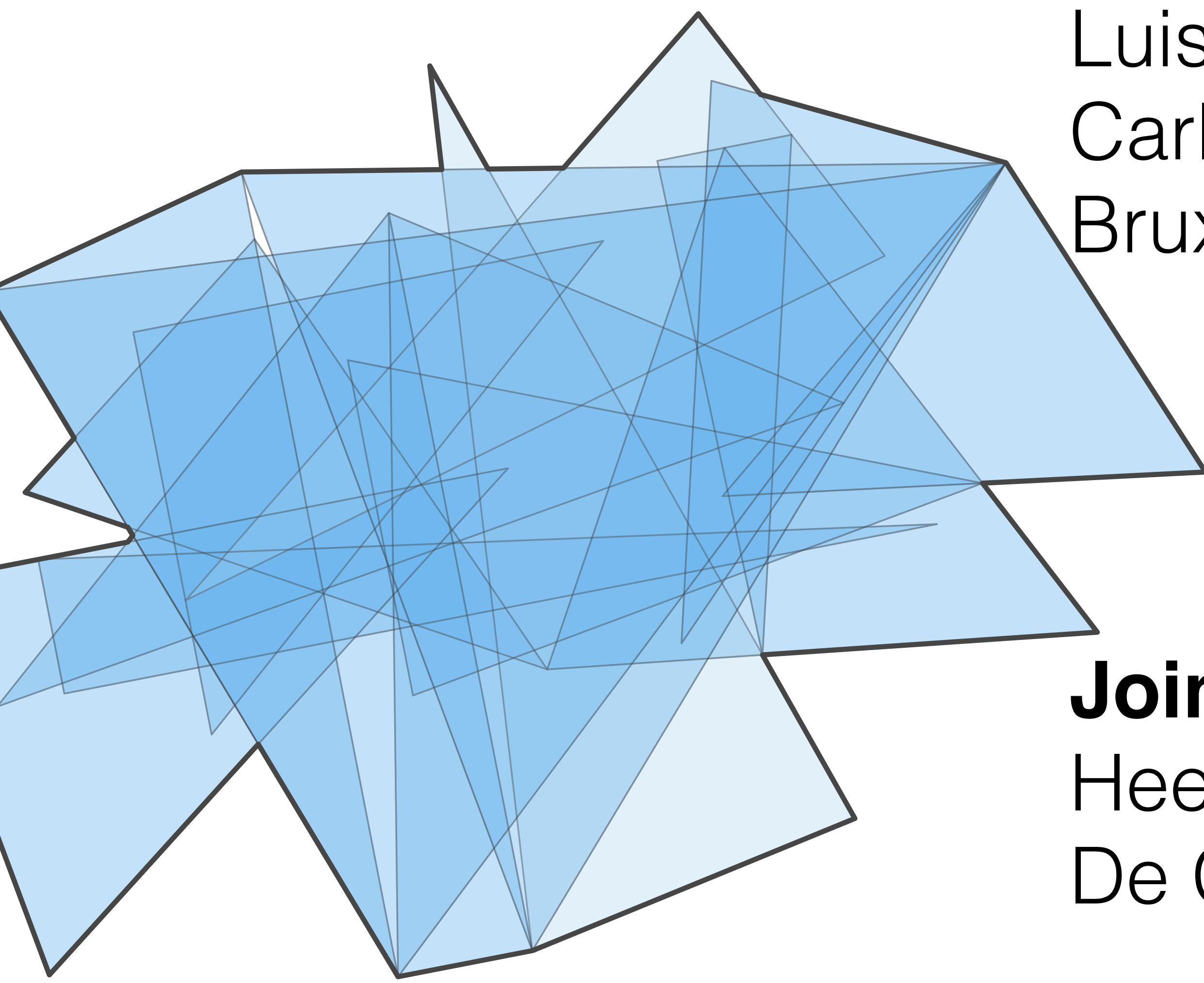
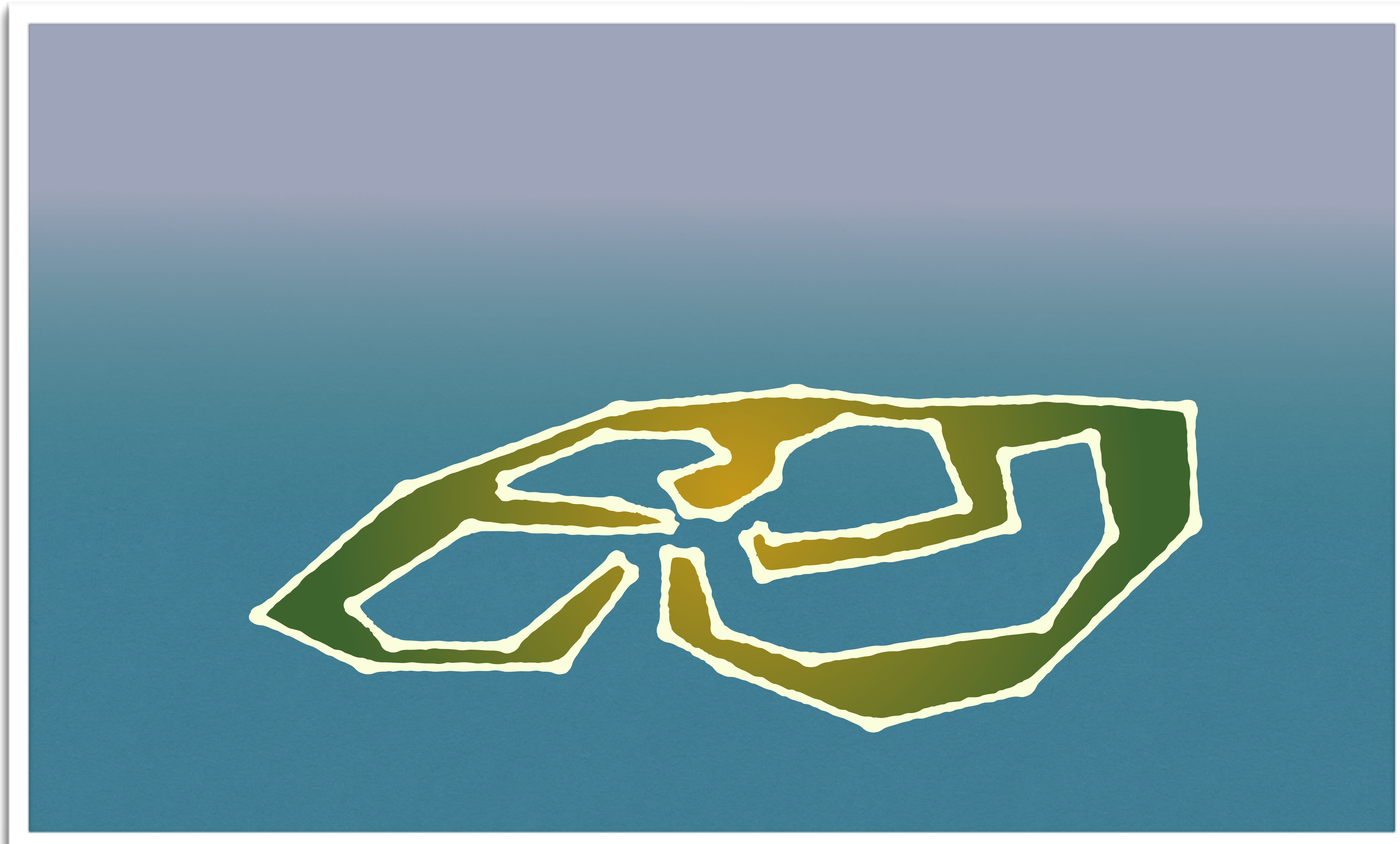
Linear time algorithms for geodesic problems on simple polygons.



Luis Barba
Carleton University / Université libre de Bruxelles

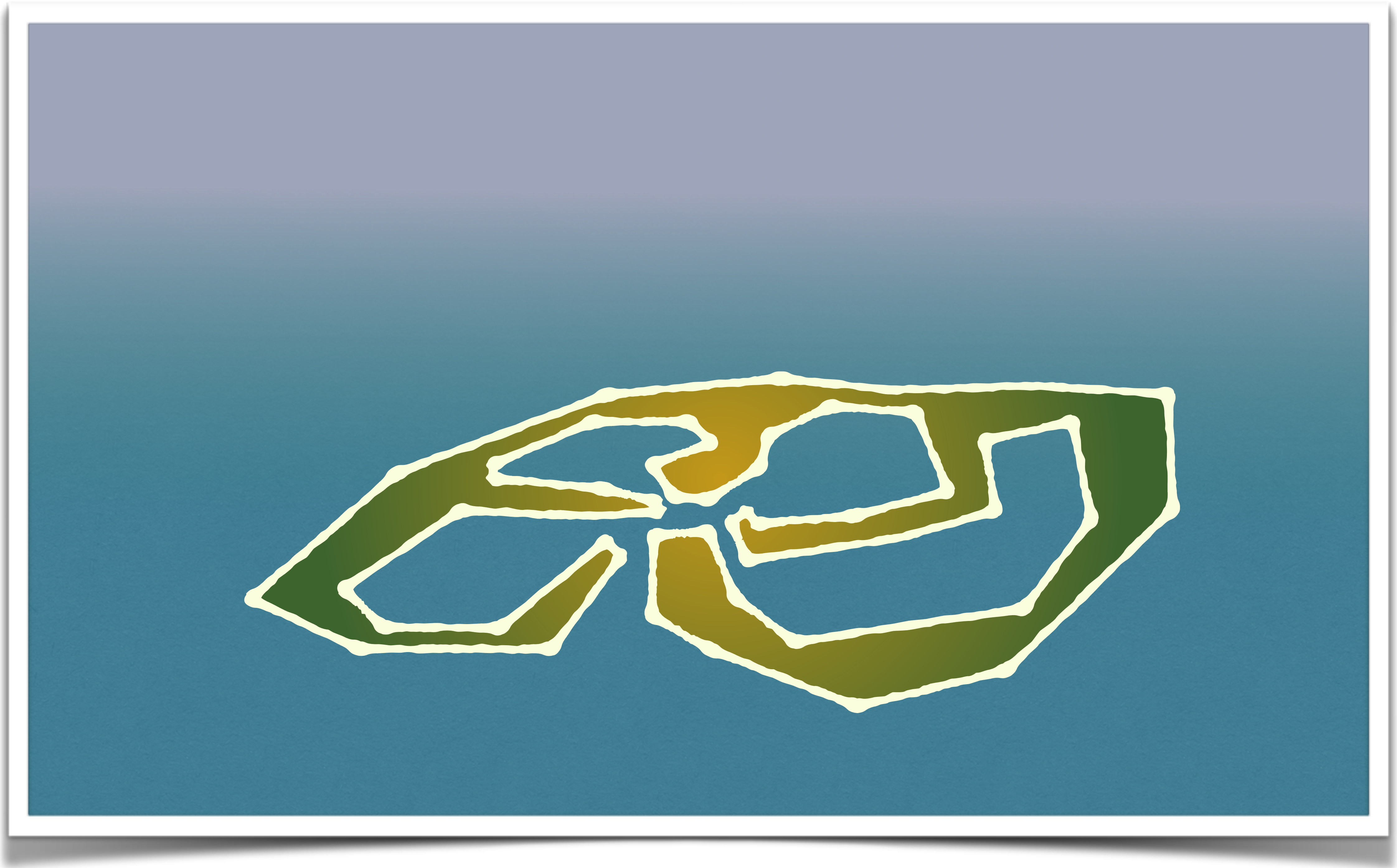
Joint work with:
Hee-Kap Ahn, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman and Eunjin Oh

