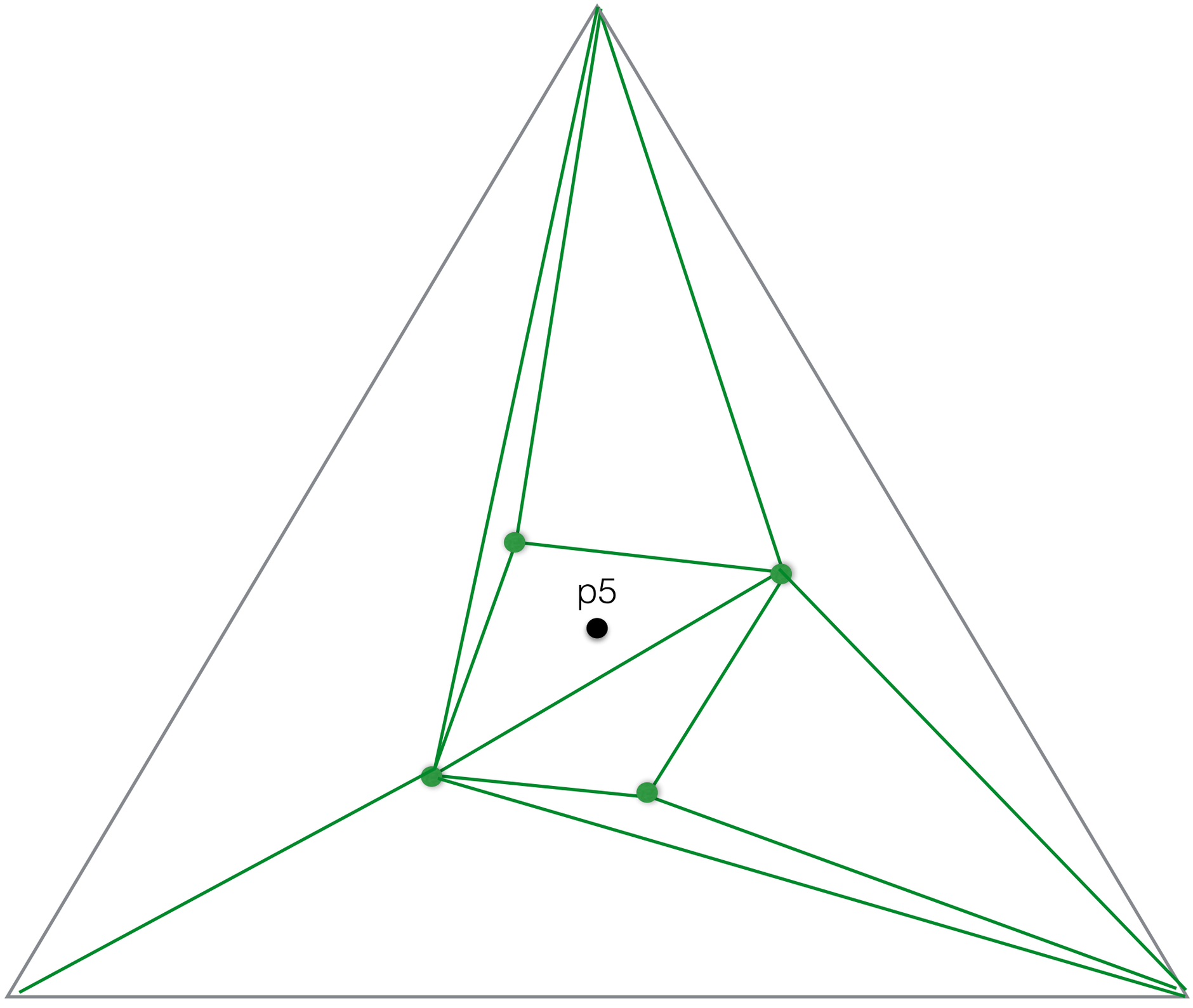

p3
p2
c
p1

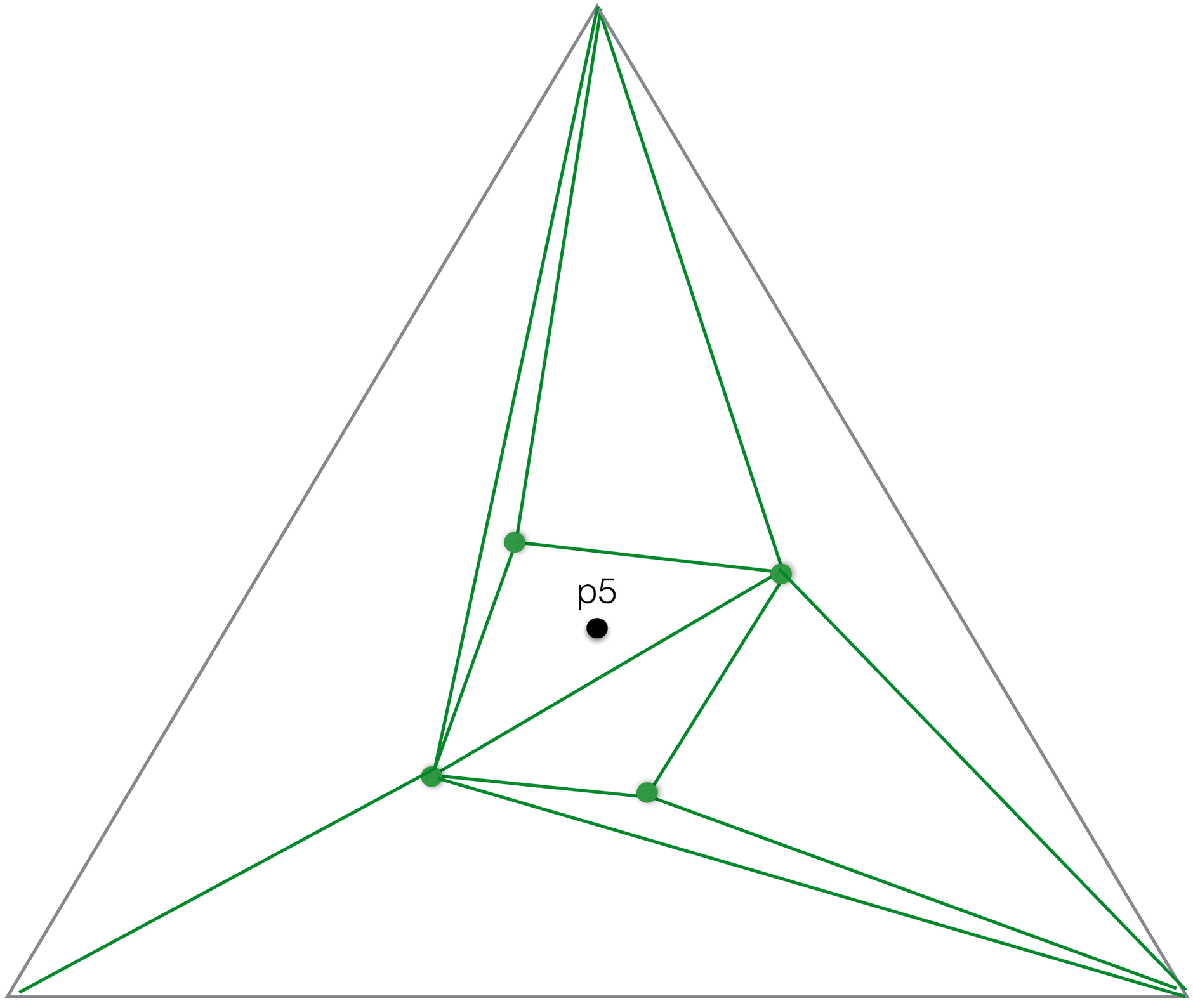

p3
p2
p1

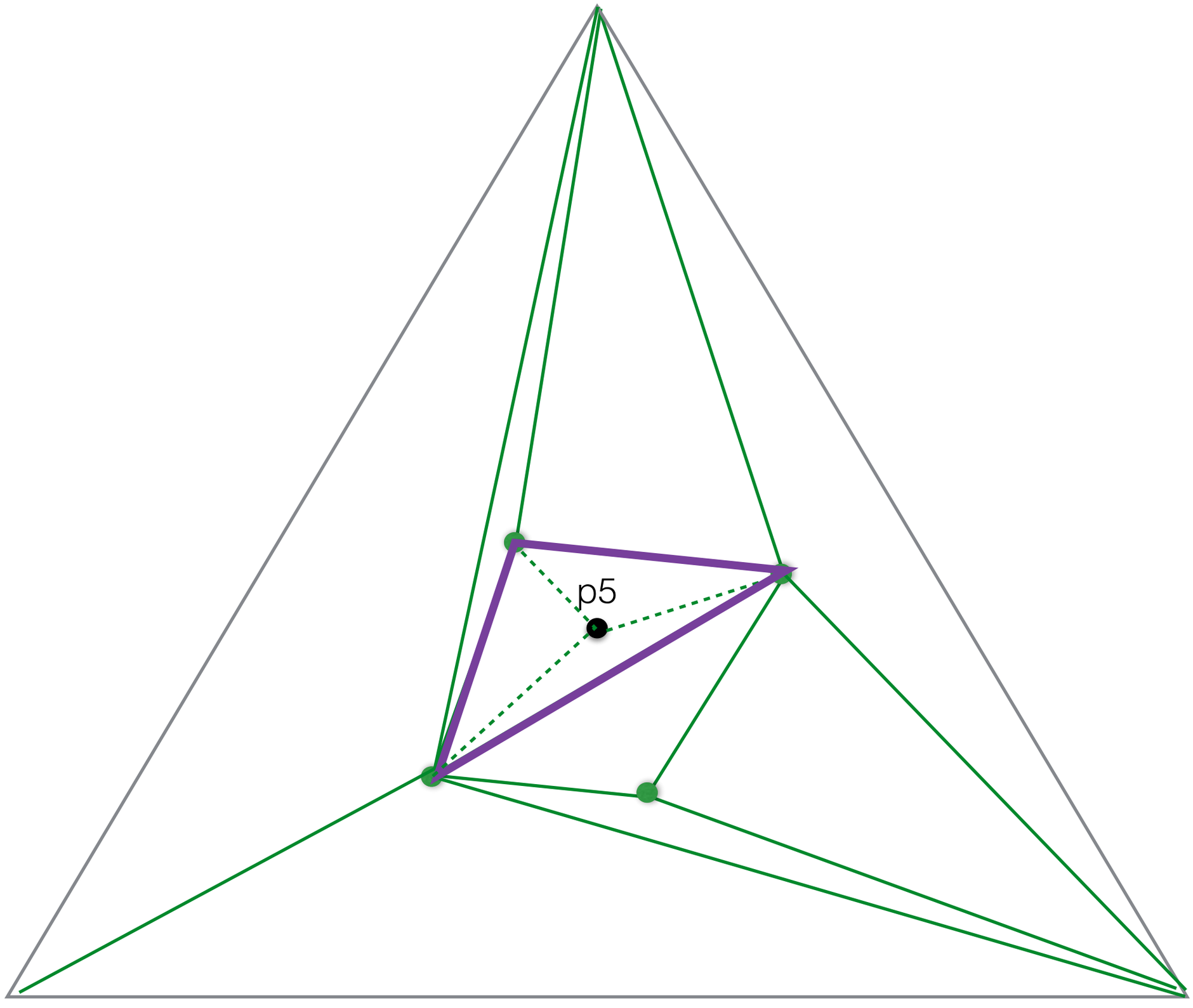

p4

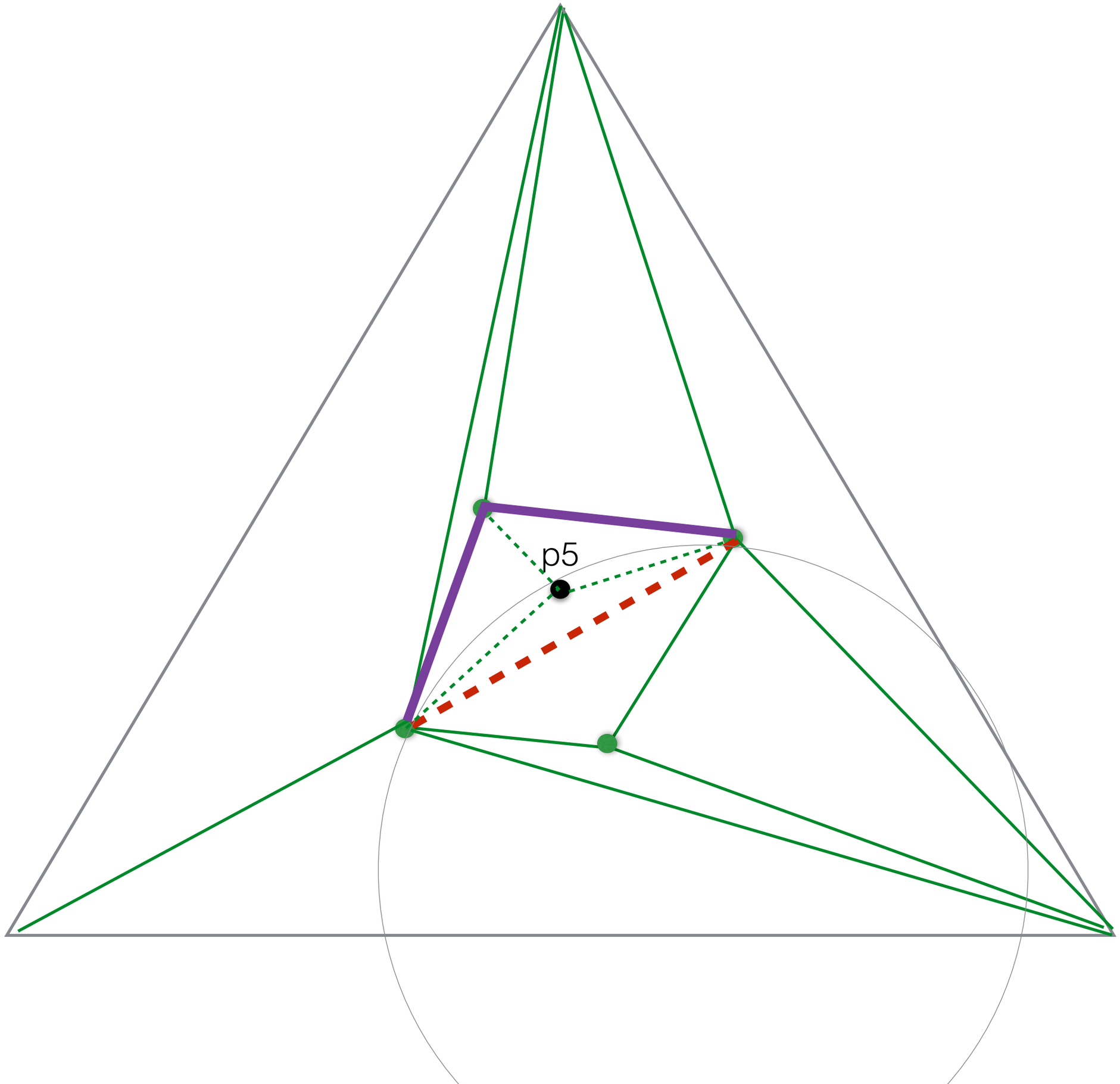