THE PROBLEM

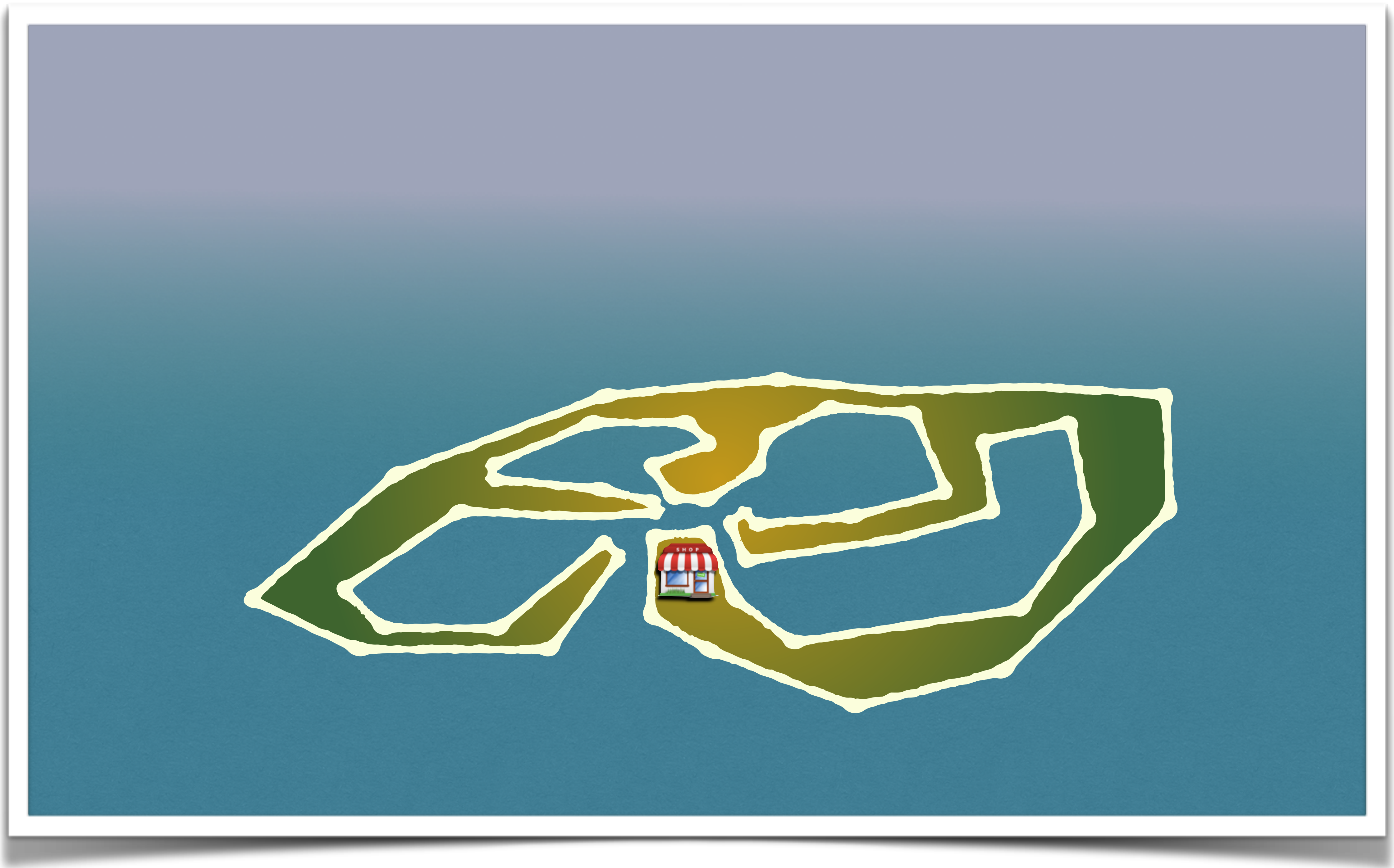


THE PROBLEM

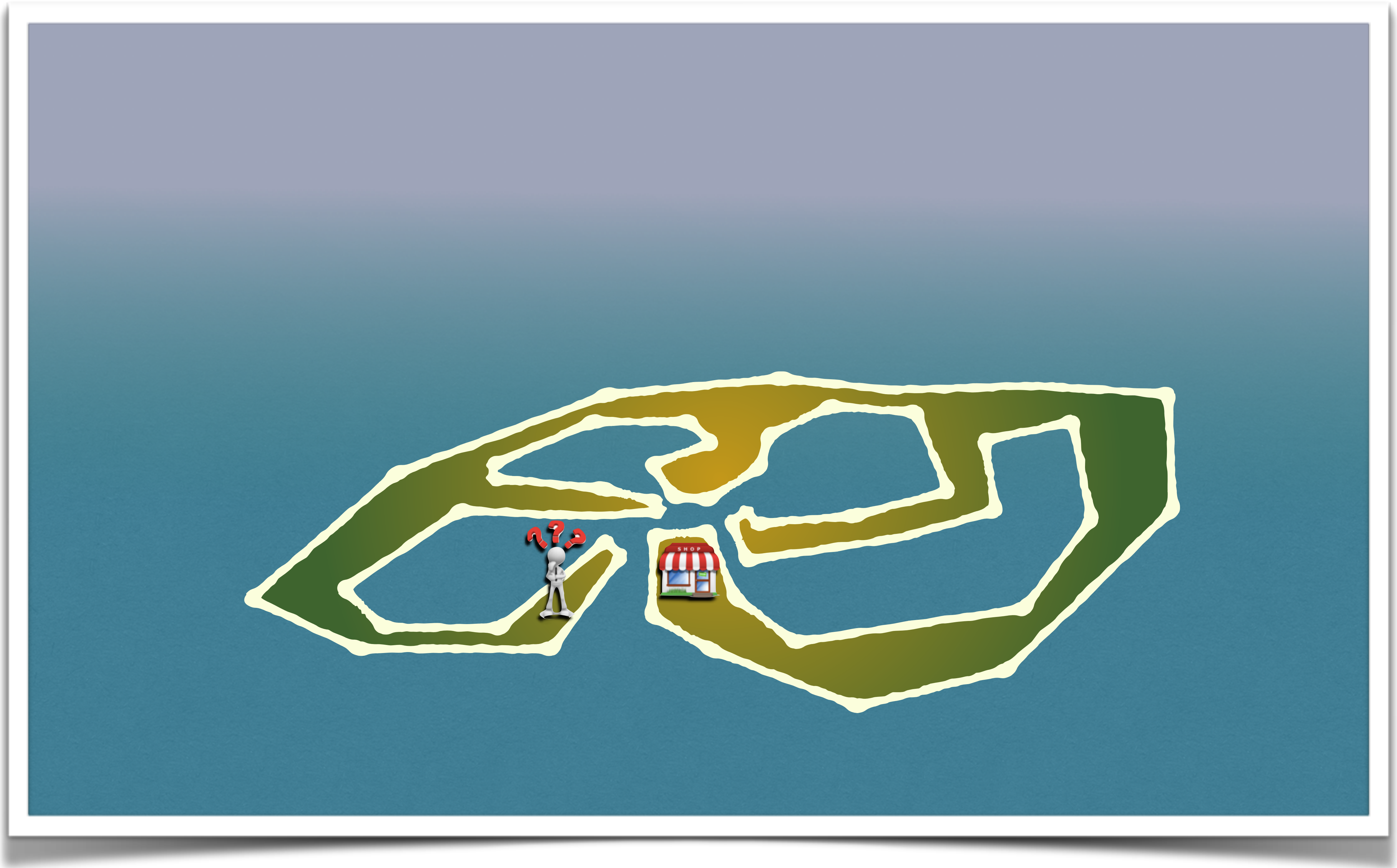
THE PROBLEM



THE PROBLEM

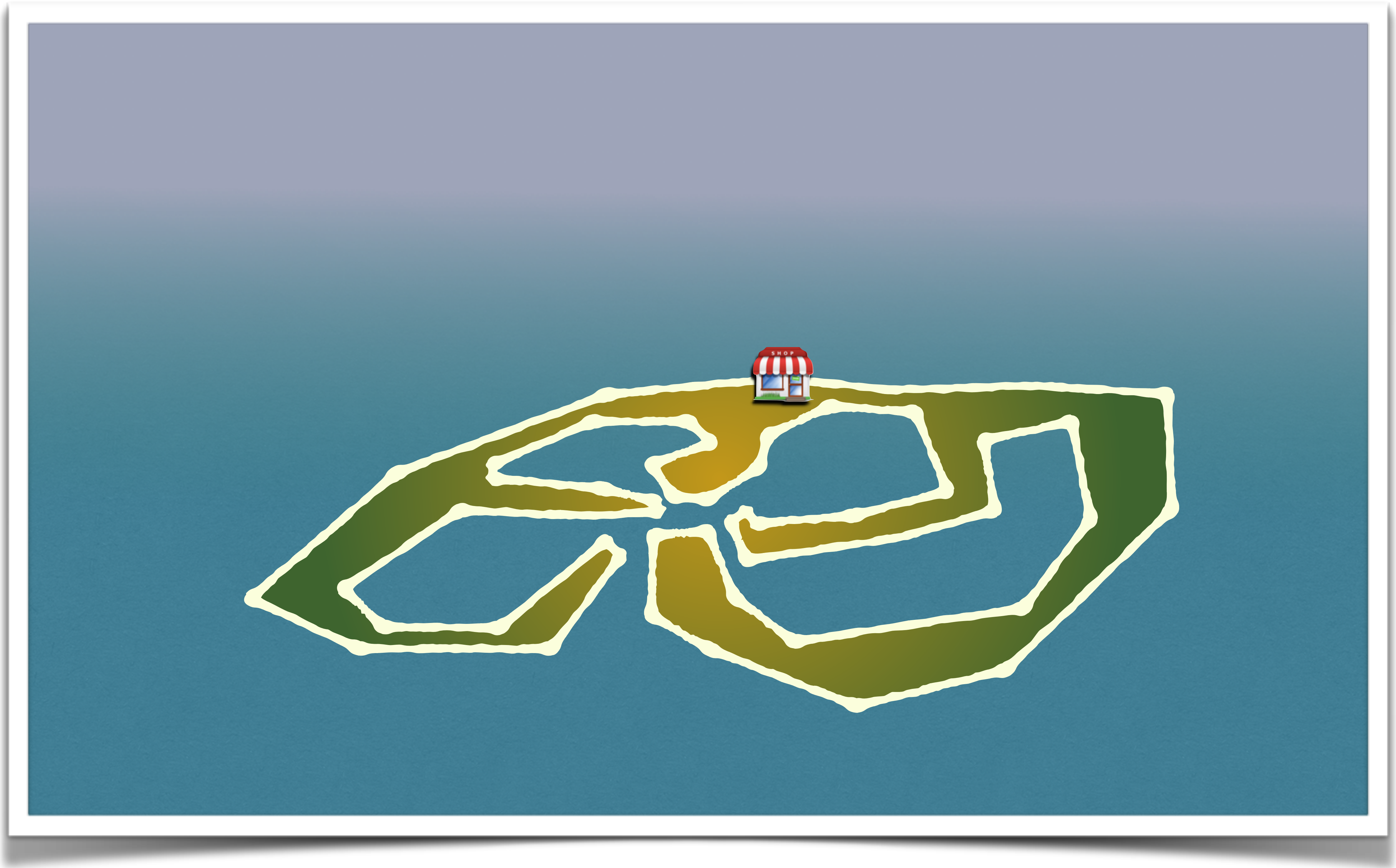


THE PROBLEM

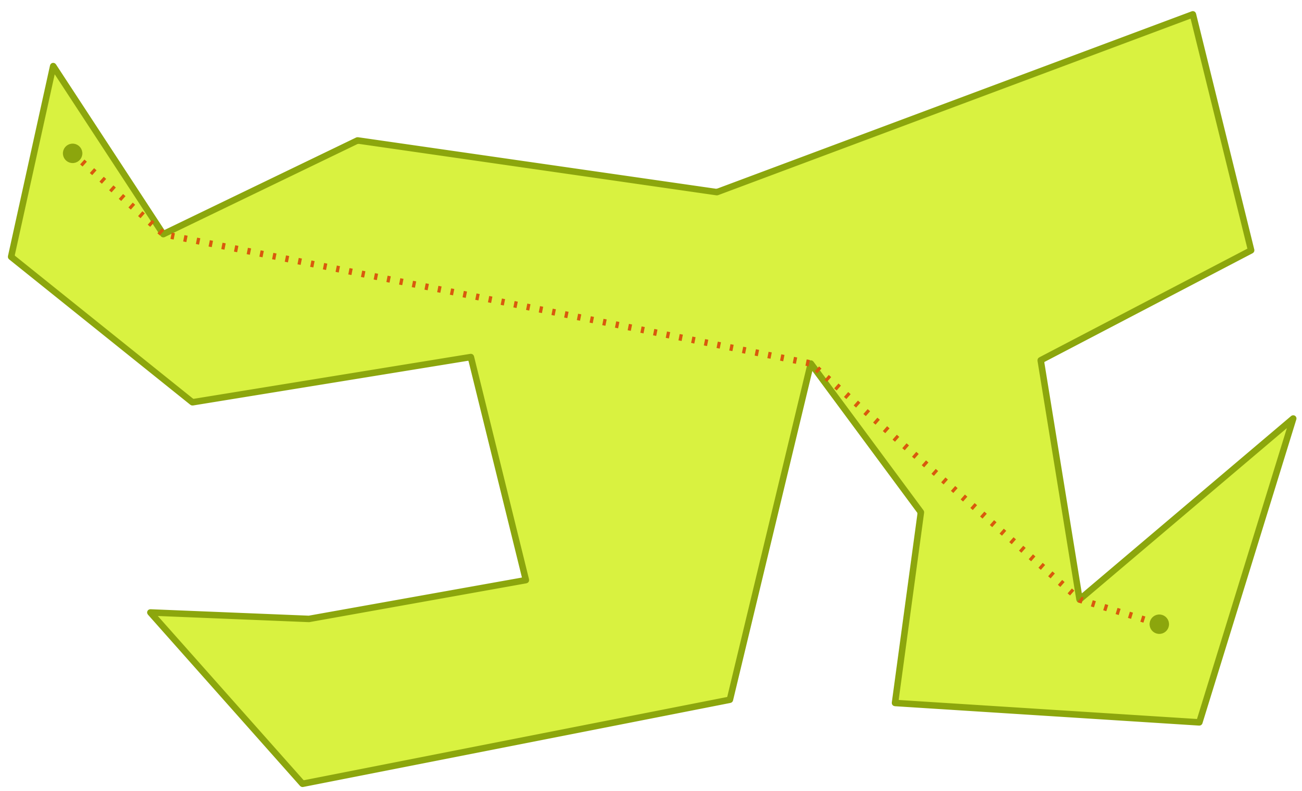


THE PROBLEM

THE PROBLEM

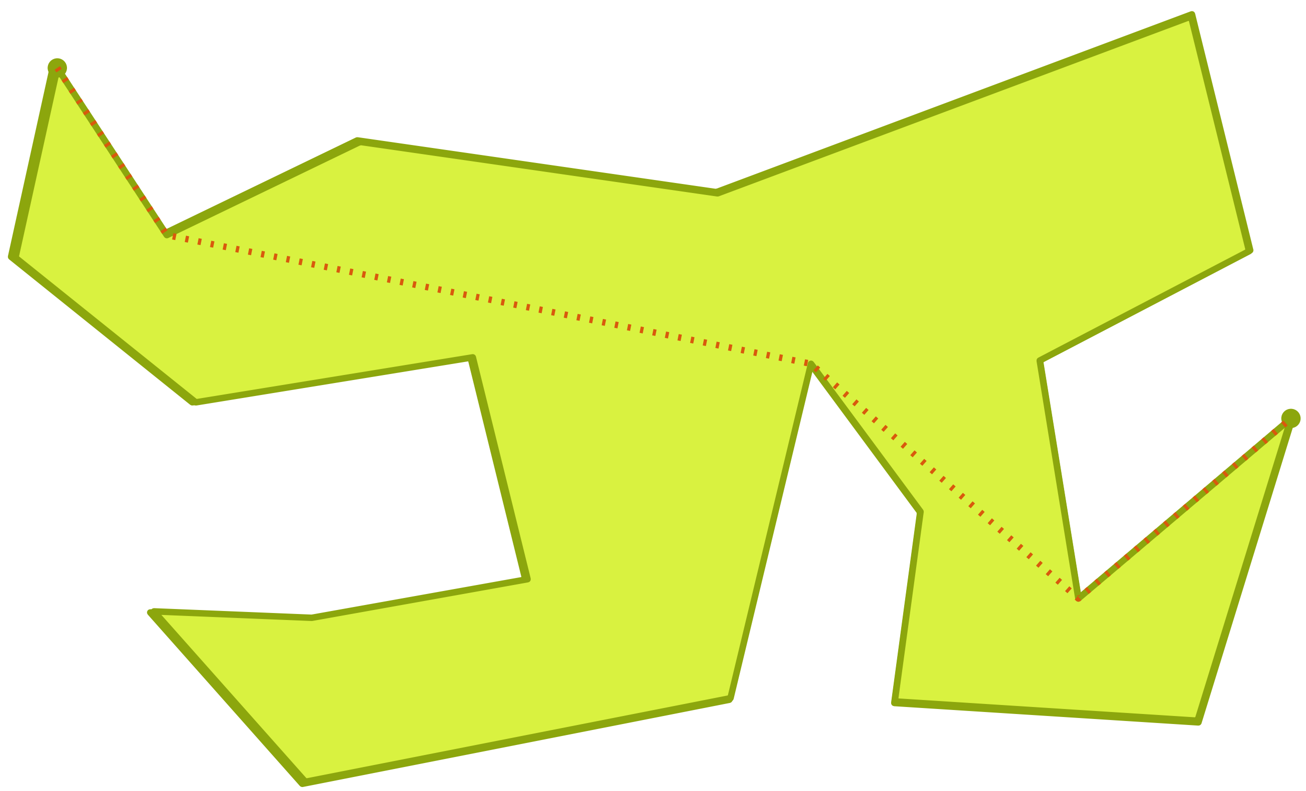


INTRODUCTION



- **Geodesic**: shortest path that stays within P
- Always exists and is unique
- Geodesic distance: sum of lengths of the segments

DIAMETER AND RADIUS



- **Diameter (diametral pair)**: largest geodesic distance
- **Radius (center)**: smallest distance to farthest neighbor

KNOWN RESULTS

Polygon	Diameter	Radius / Center	Farthest Voronoi
Simple	O(n) [HS'93]	O(n log n) [PSR'89]	O((n+m) log (n+m)) [AFW'93]

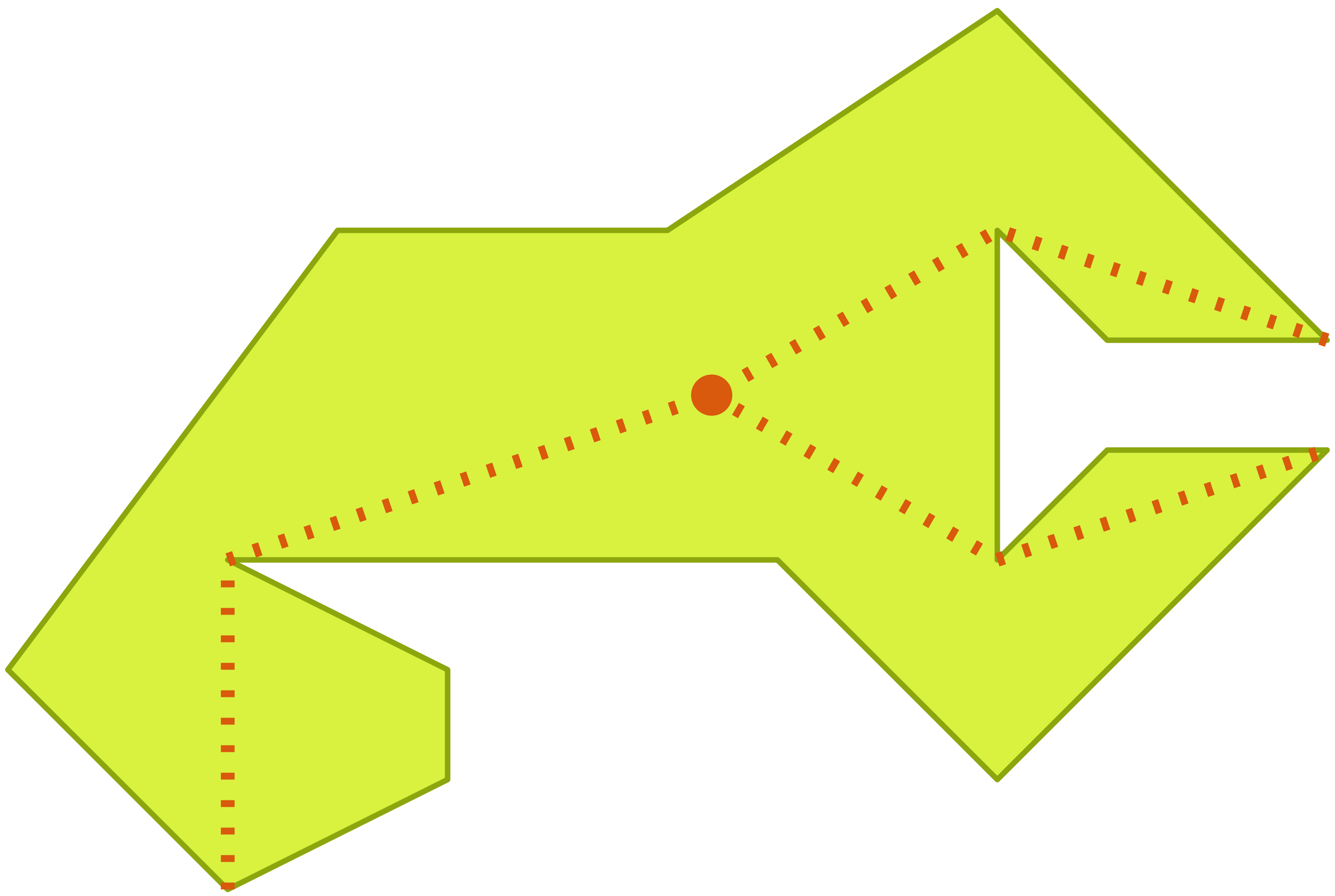
- All problems have been heavily studied in simple polygons
- Only the diameter problem matches the lower bound
- Several results with other metrics in recent years, but no progress in these problems

OUR RESULTS

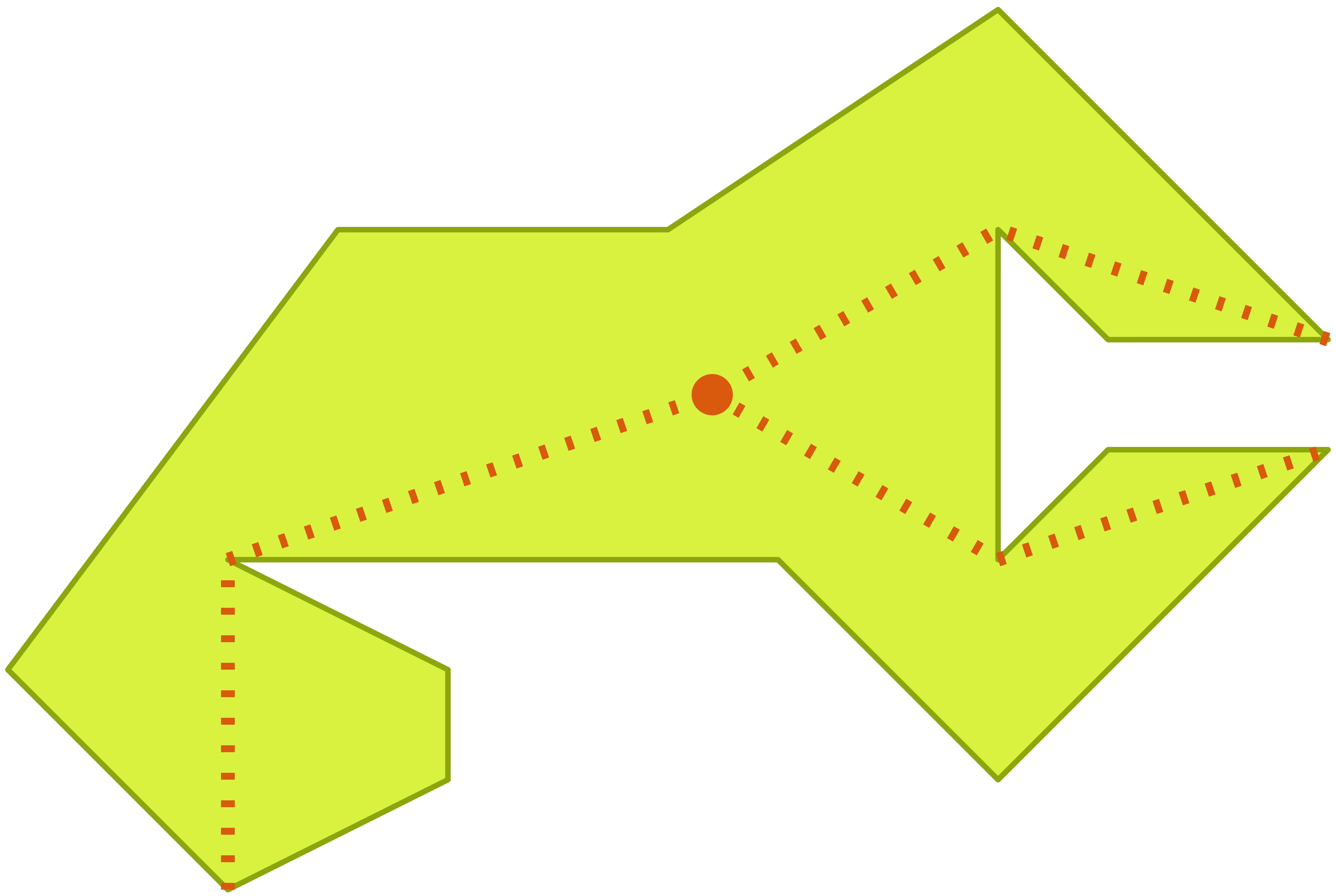
Polygon	Diameter	Radius / Center	Farthest Voronoi
Simple	O(n) [HS'93]	**O(n) [ABBCKO'15]**	O((n+m) log (n+m)) [AFW'93]

- All problems have been heavily studied in simple polygons
- Only the diameter problem matches the lower bound
- Several results with other metrics in recent years, but no progress in these problems