p4

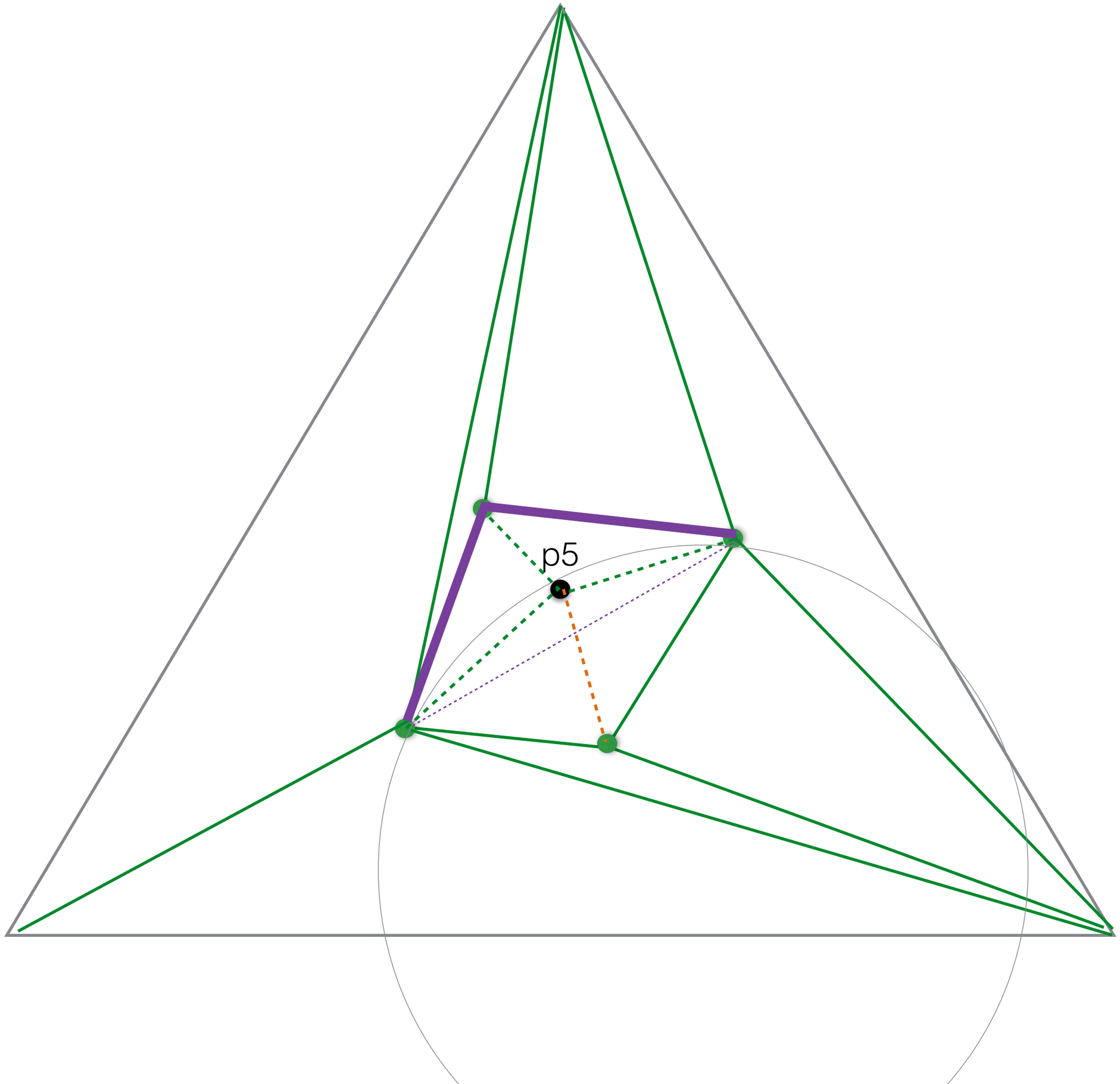

p4

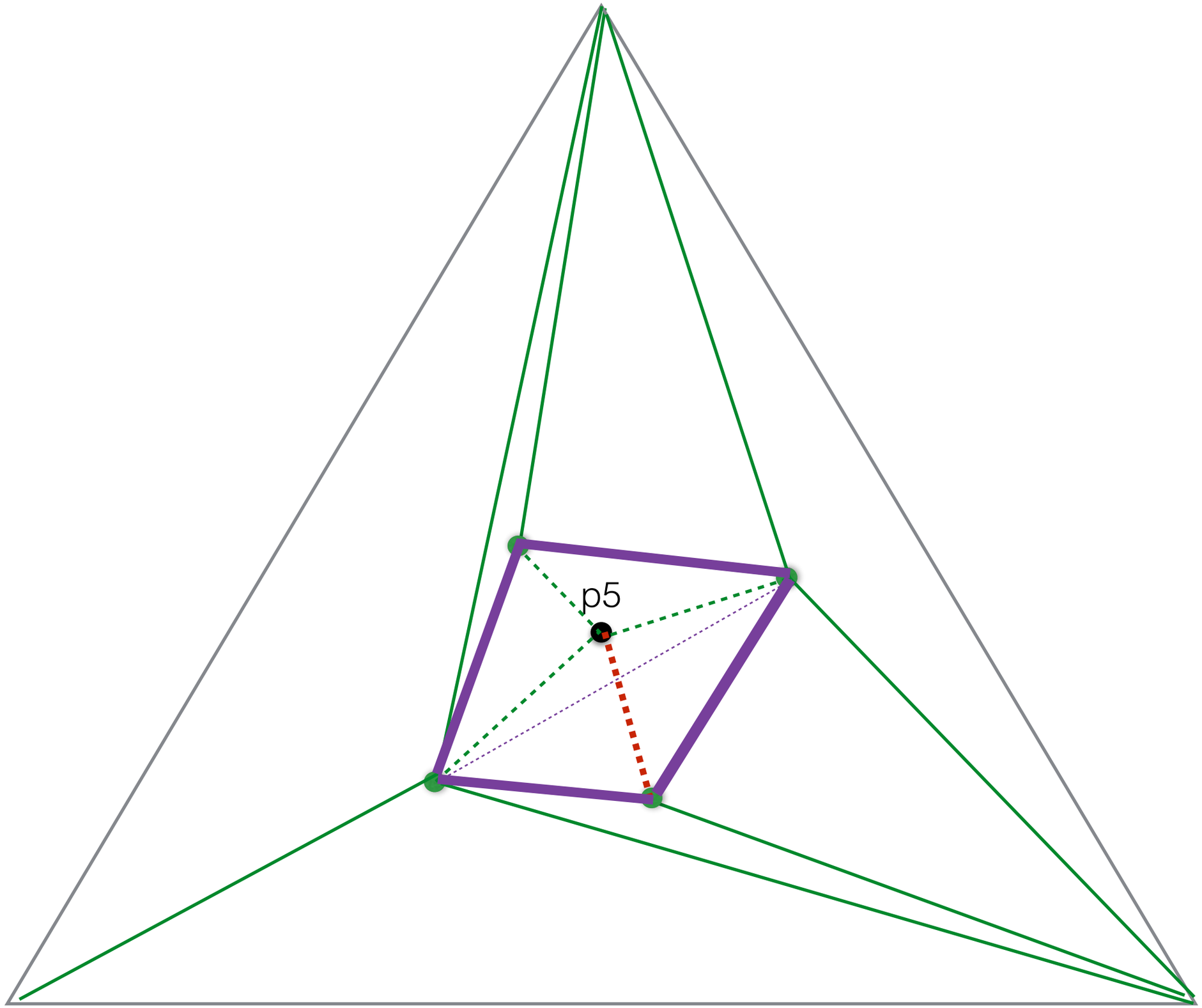

p5

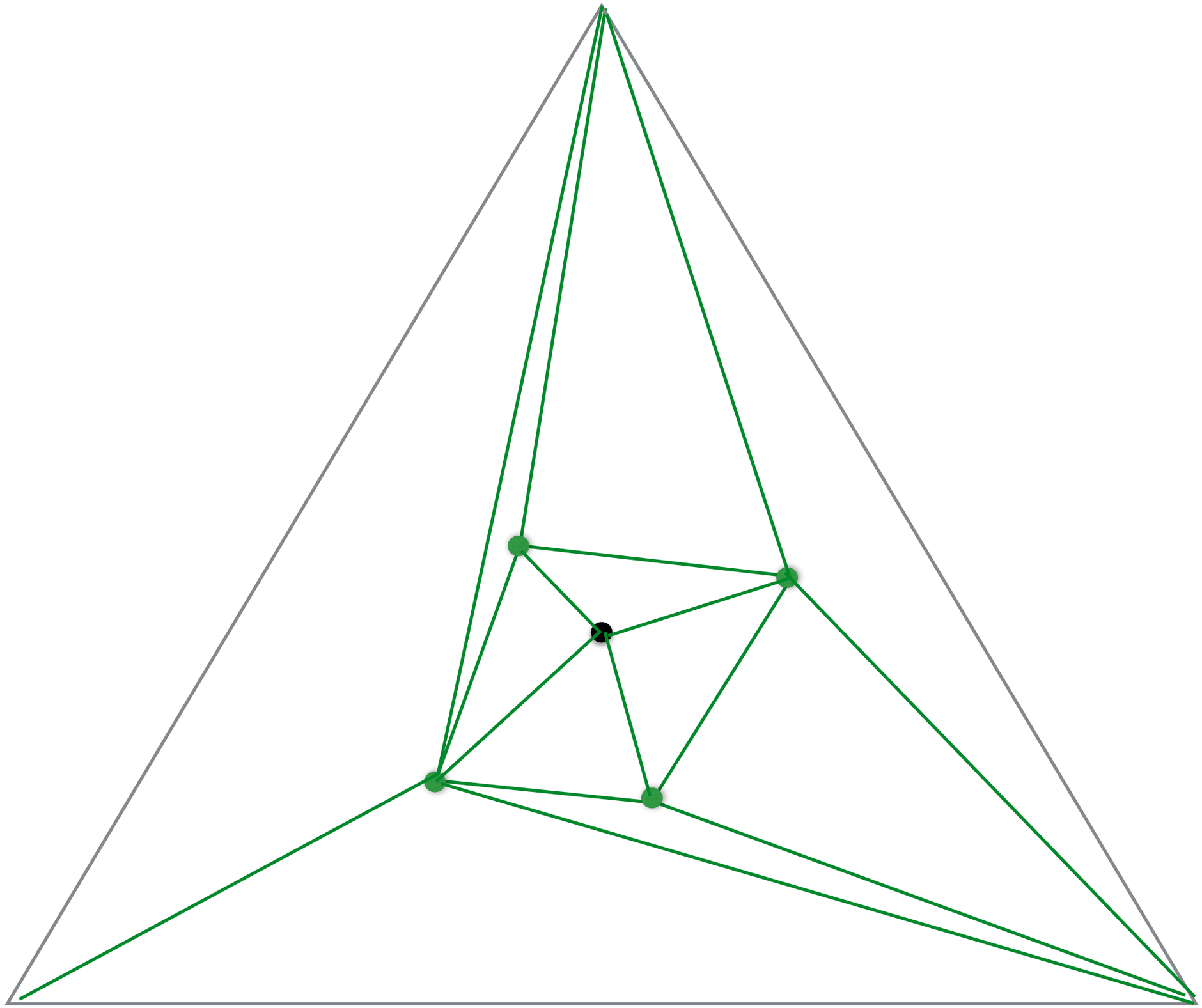

p5

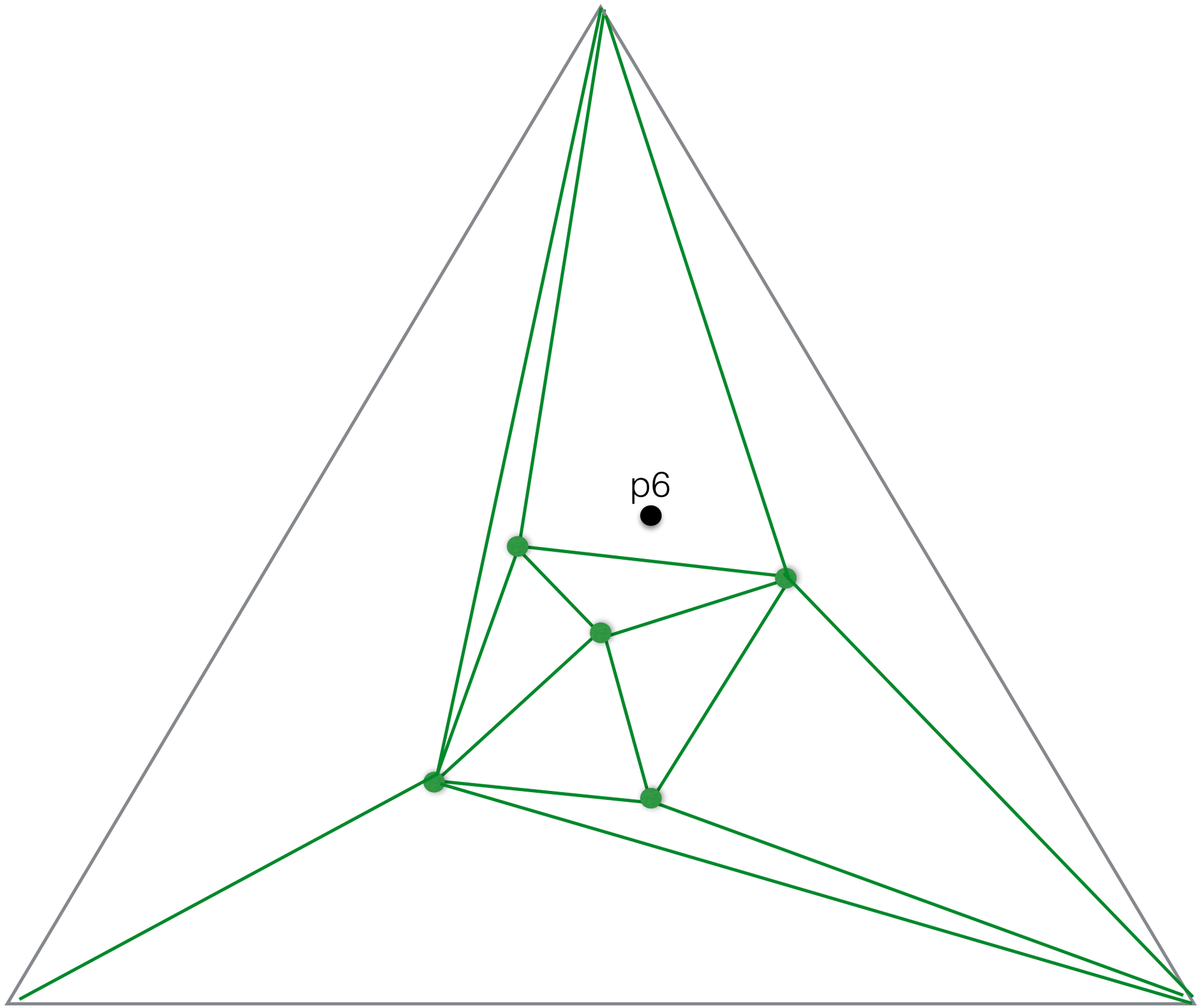

p5

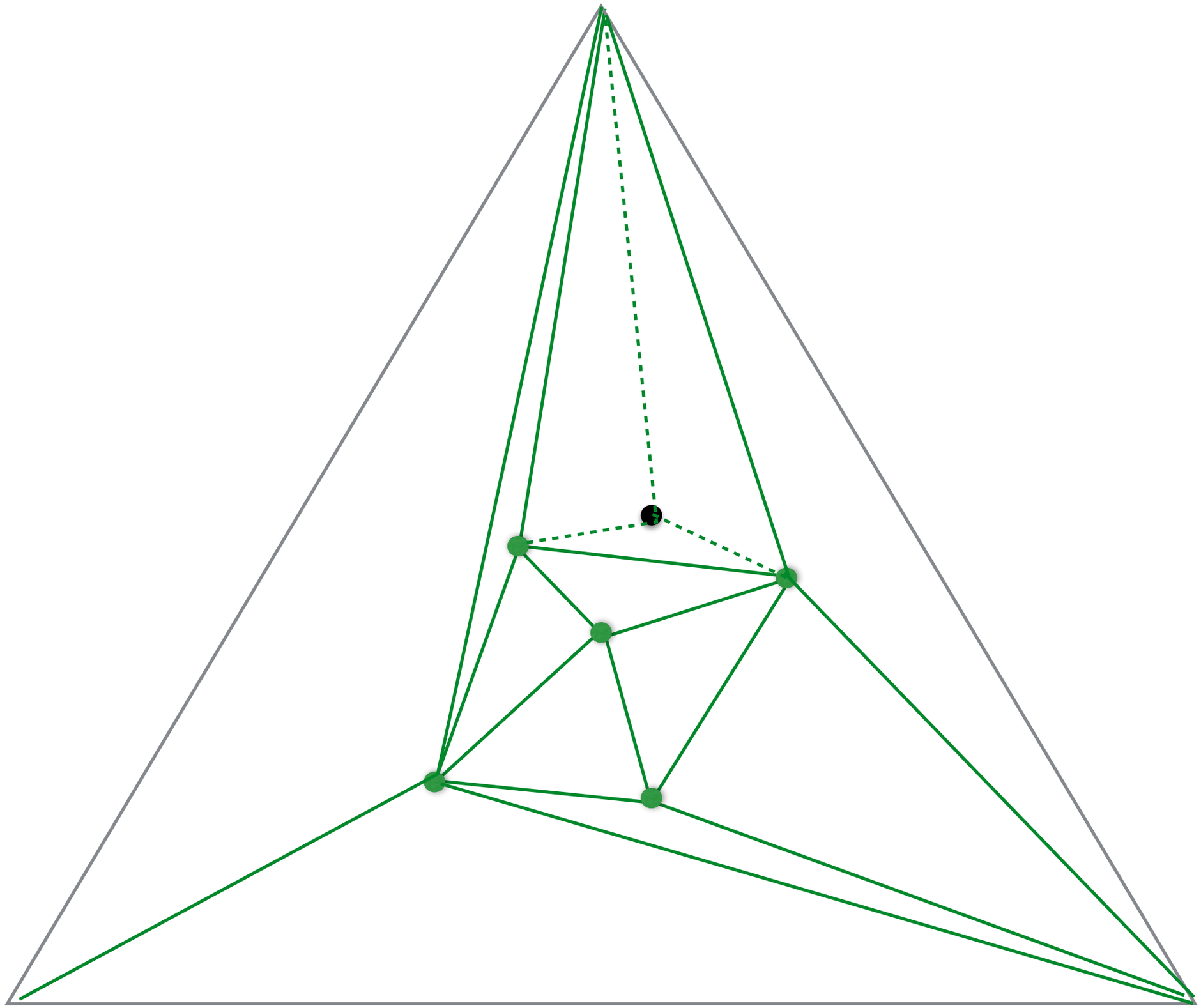

p5

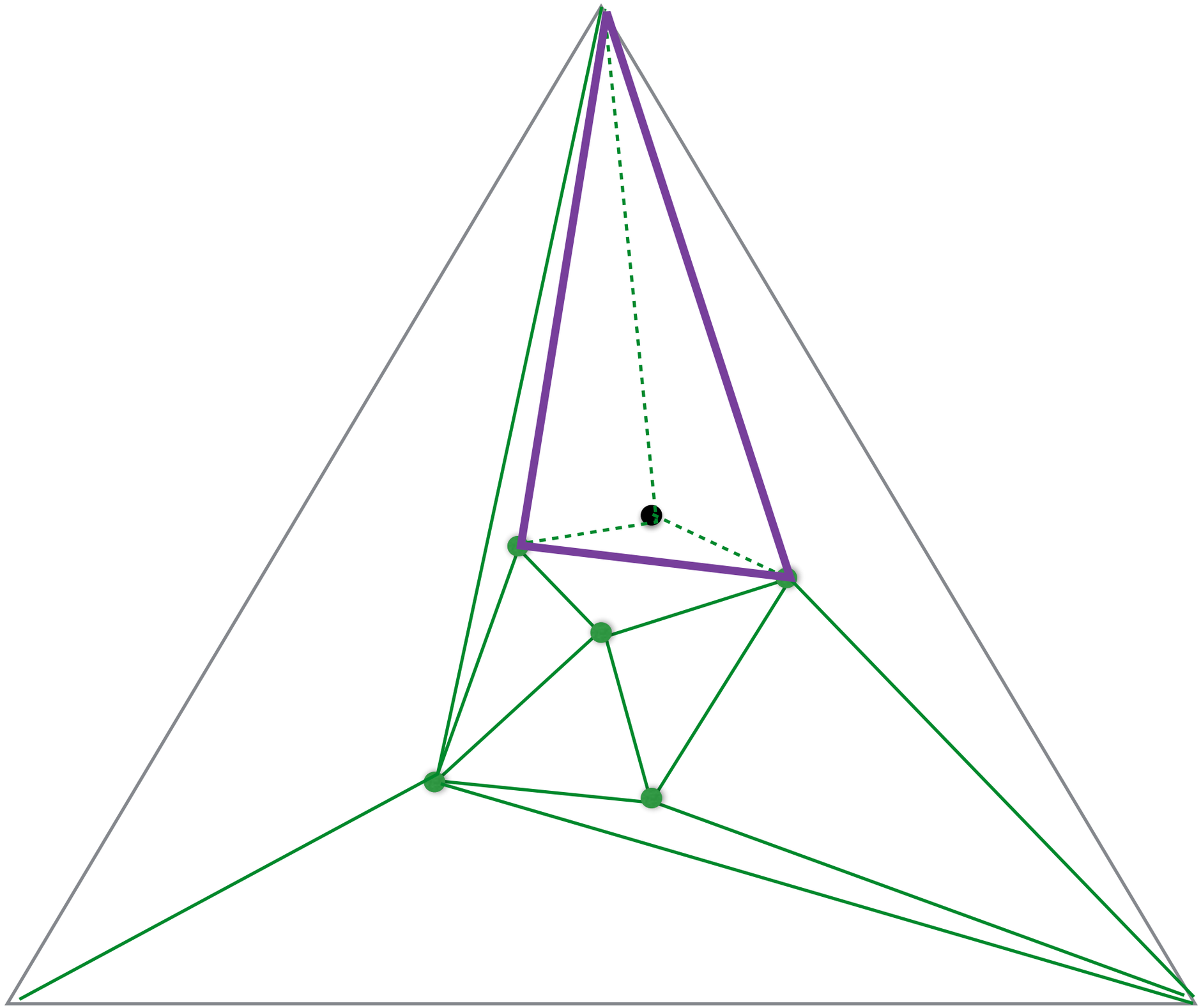

p5

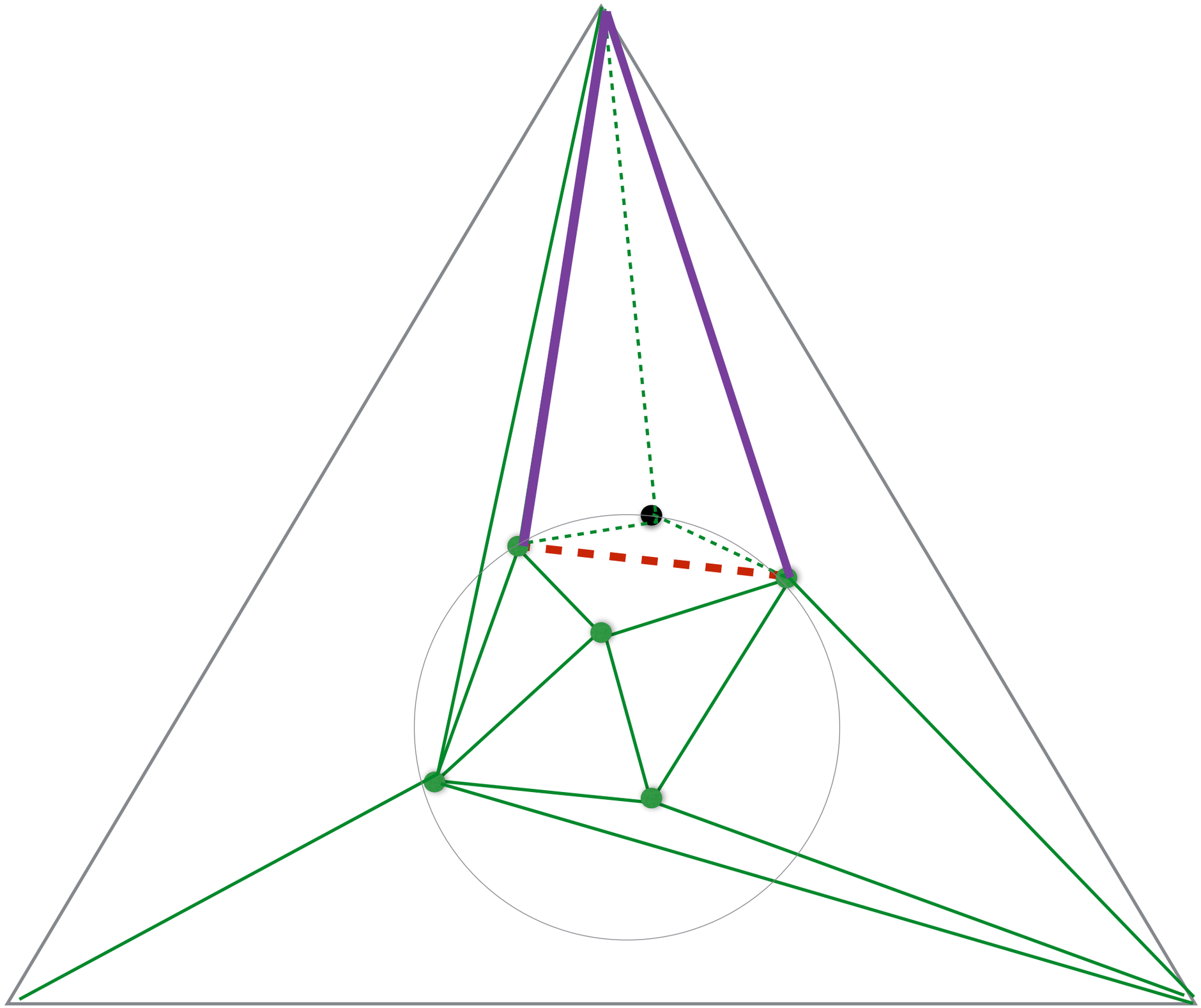

p5

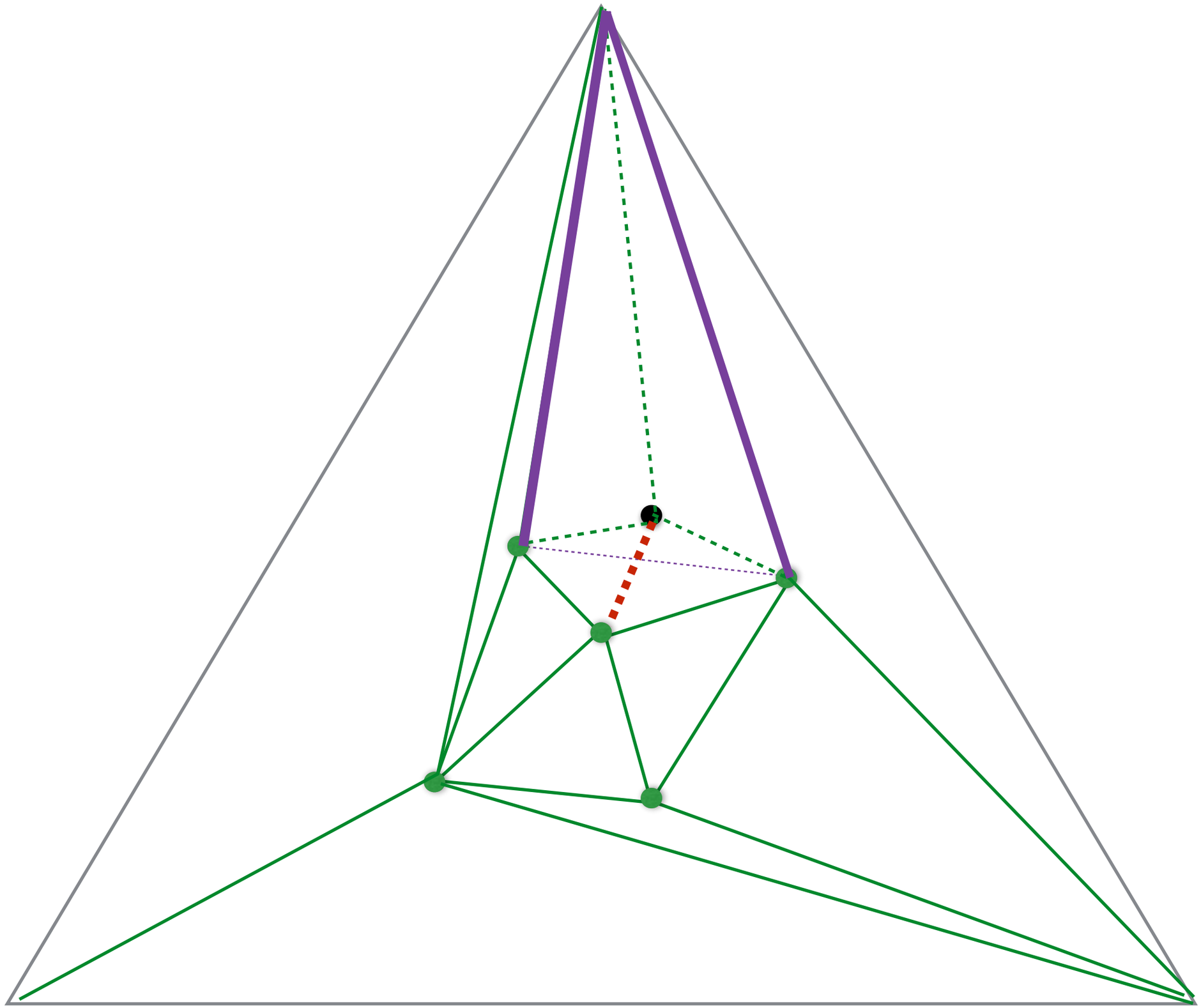