COMPUTING THE CENTER



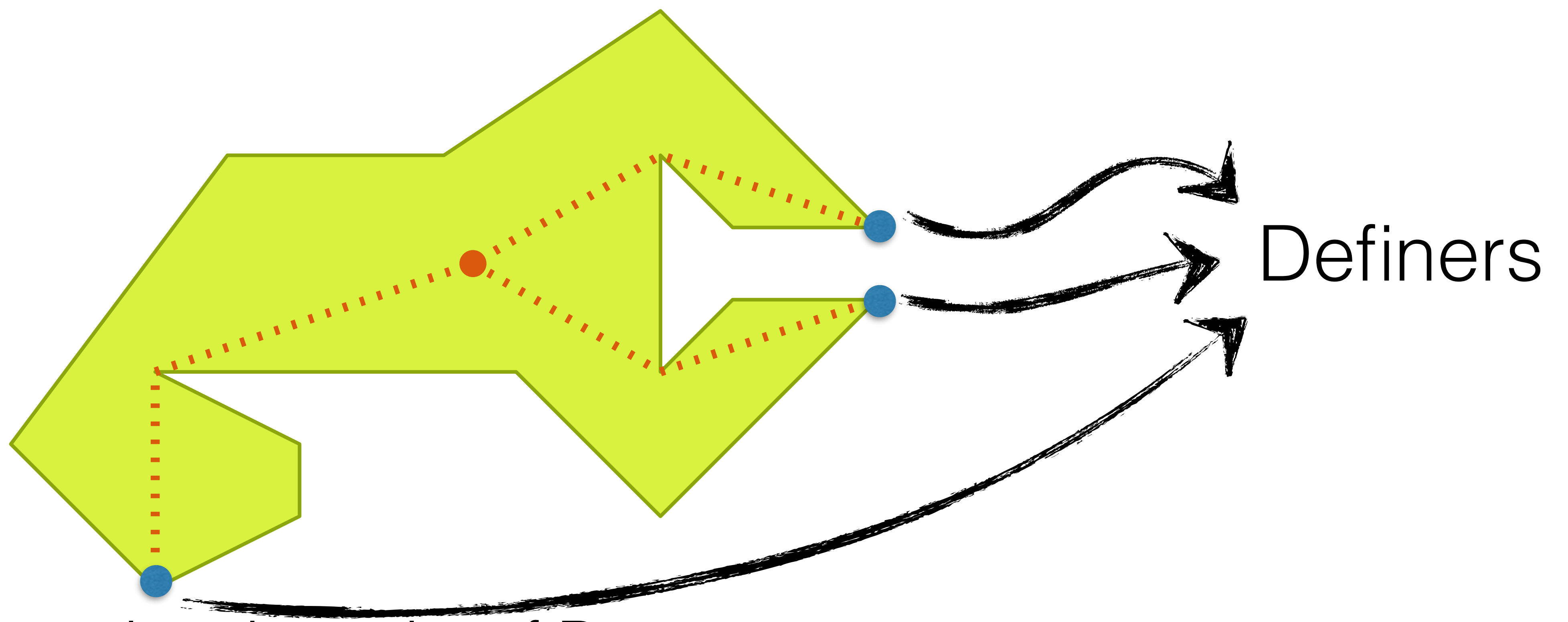
CENTER DEFINITION



Let F(p)=max{d(p,q)} for all q in P

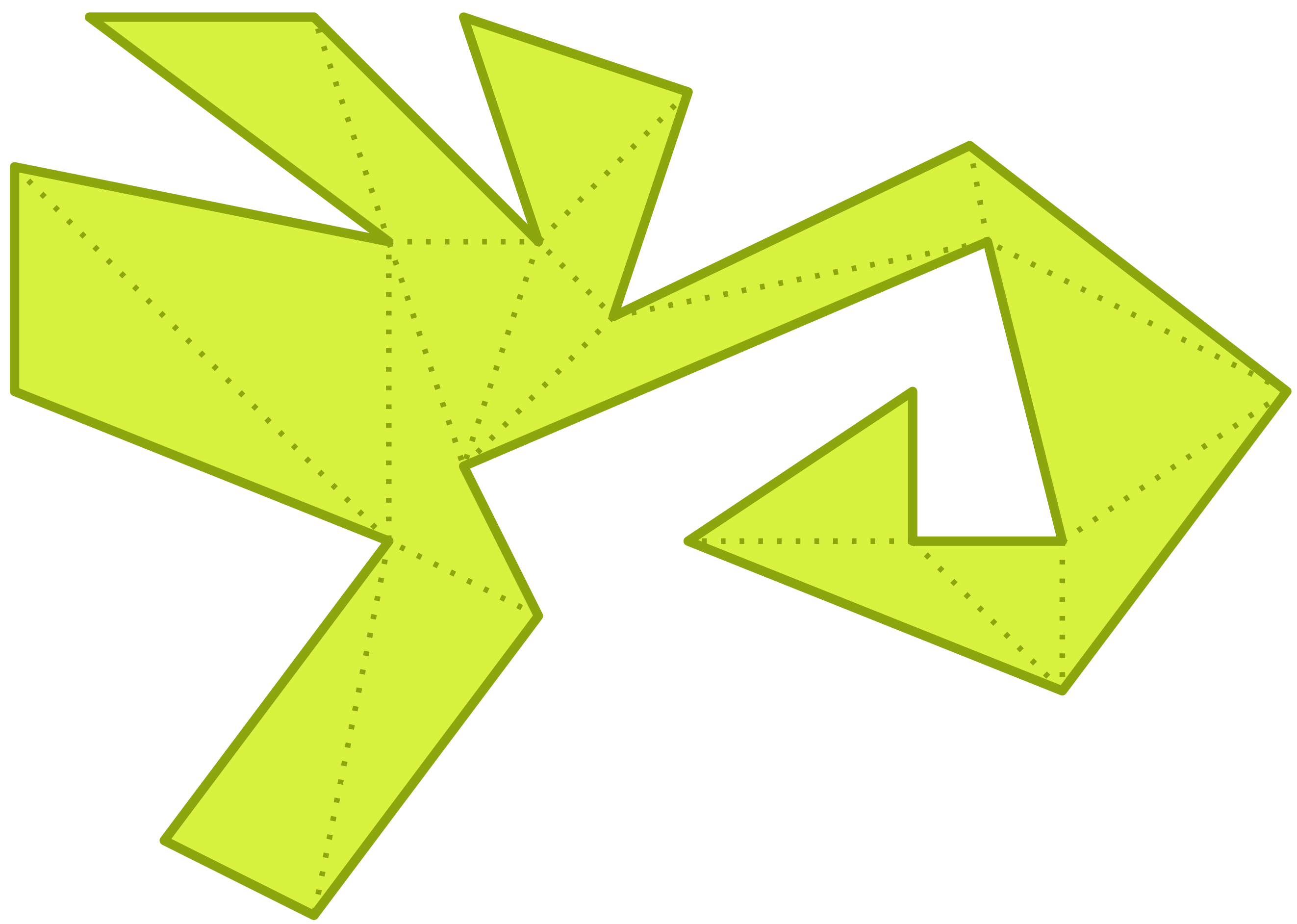
- The center minimizes F(p)
- Location is (often) unrelated to the diameter

NAIVE ALGORITHM



- Center may be an interior point of P
  - Its farthest neighbors are vertices (at most 3 in general position)
  - Vertex of the (geodesic) farthest Voronoi
- $O(n^3)$ candidates
- Verifying in polynomial time also possible

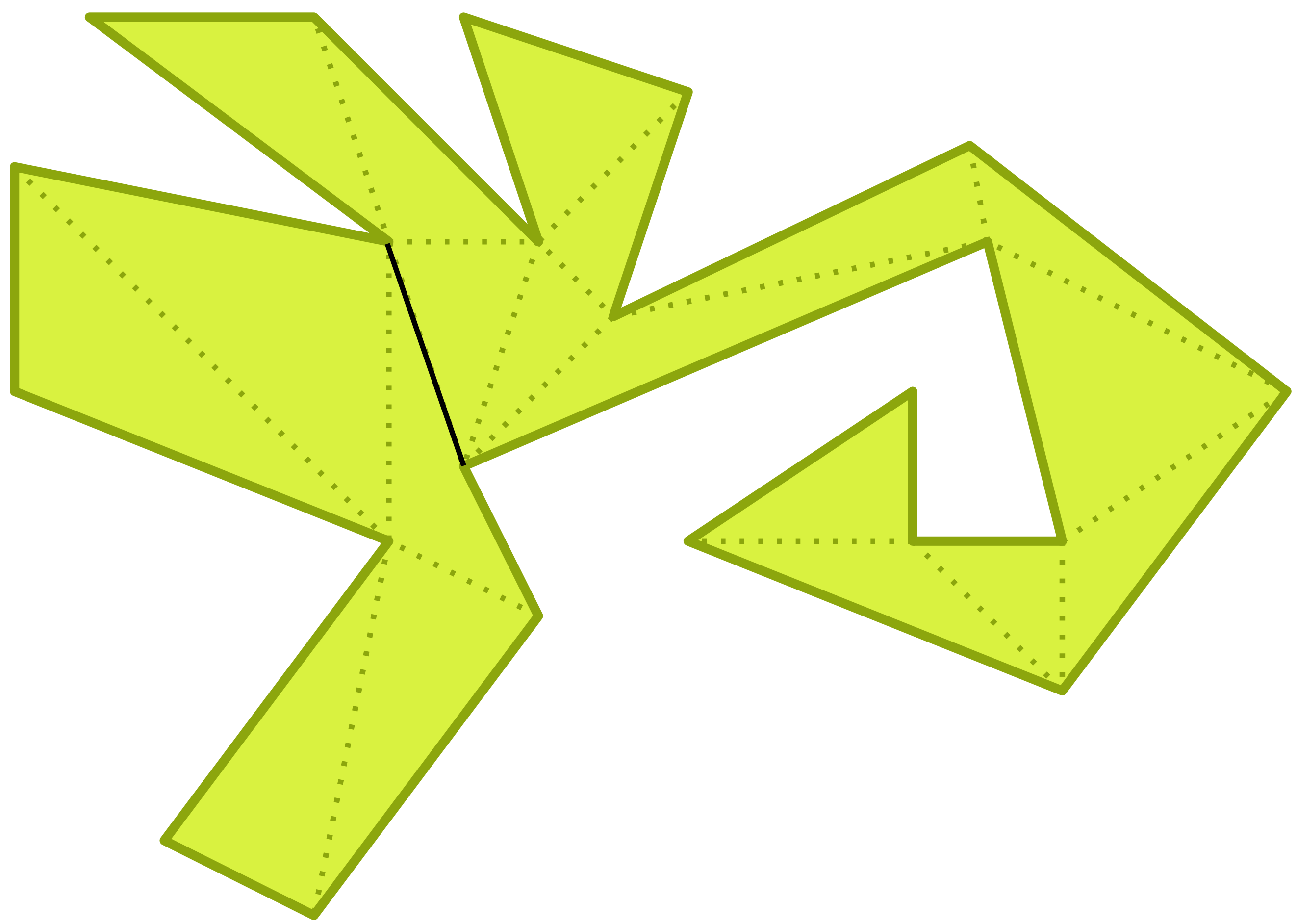
EFFICIENT ALGORITHM'89



Chord-oracle query

- Given a chord of P, determines which side contains c
- Runs in O(n) time

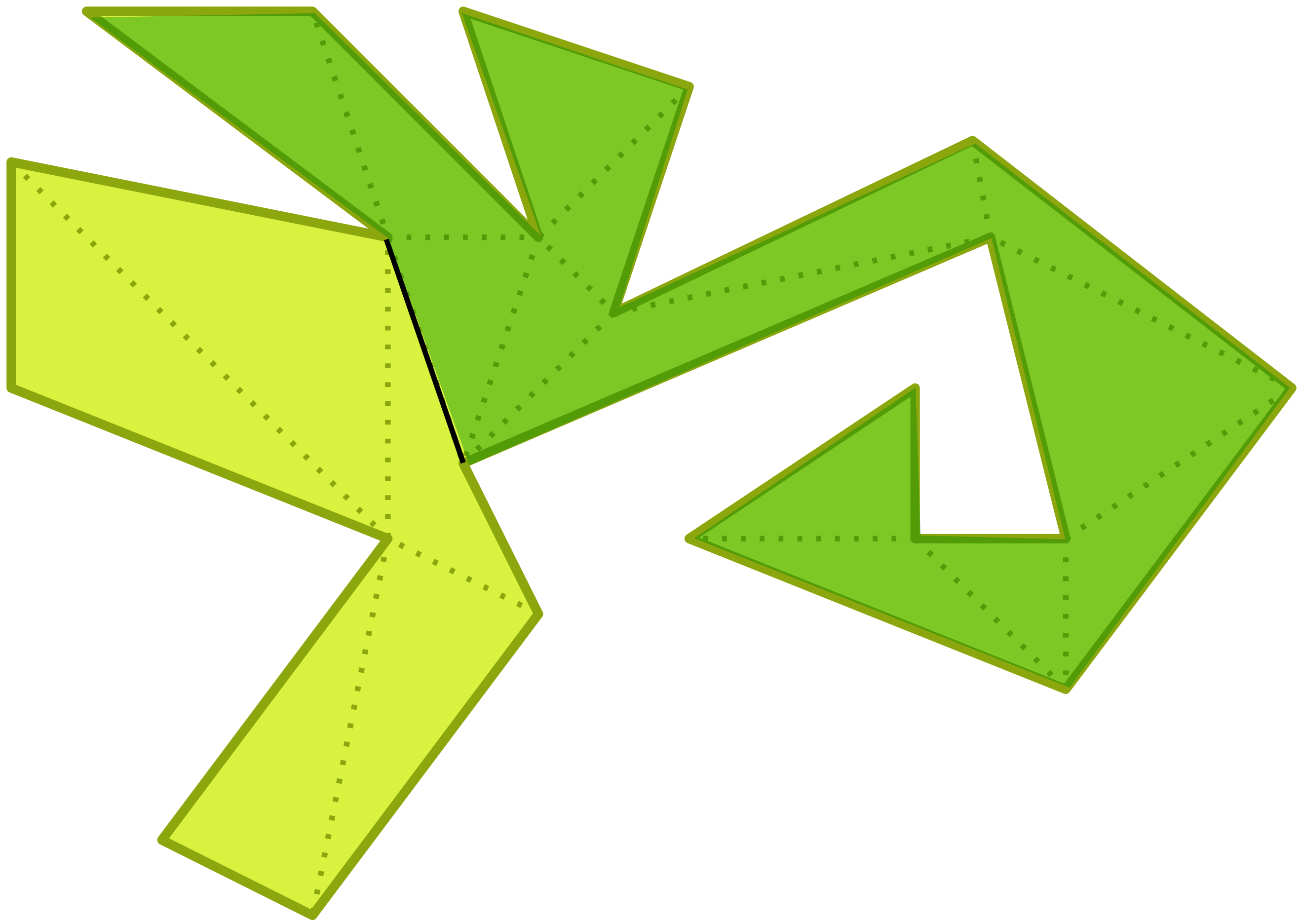
EFFICIENT ALGORITHM'89



Chord-oracle query

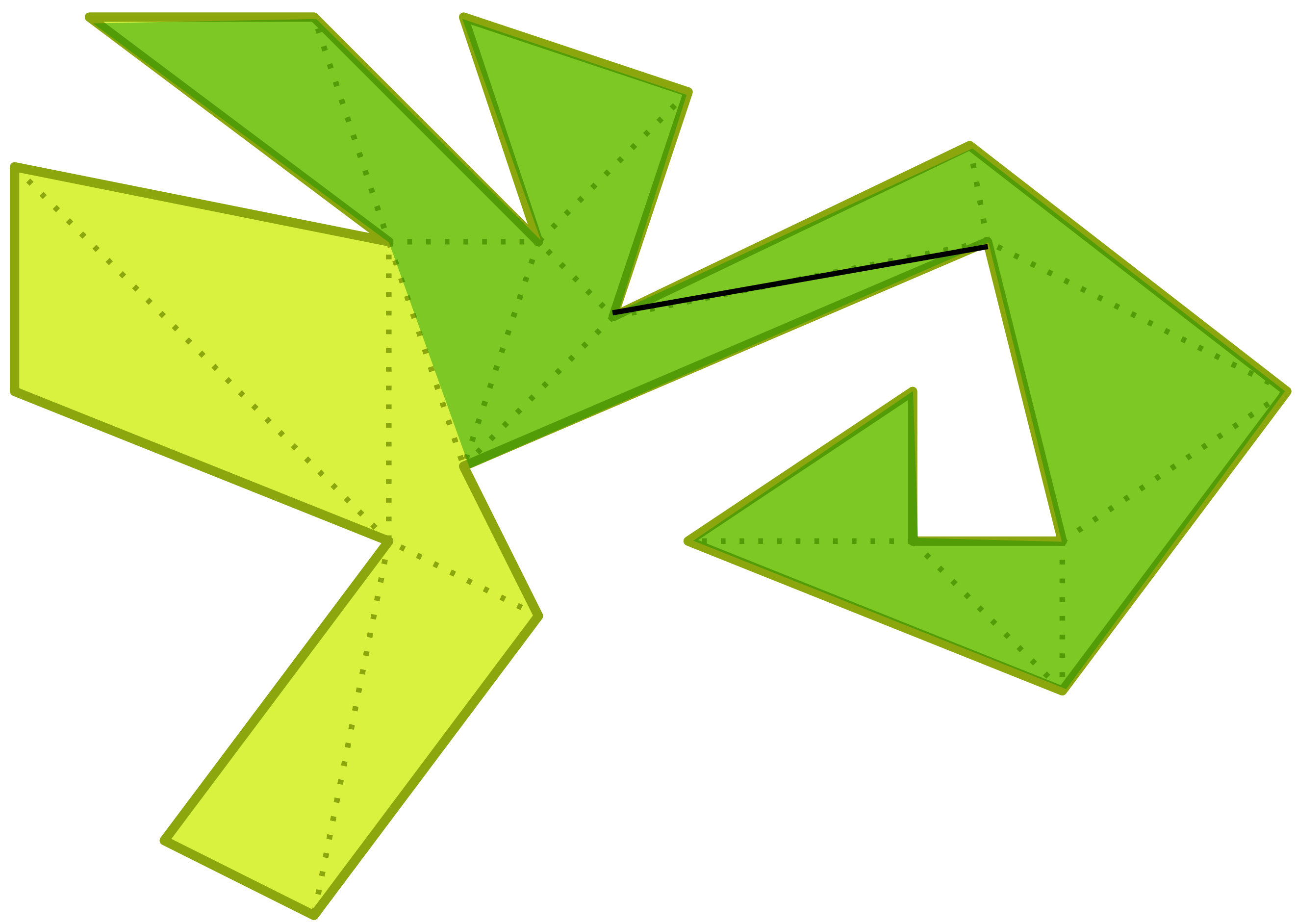
- Given a chord of P, determines which side contains c
- Runs in O(n) time

EFFICIENT ALGORITHM'89



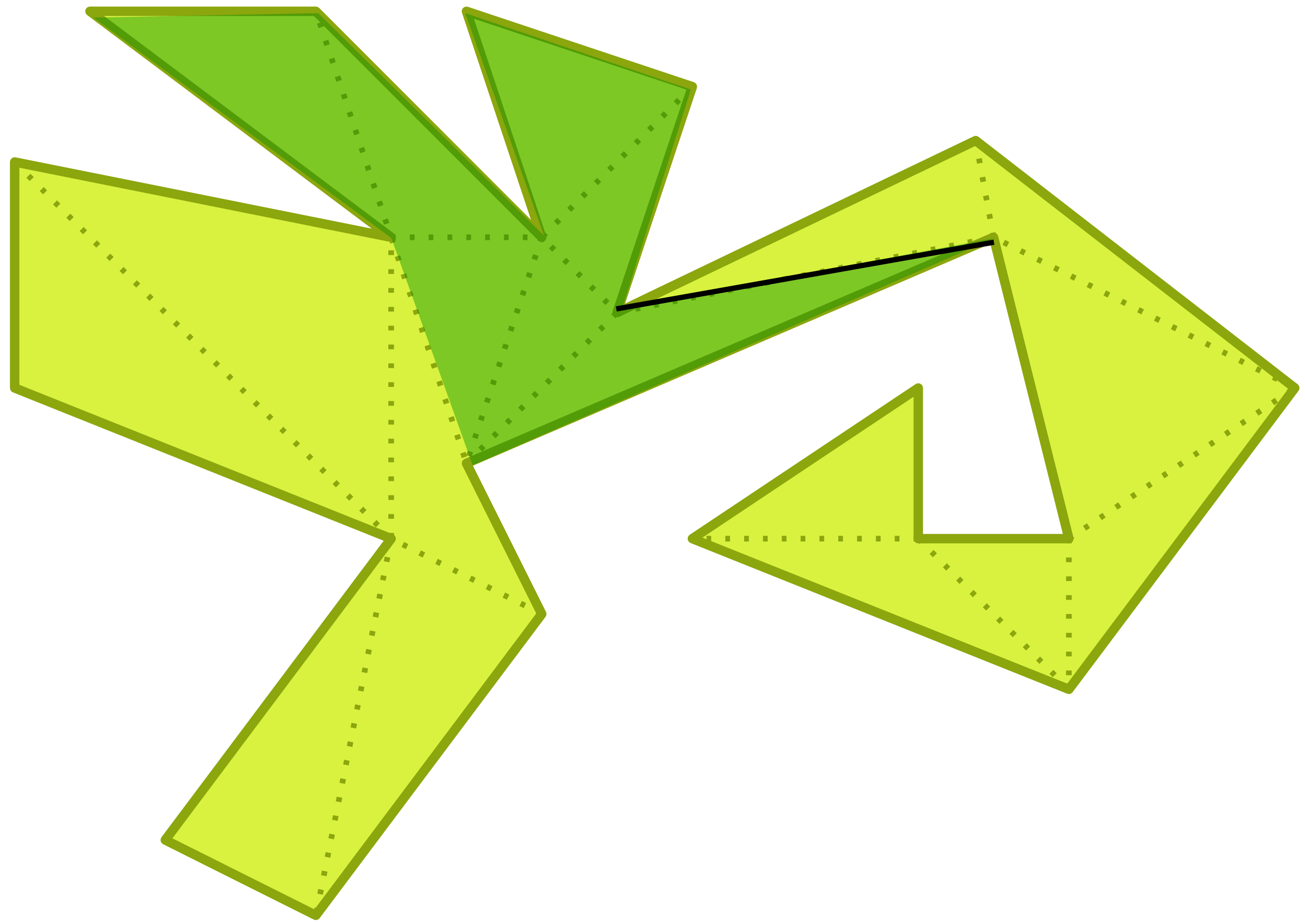
- Binary search to narrow the search to a triangle

EFFICIENT ALGORITHM'89



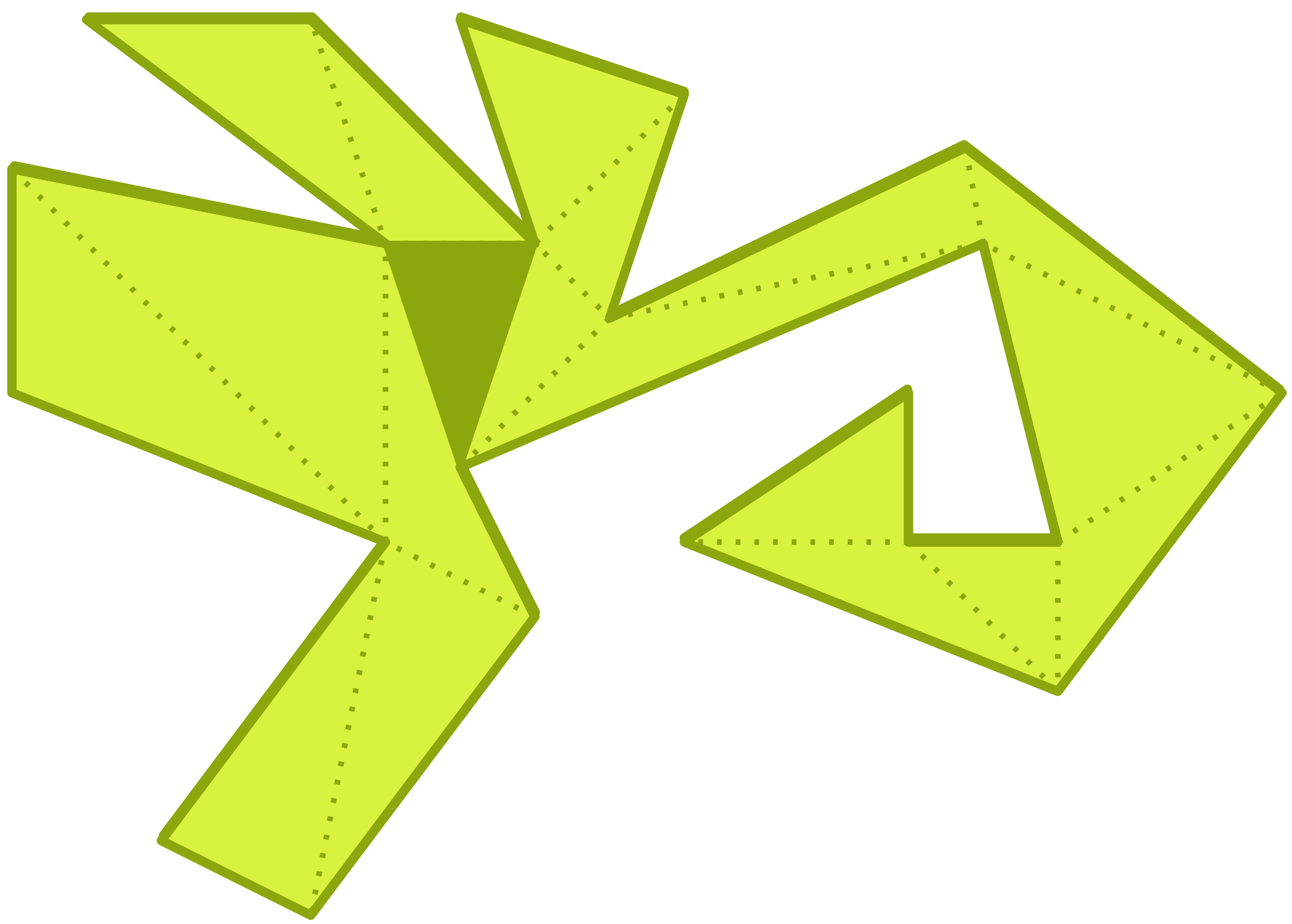
- Binary search to narrow the search to a triangle

EFFICIENT ALGORITHM'89



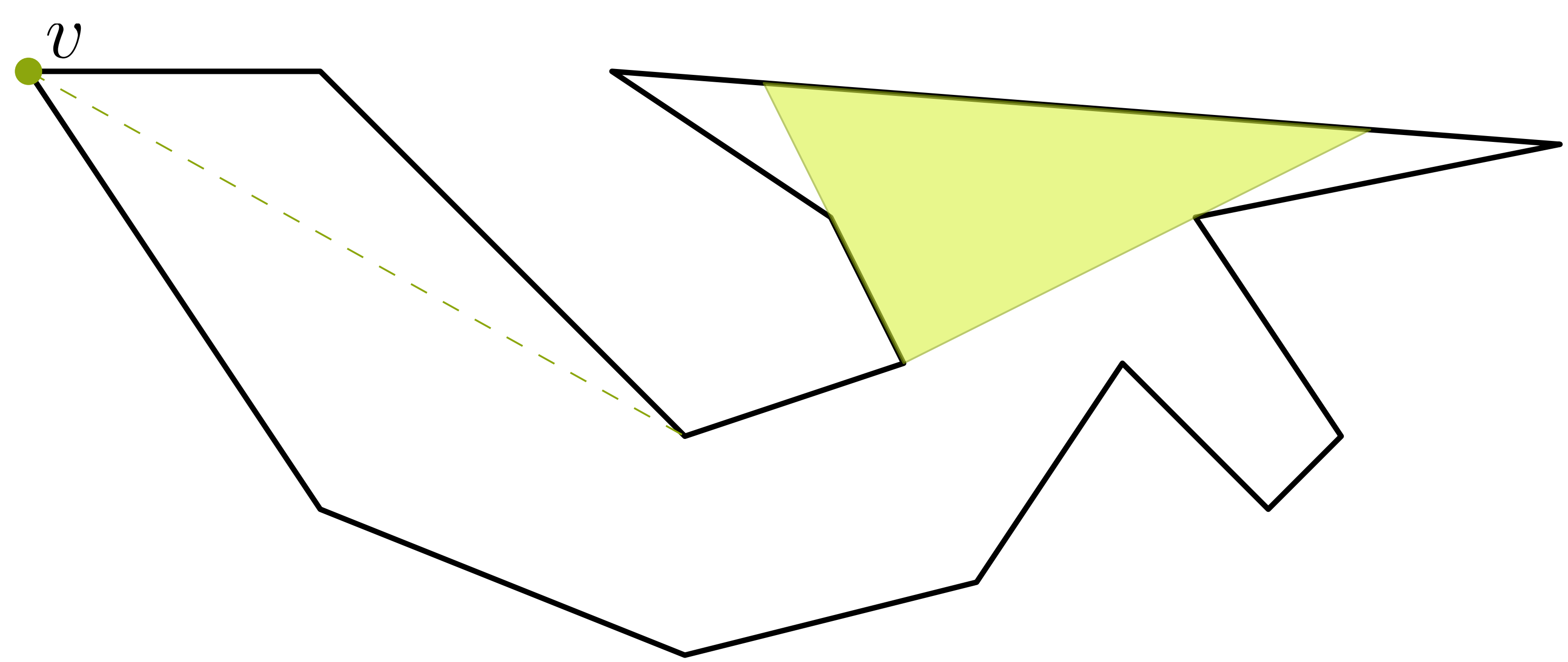
- Binary search to narrow the search to a triangle

EFFICIENT ALGORITHM'89



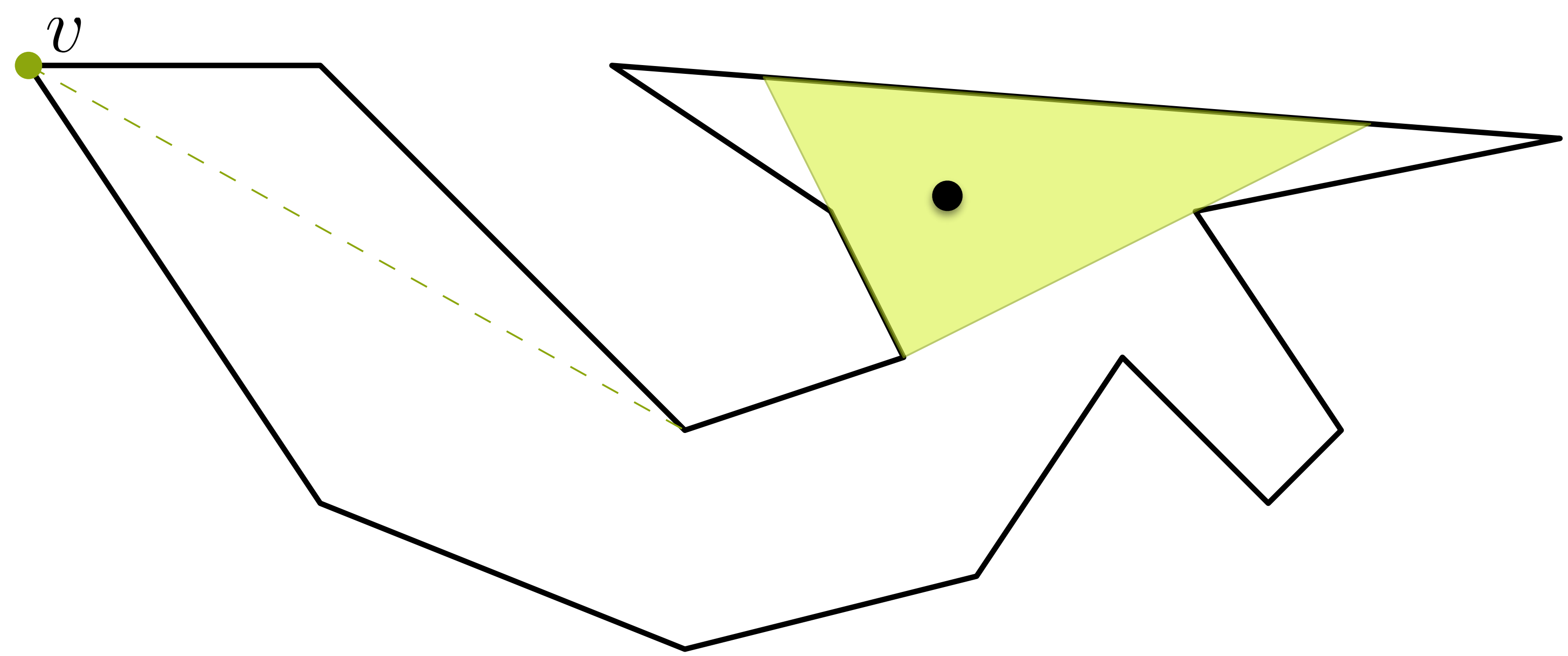
- Binary search to narrow the search to a triangle
- Use optimization tricks to find the center
- No pruning possible -> Θ(n log n) time