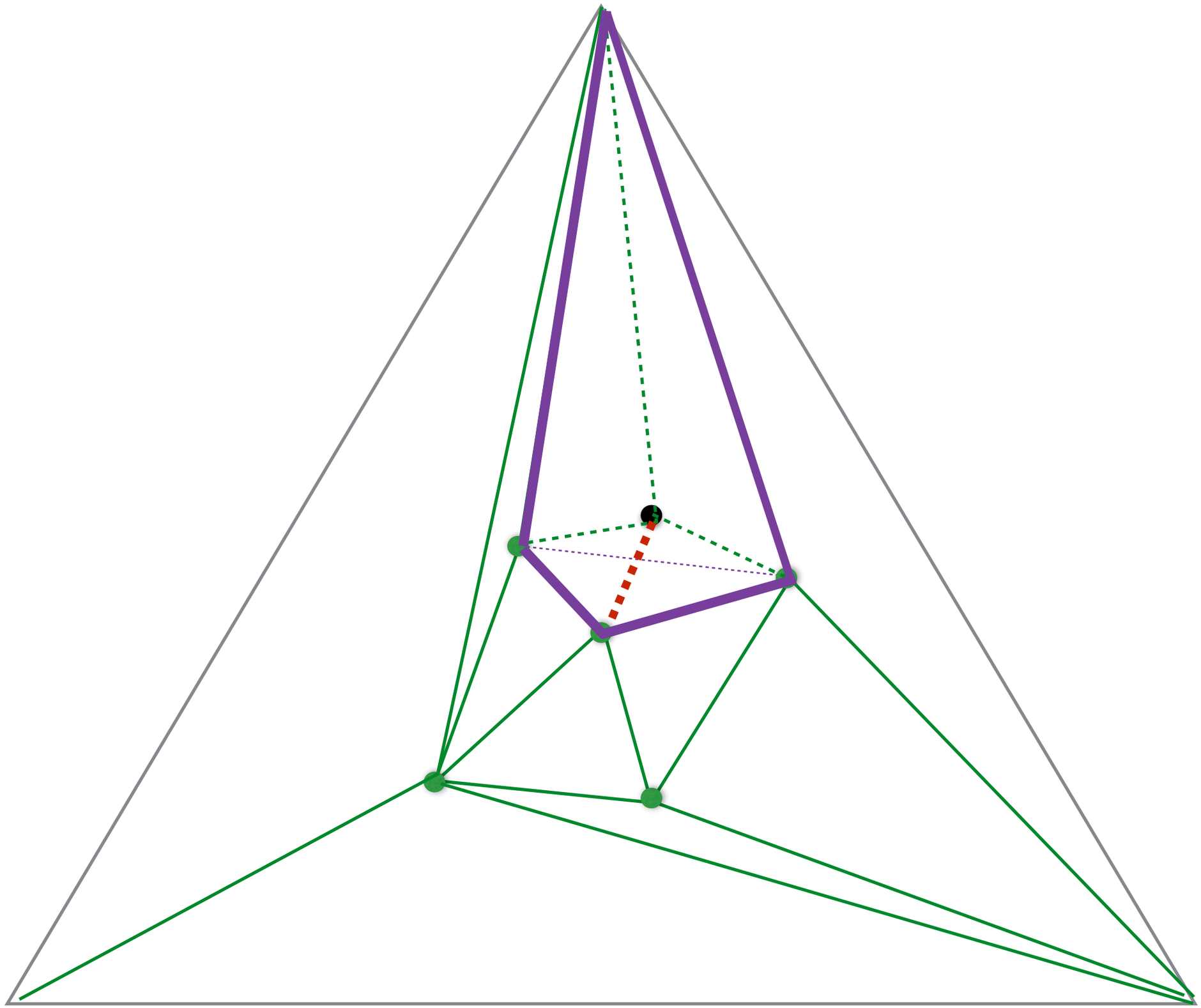

p6

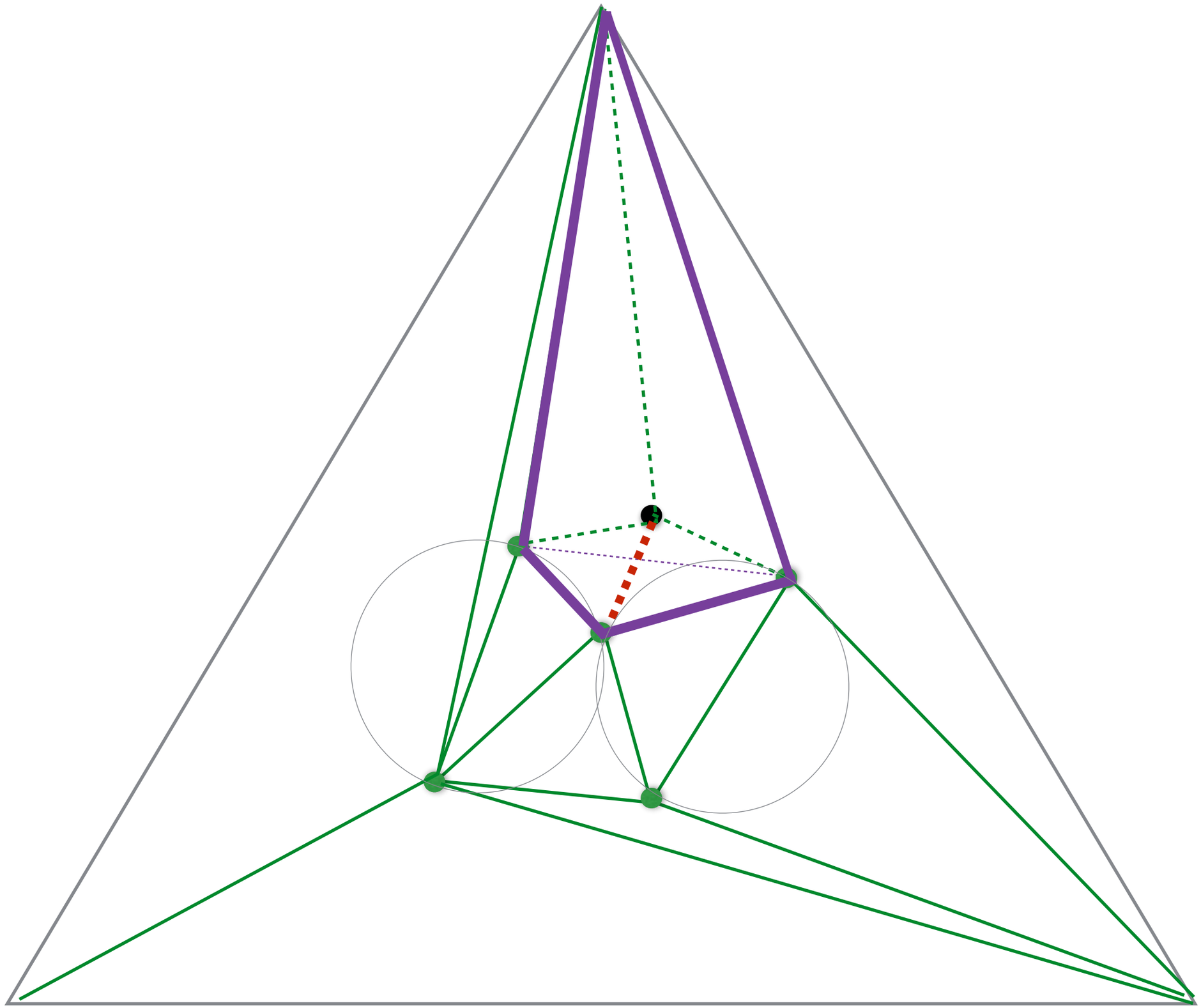


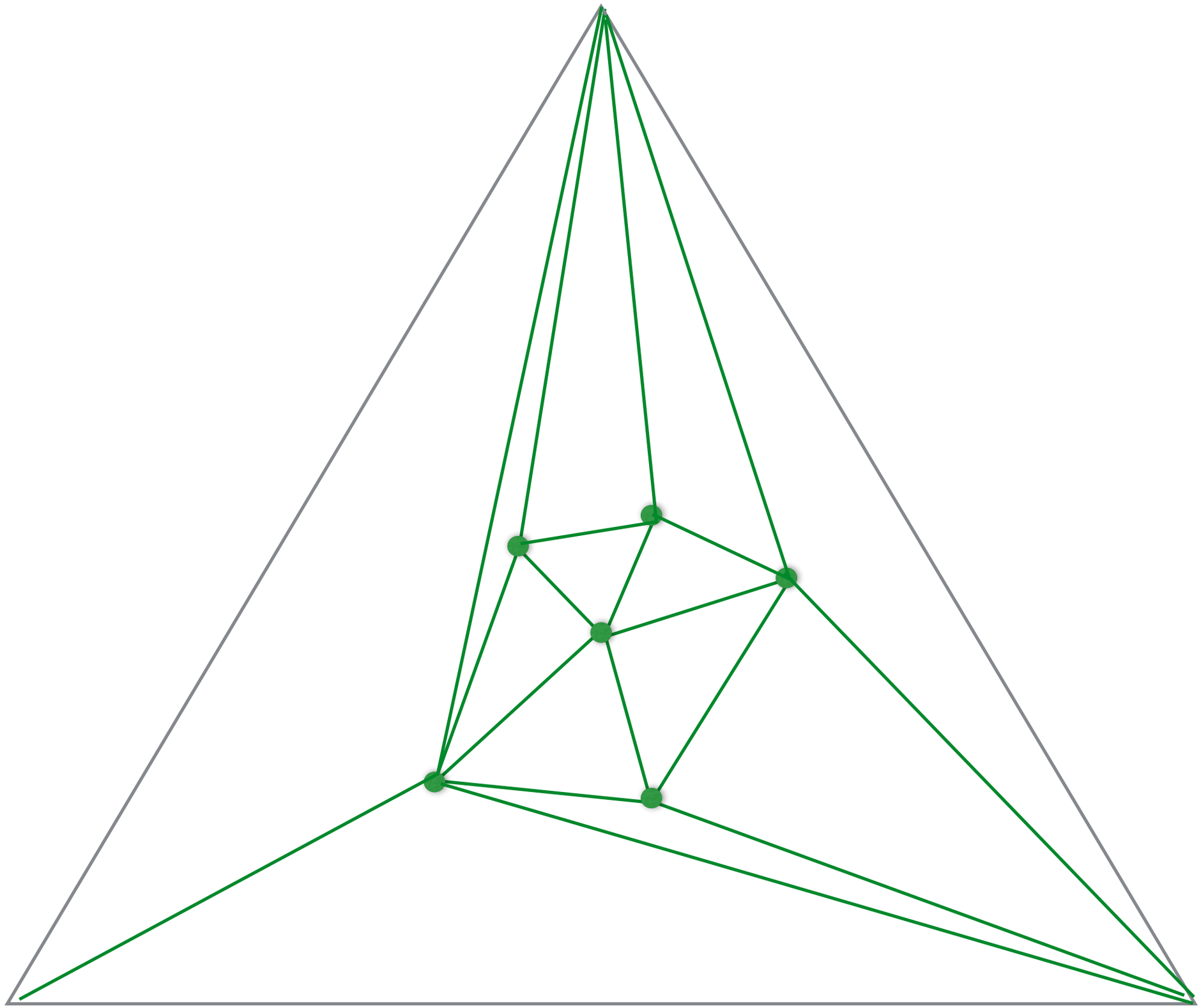


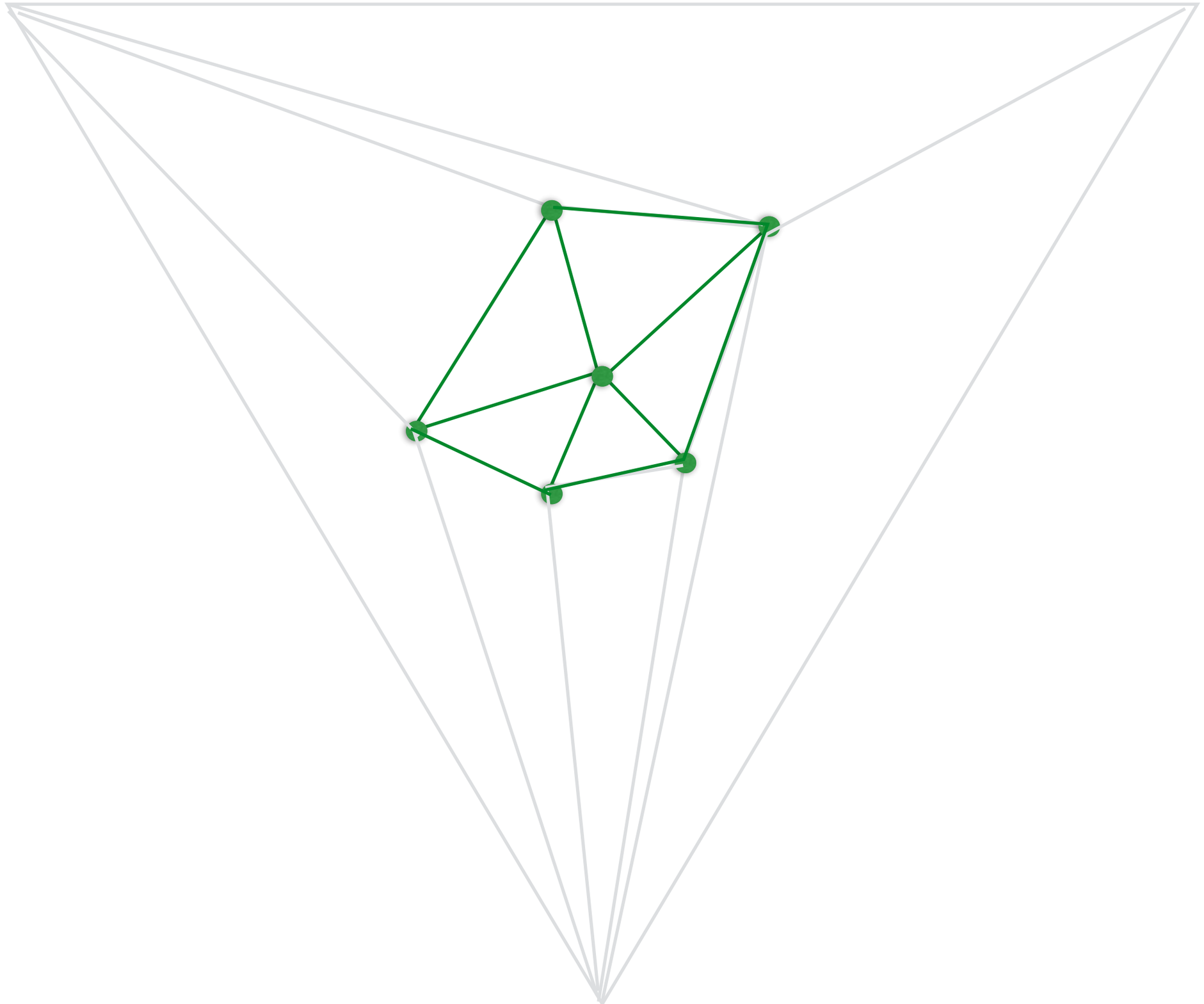












Computing DT via RIC

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

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)

Computing DT via RIC

The usual questions:

Is it correct?

Running time?

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

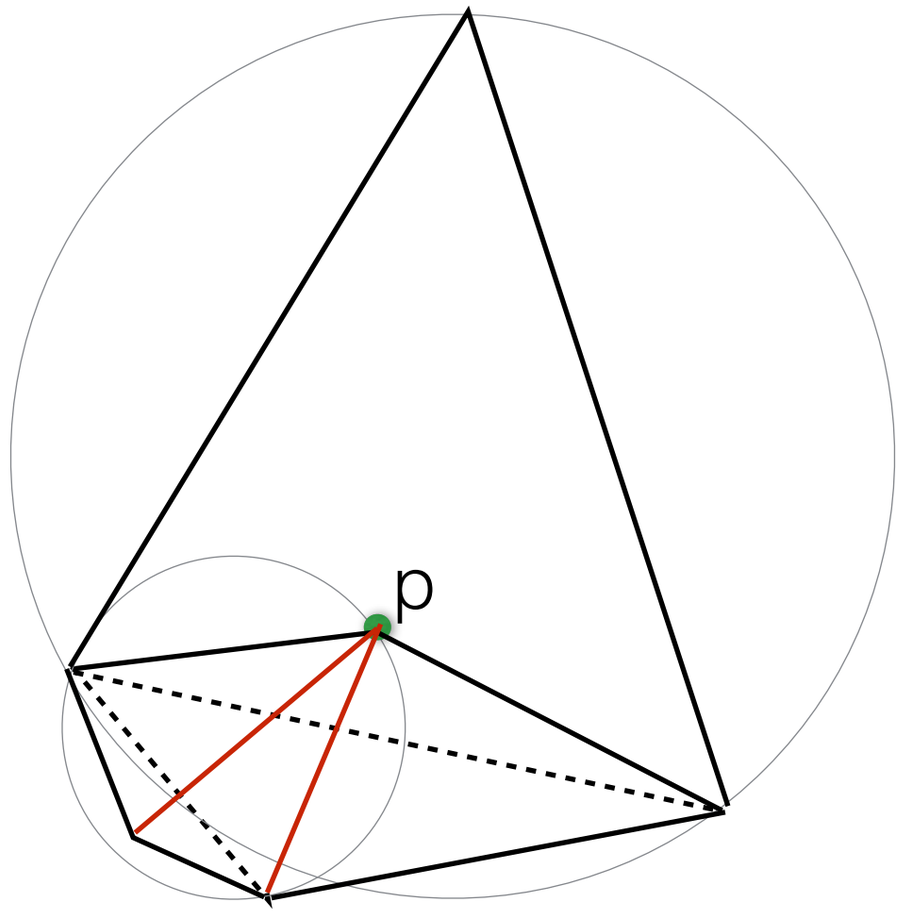
  - find triangle abc of T that contains p_i
  - splitting into 3 triangles: abp_i, bcp_i, cap_i and update T
  - LegalizeEdge(p_i, ab, T)
  - LegalizeEdge(p_i, bc, T)
  - LegalizeEdge(p_i, ca, T)
- Discard the initial triangle T and its edges
- return T

locate triangle that contains p

update T: O(1)

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

RIC Running time



- Inserting p may trigger many edge flips

- Each edge flip creates an edge incident to p.

- Total time spent in 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, but it's O(1) on the average

RIC at high level

RIC (DT Randomized Incremental Construction)

- Let (p_1, p_2, p_3, …p_n) be a random permutation of P
- Initialize T as a large triangle that contains P.
- For next point p_i in P do:
  - point location: find triangle abc of T that contains p_i

    //this can be done in O(lg n) per point
  - insert p in abc and perform edge flips around p

    //on the average this takes O(1) per point

Note: the average degree of a vertex in DT is 6.

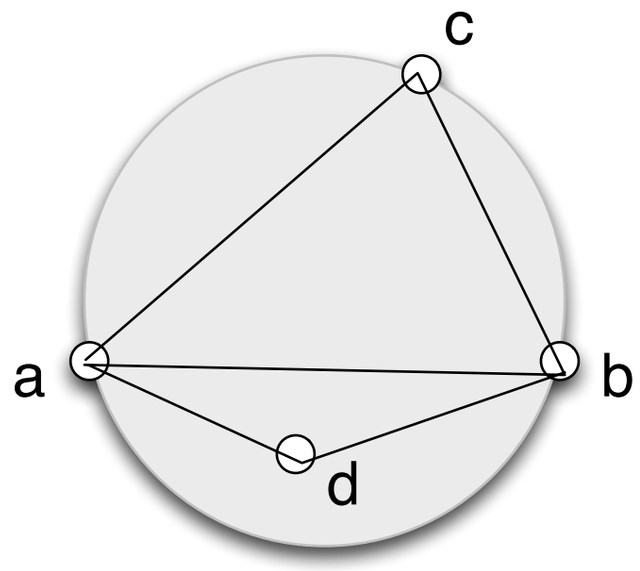
Running time: O(n lg n) expected

DT and the Convex Hull

In-Circle test

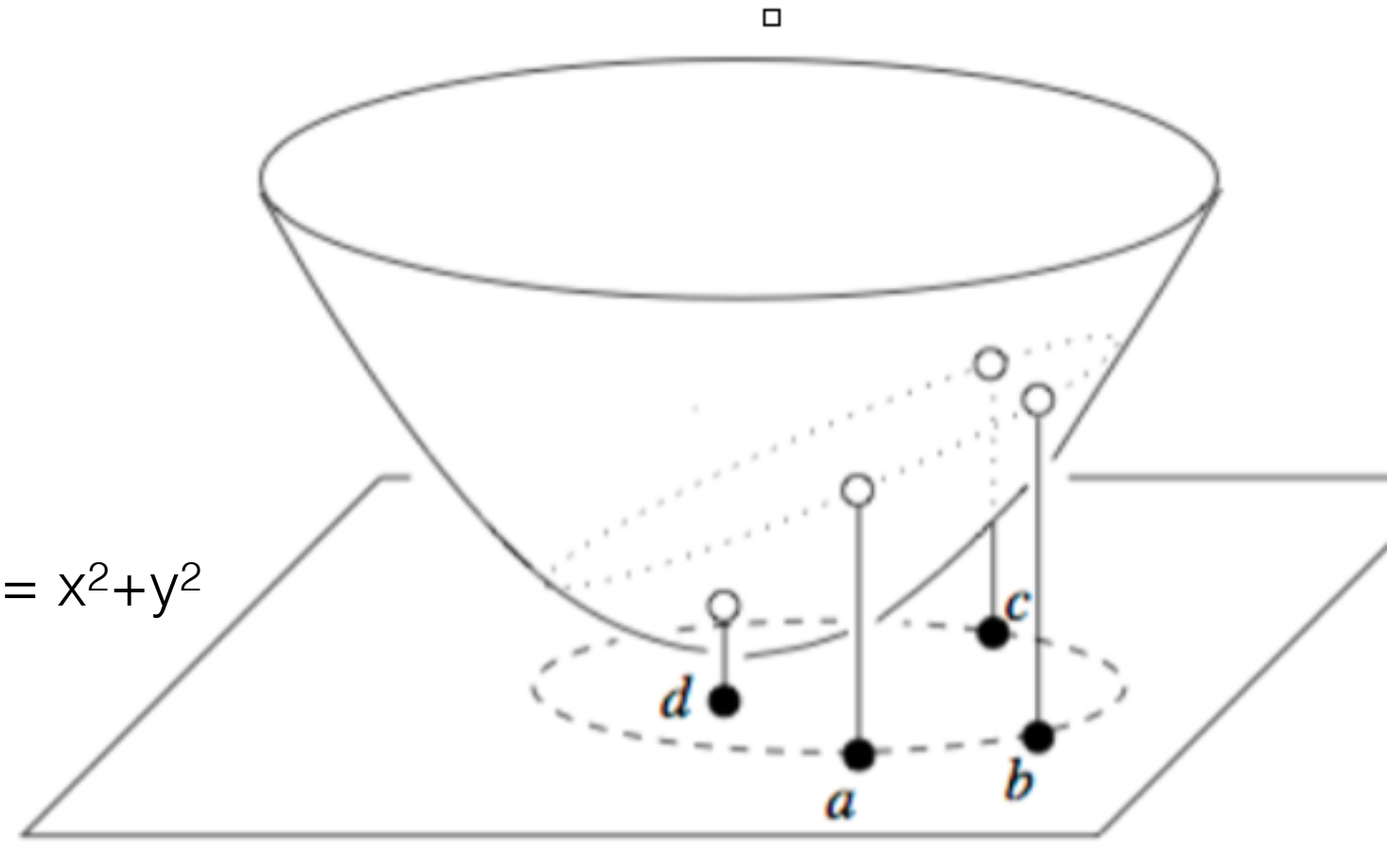
d is inside circle(a,b,c) if

$$\det \begin{vmatrix} 1 & a_x & a_y & a_x^2+a_y^2 \\ 1 & b_x & b_y & b_x^2+b_y^2 \\ 1 & c_x & c_y & c_x^2+c_y^2 \\ 1 & d_x & d_y & d_x^2+d_y^2 \end{vmatrix} < 0$$



DT and Lifted Paraboloid

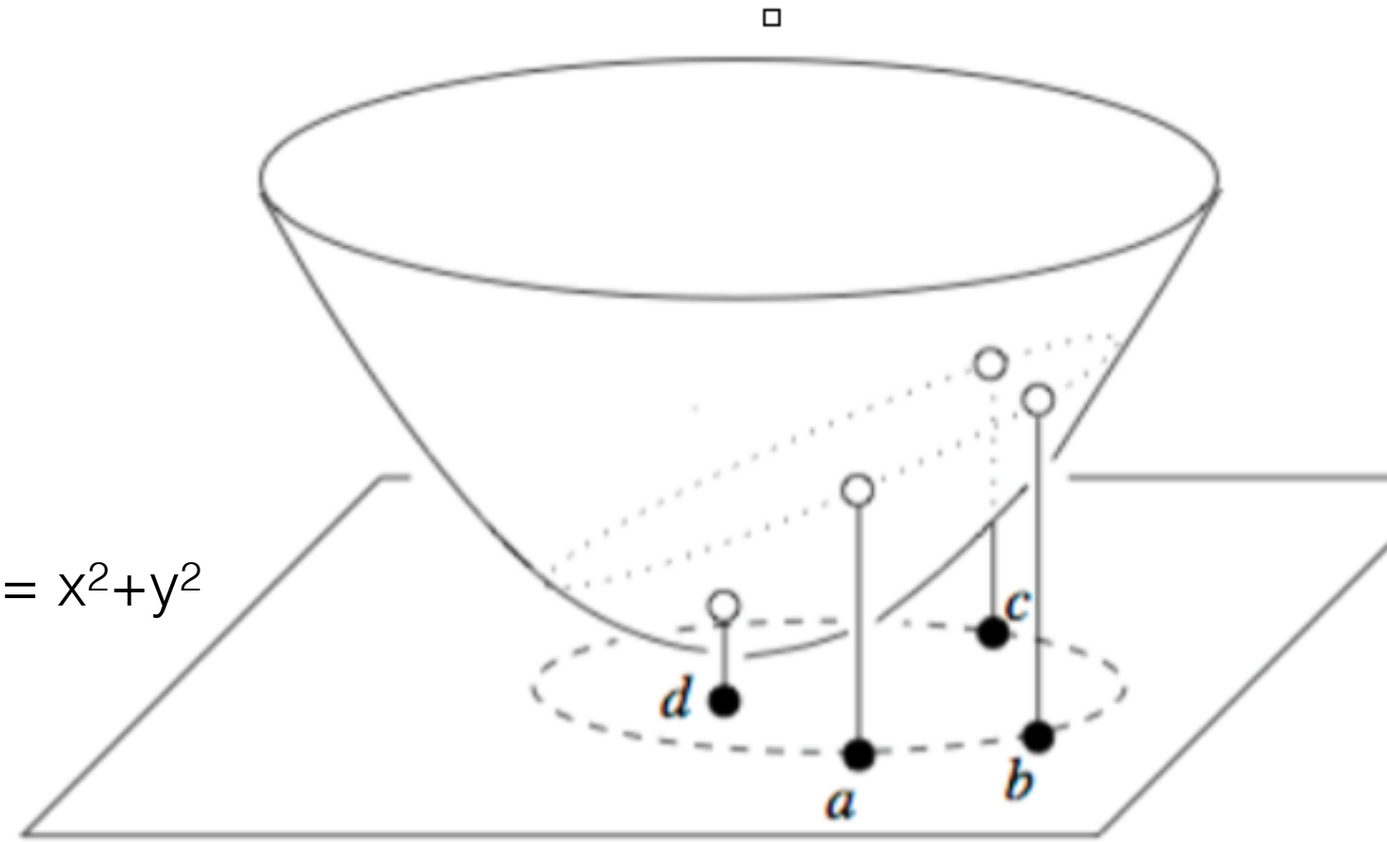
- Let a,b,c,d be points in the plane
- Let a', b', c', d' be their vertical projections onto the paraboloid $z = x^2+y^2$