APEXED TRIANGLES



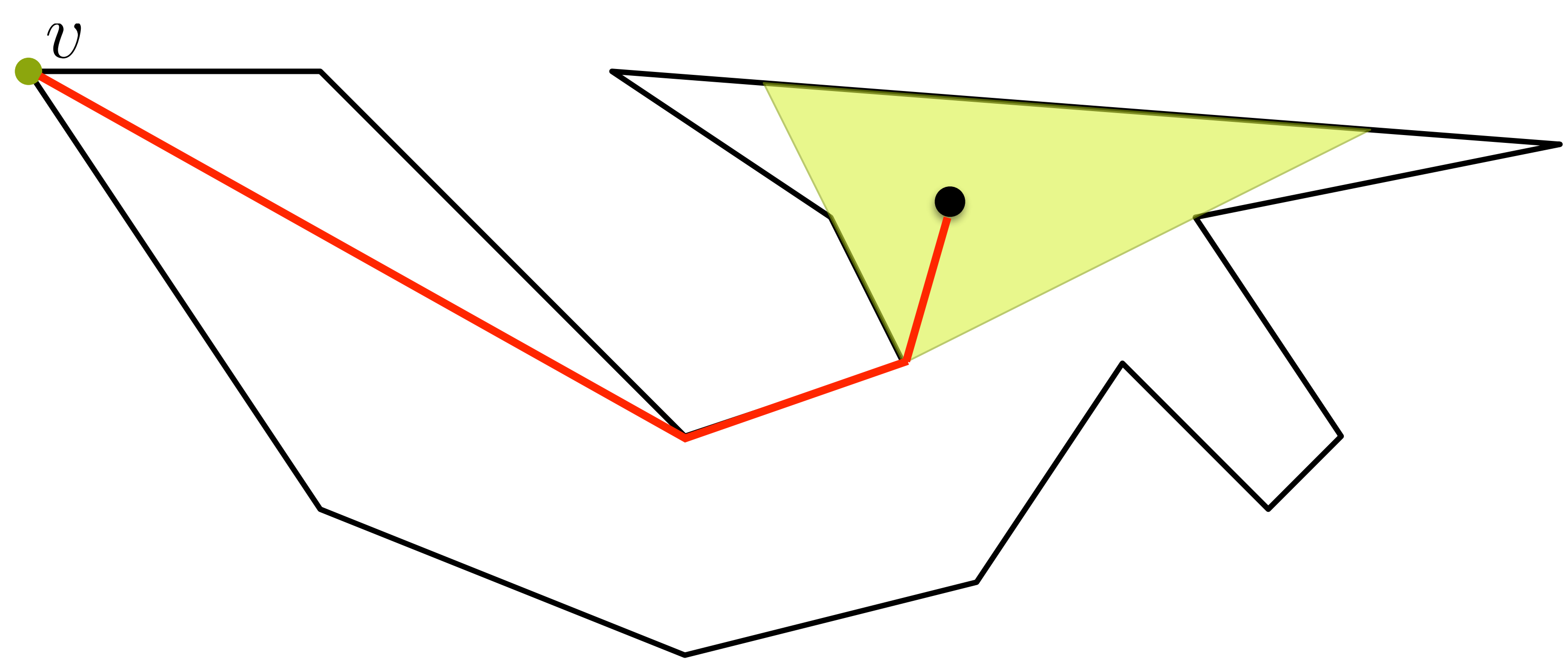
- Encodes distance to a potential farthest neighbor v
- All points in the triangle have (topologically equivalent) paths to v
  - Simple (quadratic) function to encode distance
- We ignore P and work on the triangles

APEXED TRIANGLES



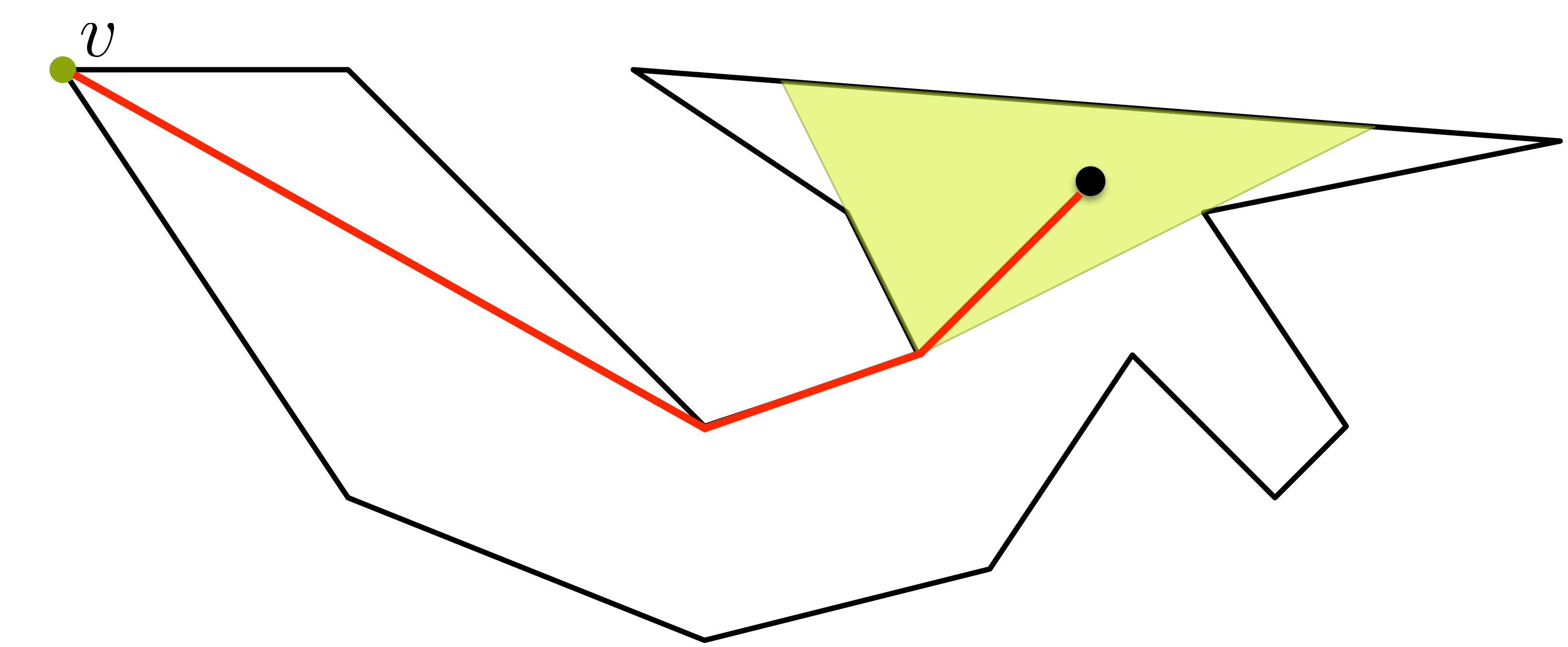
- Encodes distance to a potential farthest neighbor v
- All points in the triangle have (topologically equivalent) paths to v
  - Simple (quadratic) function to encode distance
- We ignore P and work on the triangles

APEXED TRIANGLES



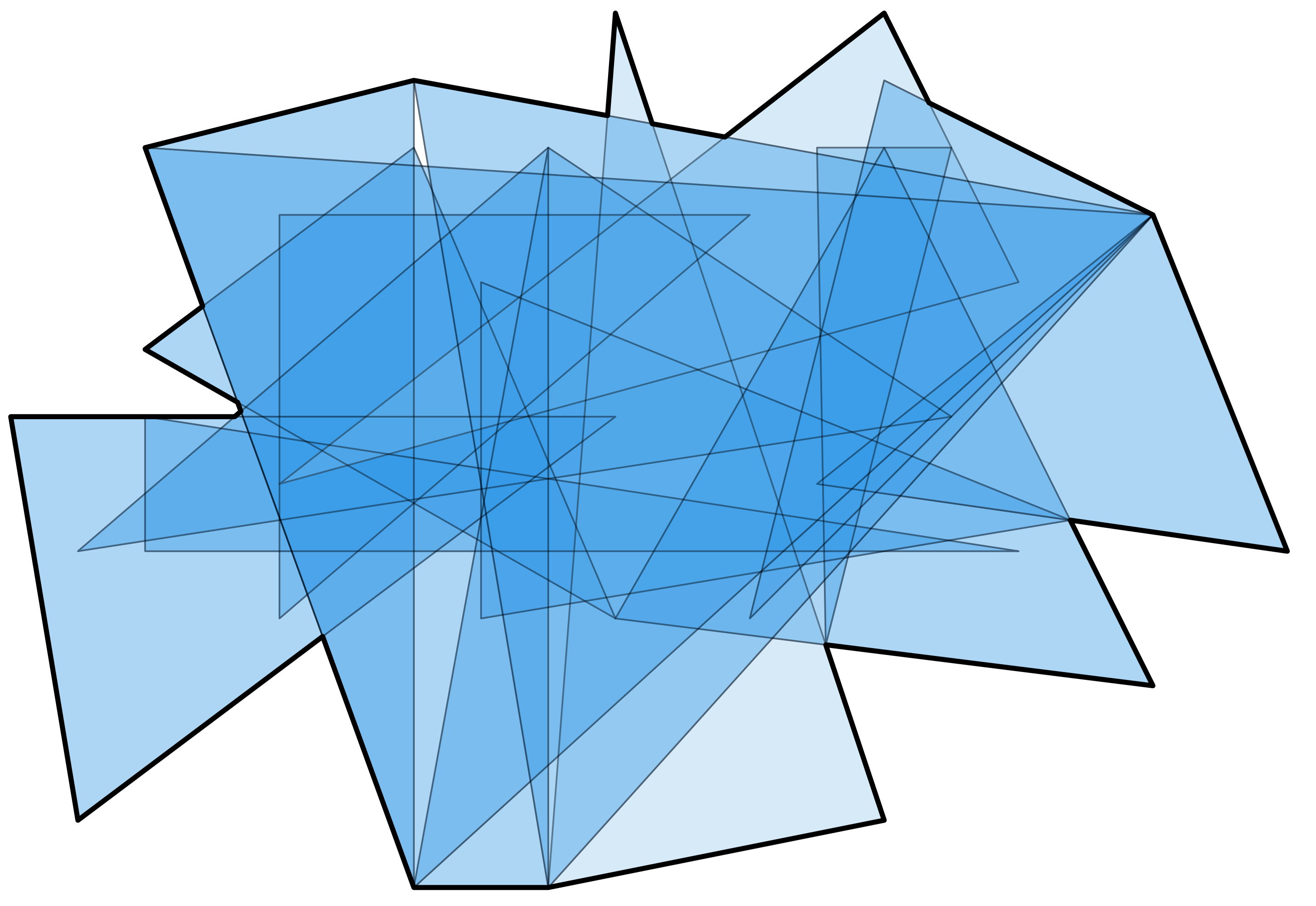
- Encodes distance to a potential farthest neighbor v
- All points in the triangle have (topologically equivalent) paths to v
  - Simple (quadratic) function to encode distance
- We ignore P and work on the triangles

APEXED TRIANGLES



- Encodes distance to a potential farthest neighbor v
- All points in the triangle have (topologically equivalent) paths to v
  - Simple (quadratic) function to encode distance
- We ignore P and work on the triangles

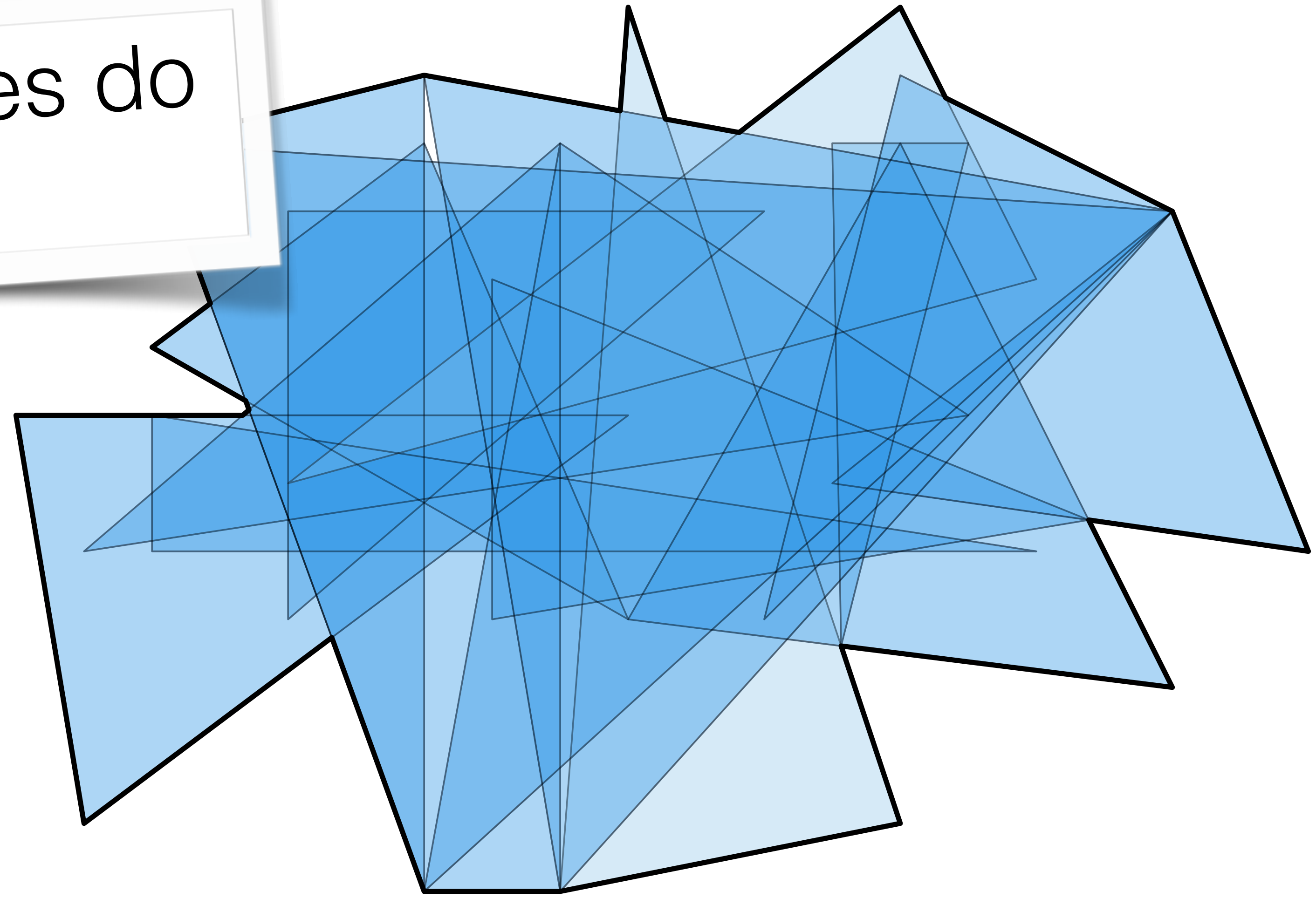
OUR APPROACH



- Cover P with apexed triangles that encode F(p)
    - Ignore P
    - Use cuttings to prune the triangles and recurse

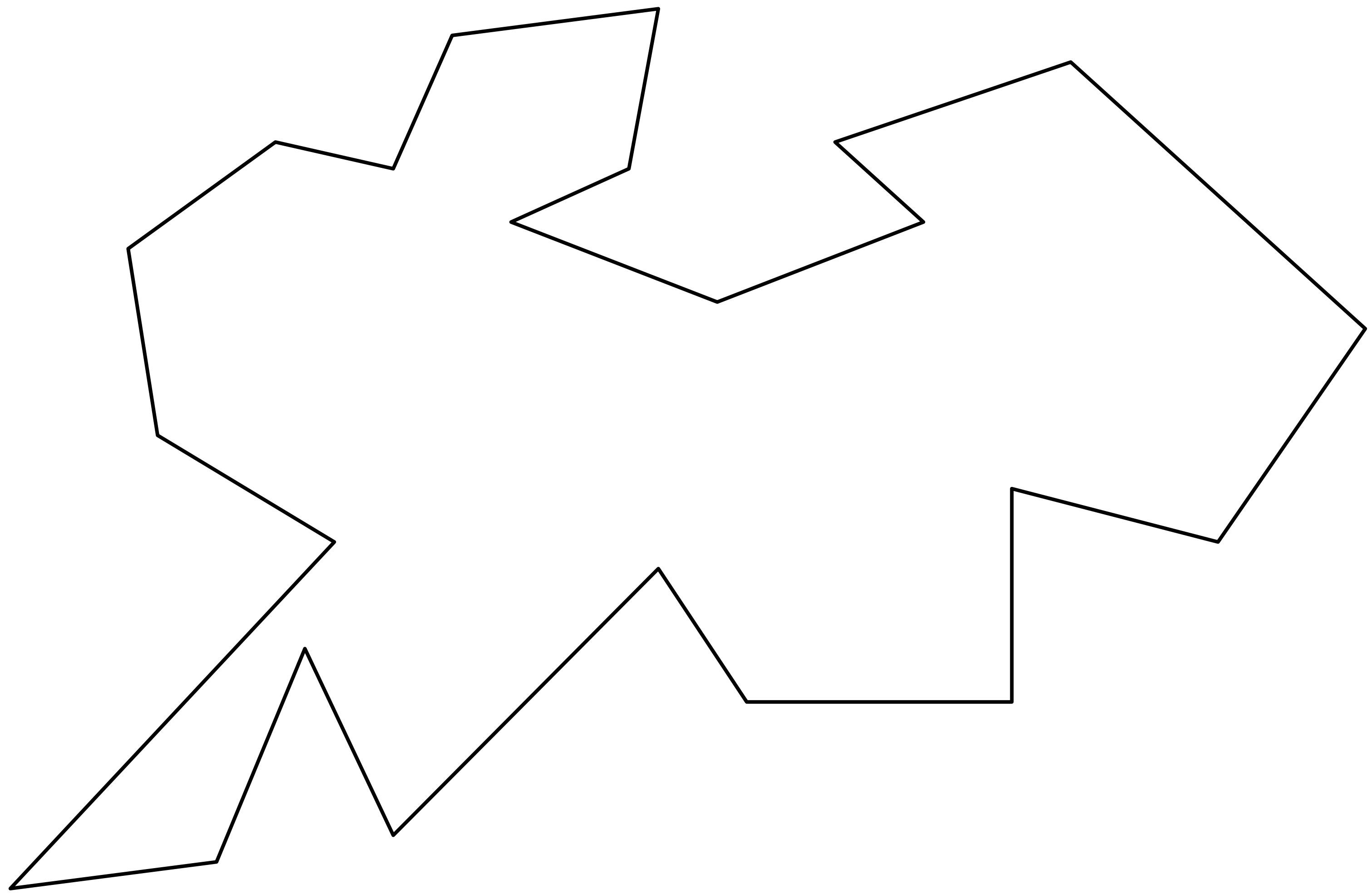
DIFFICULTIES

Which triangles do we use?



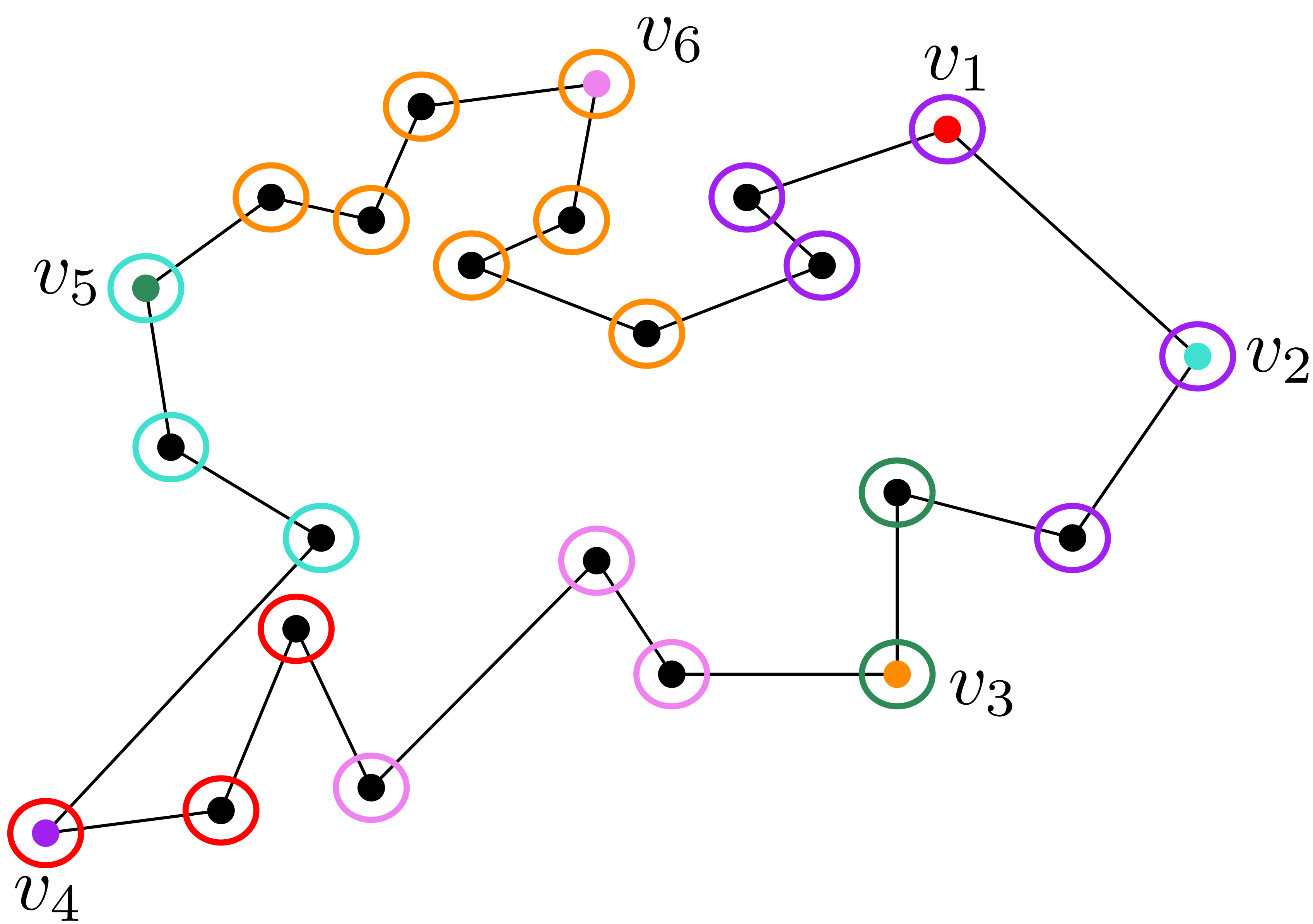
- A vertex can generate Θ(n) apexed triangles
  - Many triangles are irrelevant
  - Use geometric observations to avoid them

DETERMINING RELEVANT TRIANGLES



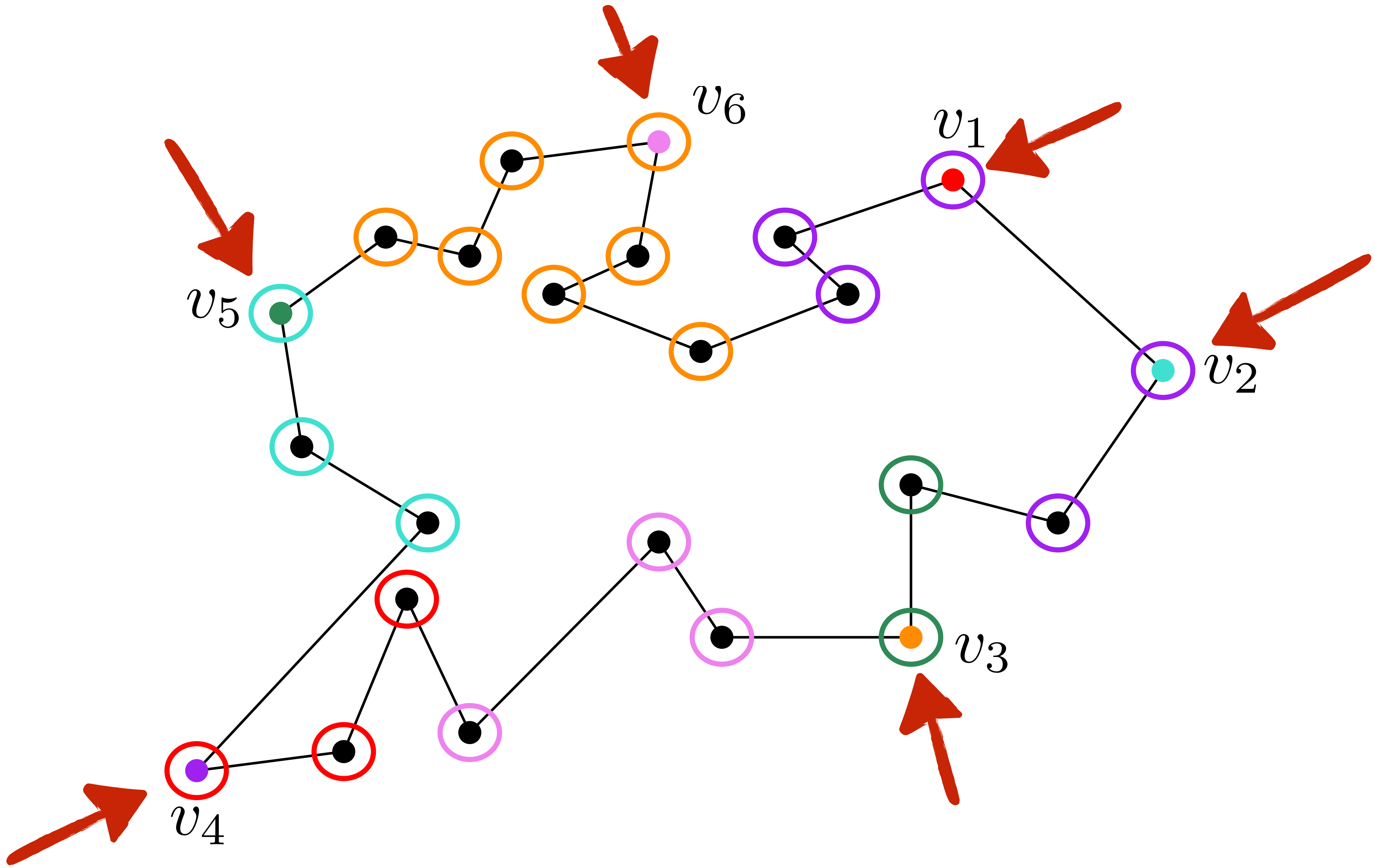
For each vertex compute its farthest neighbor