DT and Lifted Paraboloid

- Let a,b,c,d be points in the plane

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



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

- Proof: It turns out that the expression for inside(d, circle(a,b,c)) is the same as for below(d', plane(a'b'c'))

DT and Lifted Paraboloid

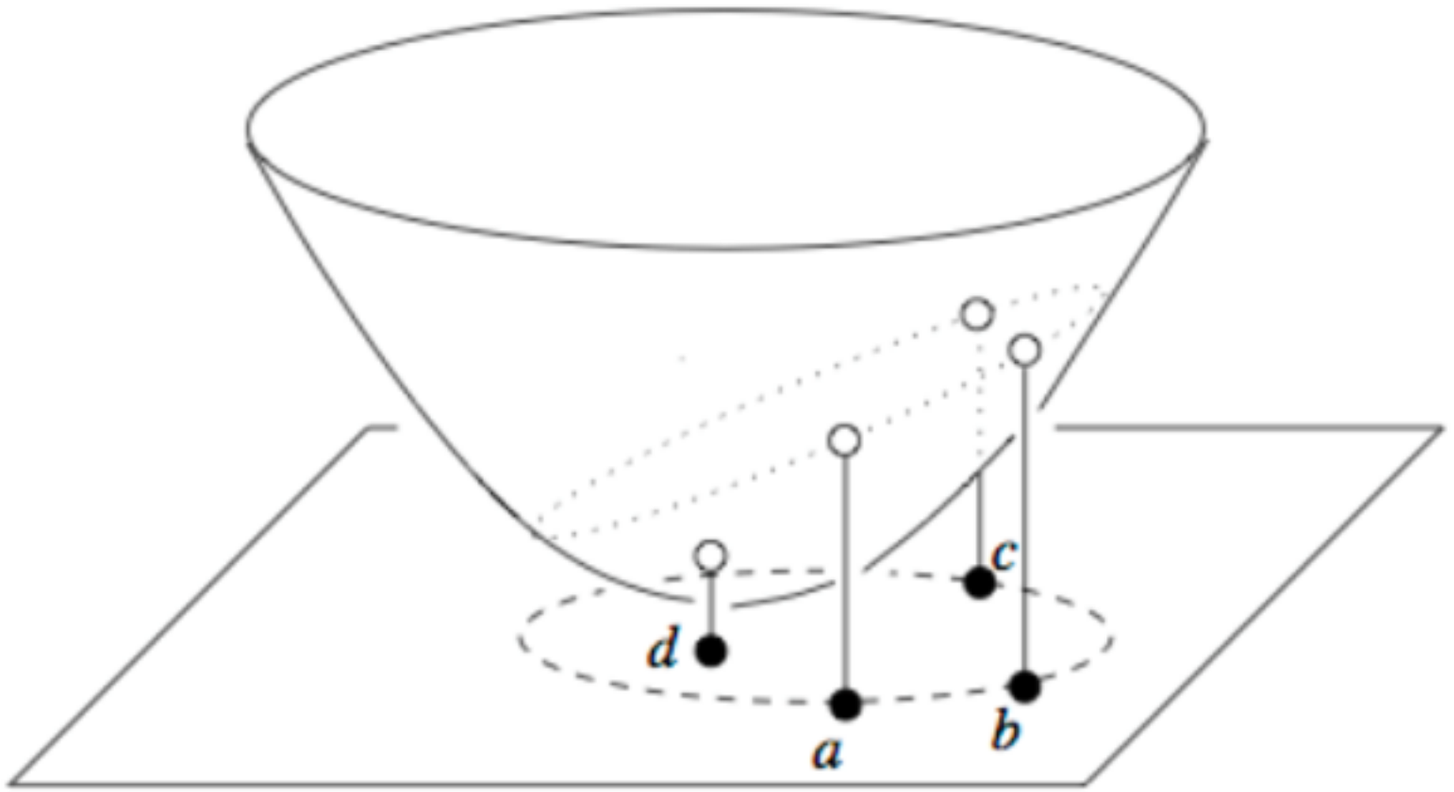
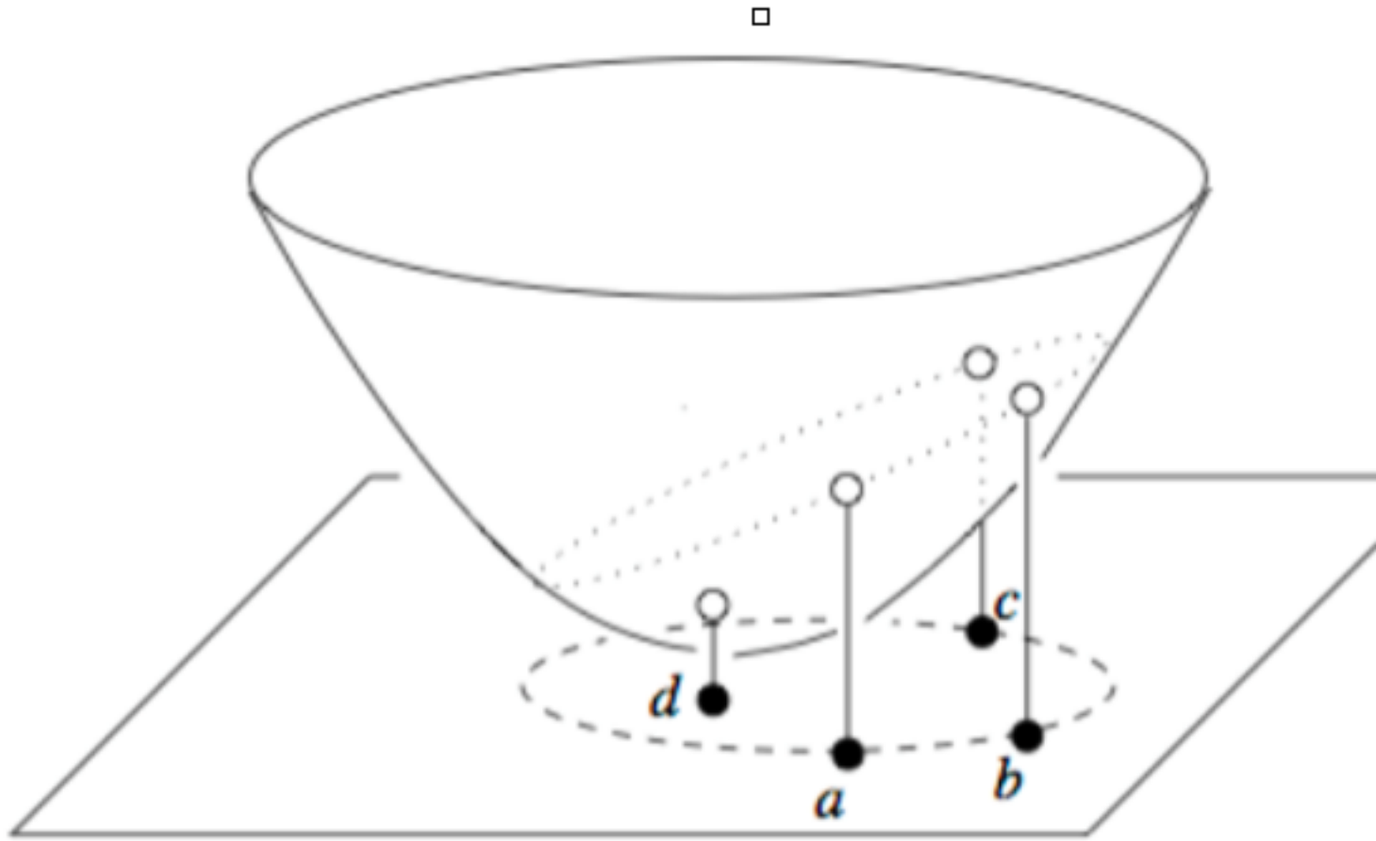


Figure 1.11. Points a, b, c lie on the dashed circle in the x_1x_2-plane and d lies inside that circle. The dotted curve is the intersection of the paraboloid with the plane that passes through $\hat{a}, \hat{b}, \hat{c}$. It is an ellipse whose projection is the dashed circle.

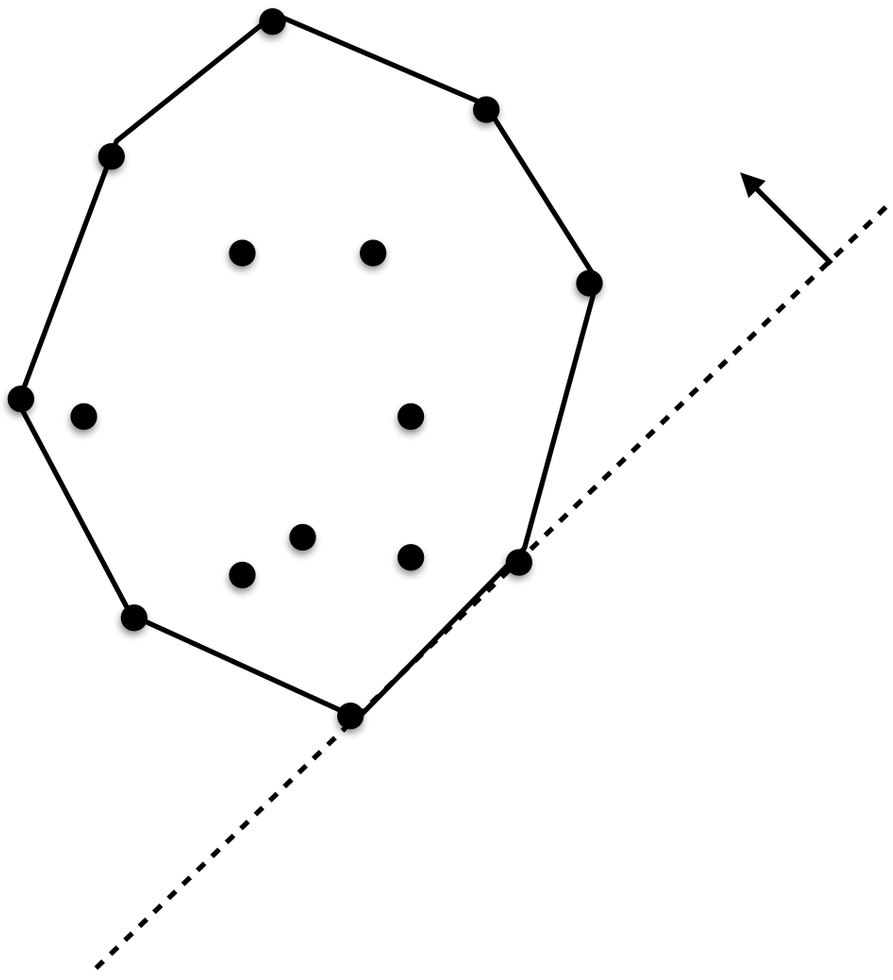
'igure from the book *Geometry and Topology for Mesh Generation* by Edelsbrunner.



- Edge flip interpretation
  - Edge ac illegal: d is inside C(abc); d’ is below plane(a’b’c’)
  - Flip ac to bd
  - Edge bd legal: a outside C(bcd) ; a’ above plane(b’c’d’)

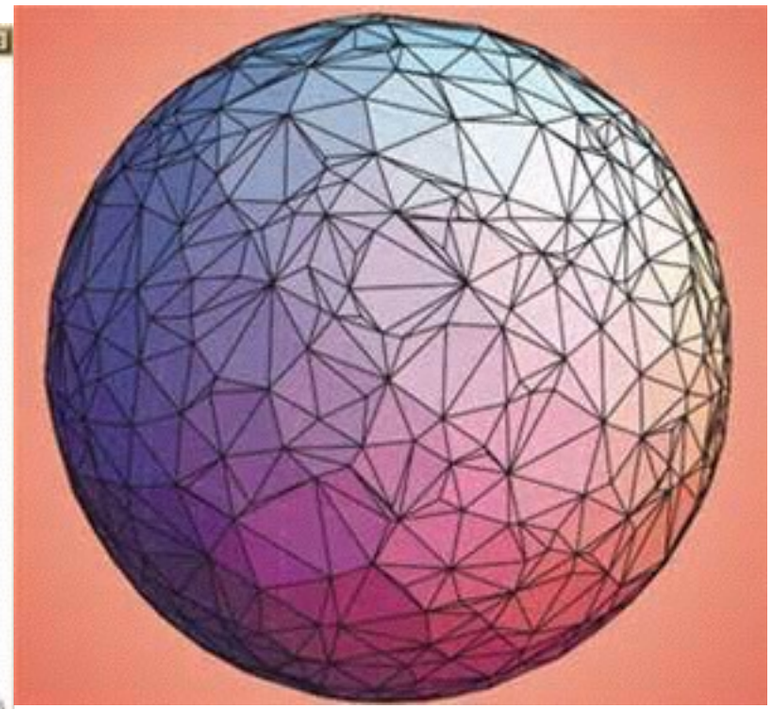
DT and Convex Hull

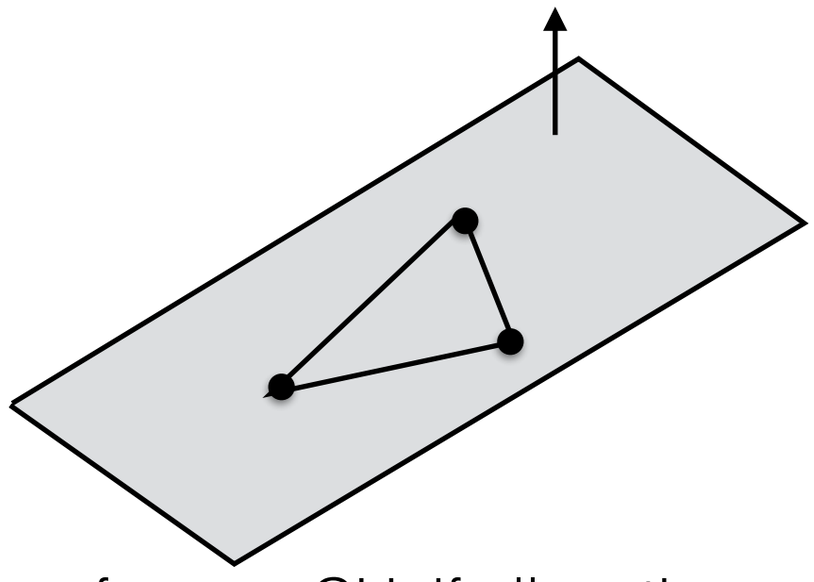
in 2D



edge on CH: if all vertices on one side

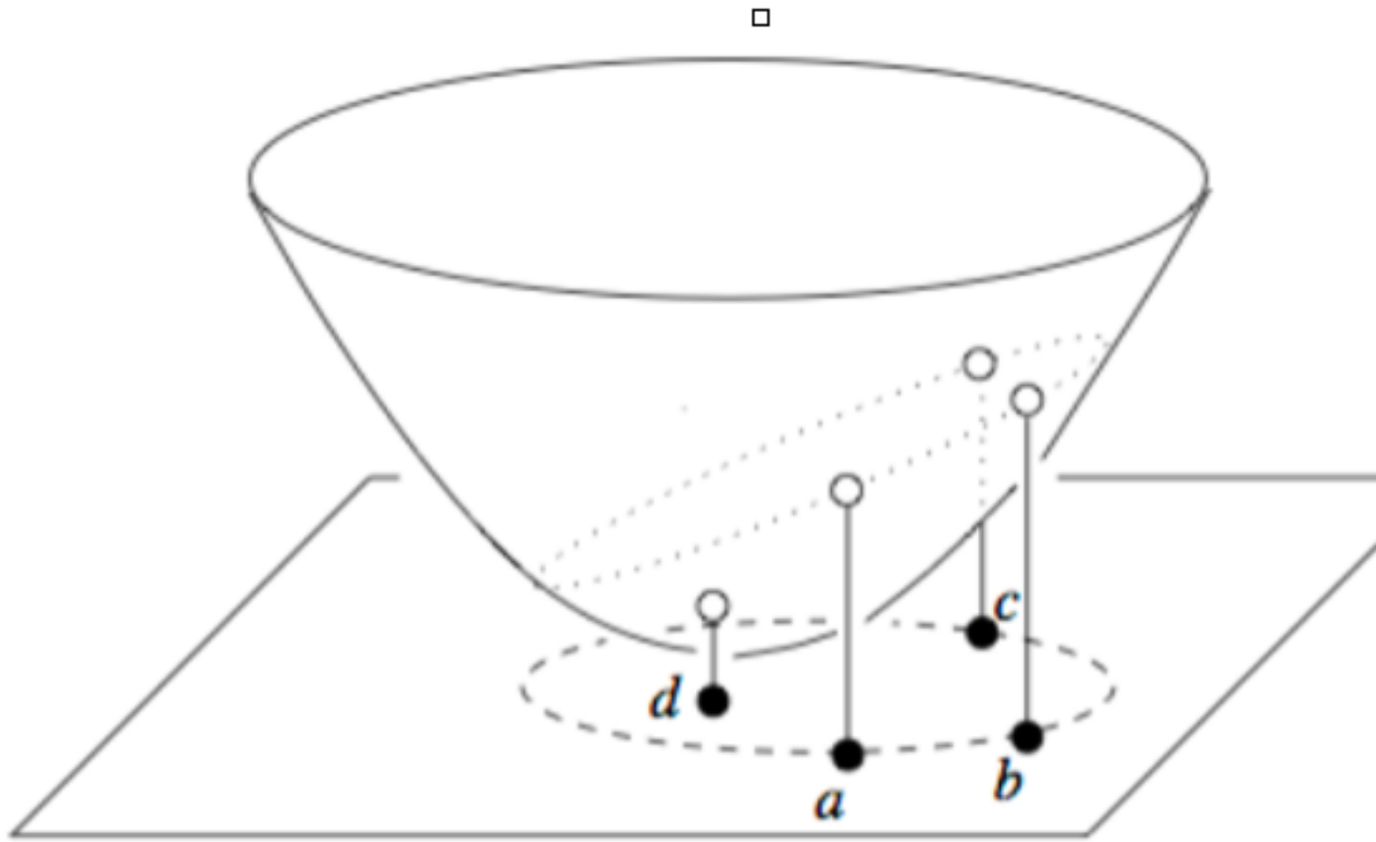
in 3D





face on CH: if all vertices on one side

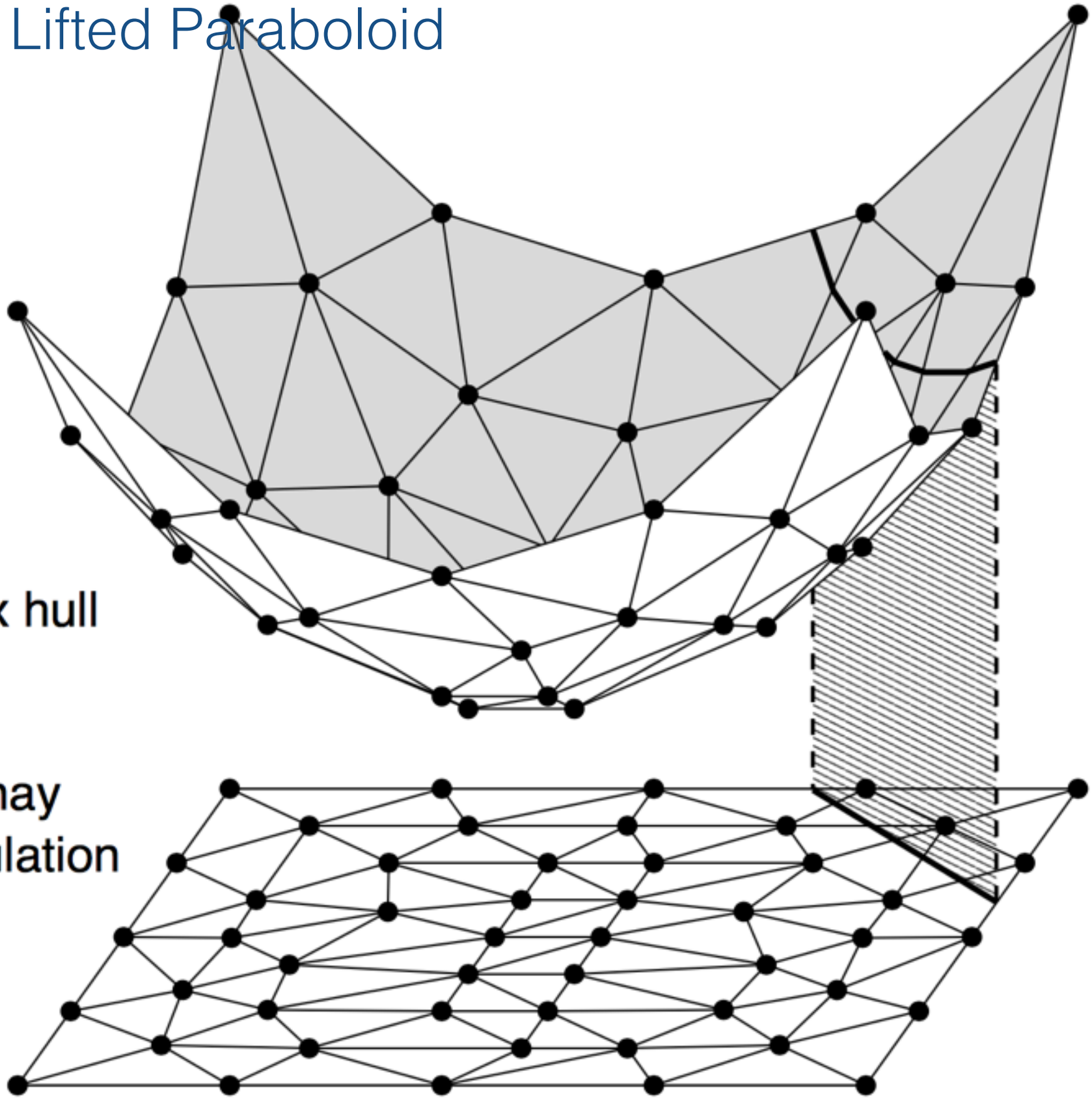
DT and Lifted Paraboloid



- Any triangle in DT is empty inside => on the paraboloid: no points below

- DT can be computed by computing the 3D CH 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