DETERMINING RELEVANT TRIANGLES



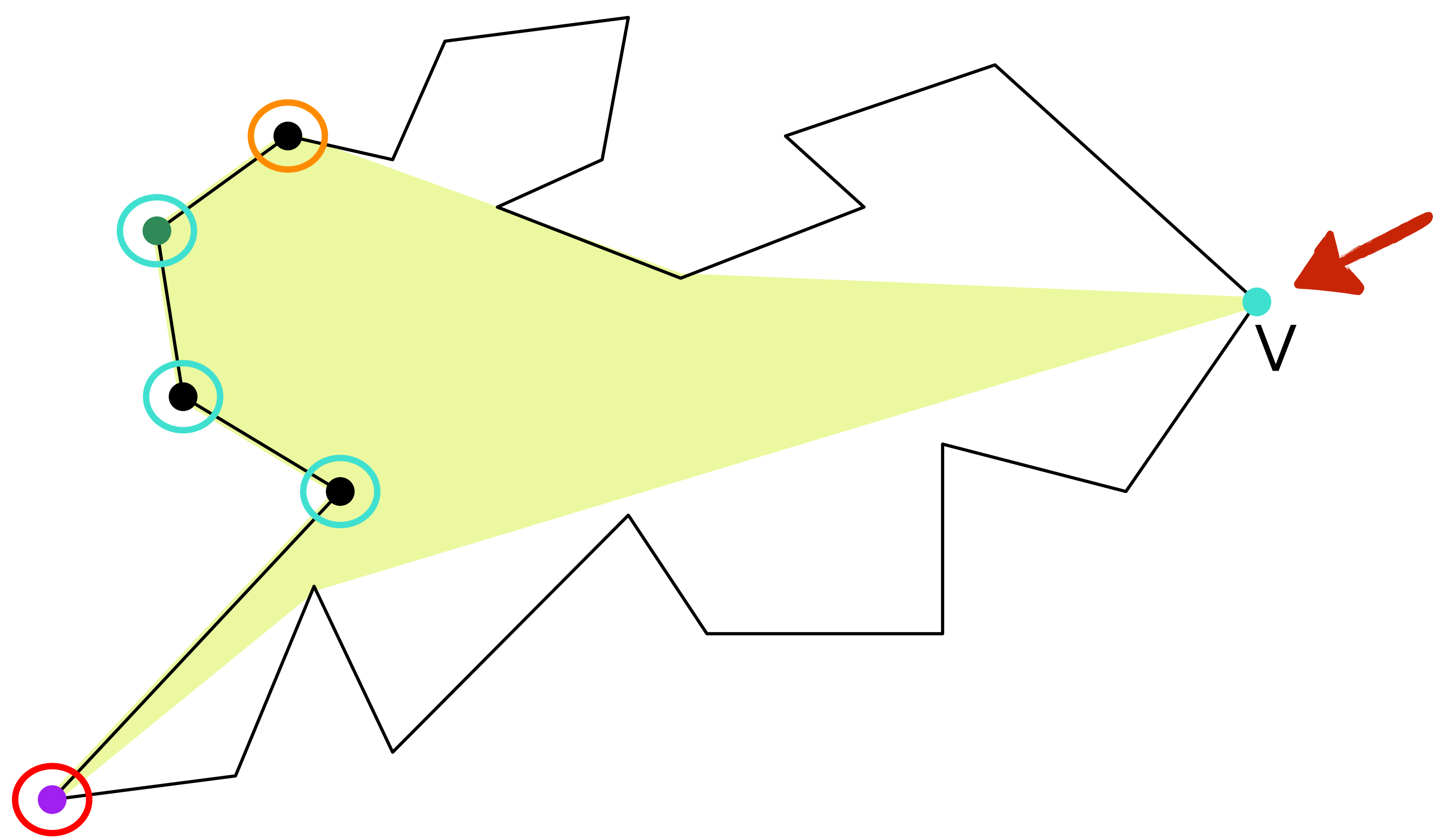
We can compute the farthest neighbor of each vertex in total O(n) time.

DETERMINING RELEVANT TRIANGLES



v is **marked** if v is farthest neighbor of some other vertex

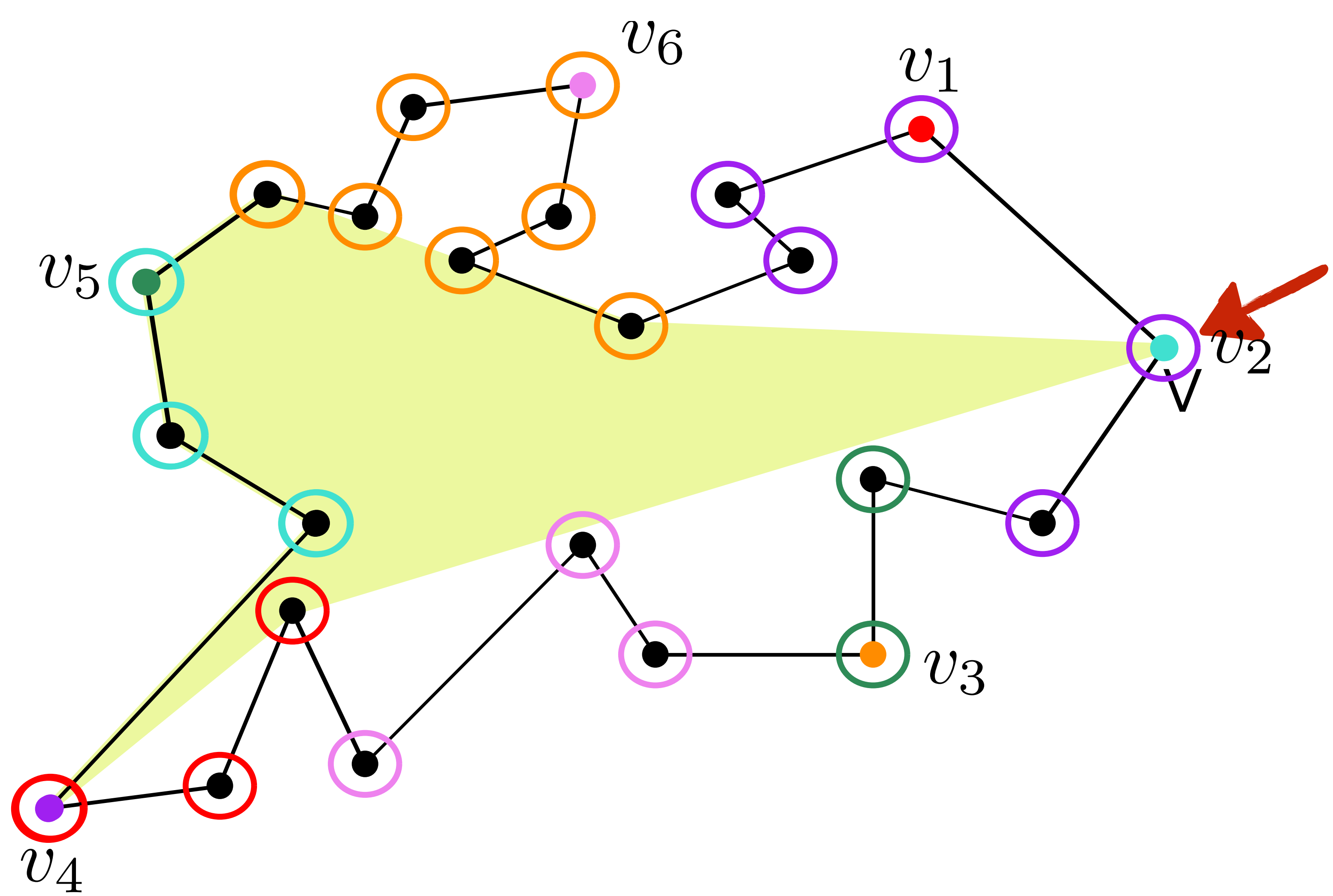
MARKED VERTICES



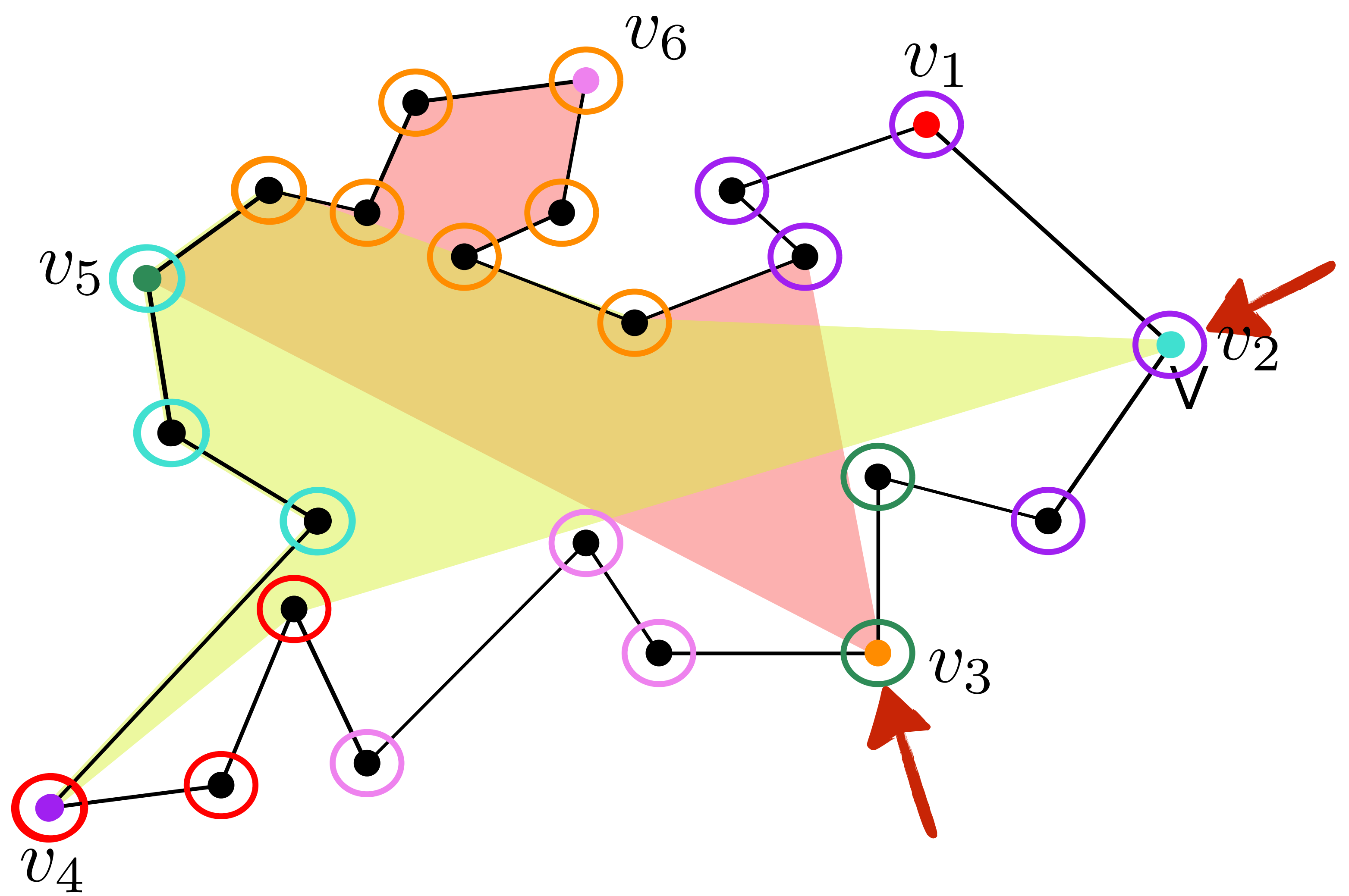
- For all marked vertex we construct its **funnel** F(v)
    - Funnel contains furthest vertices (and two additional neighbors)

Lemma FV(v)⊆F(v) for any marked vertex v

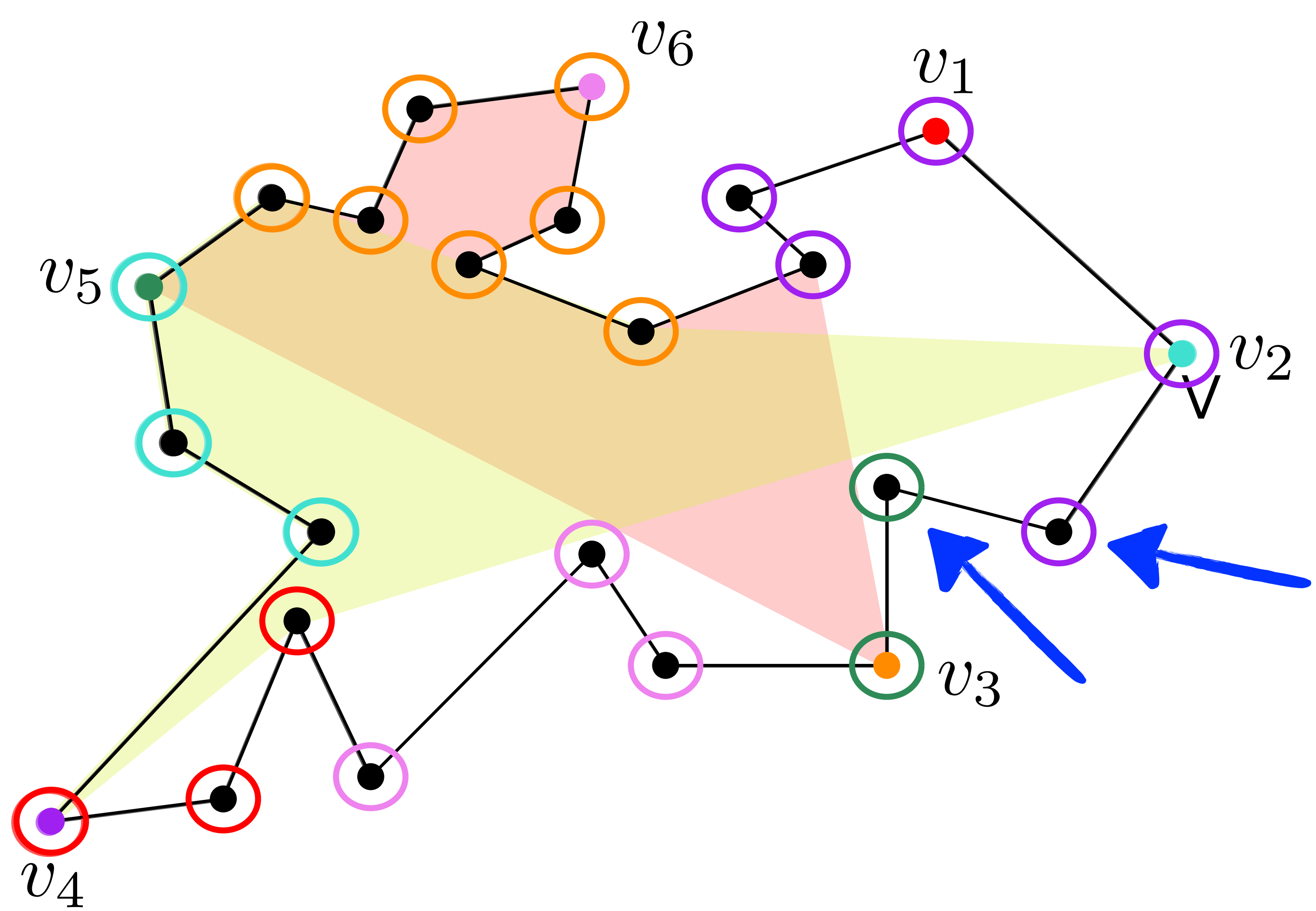
MARKED VERTICES



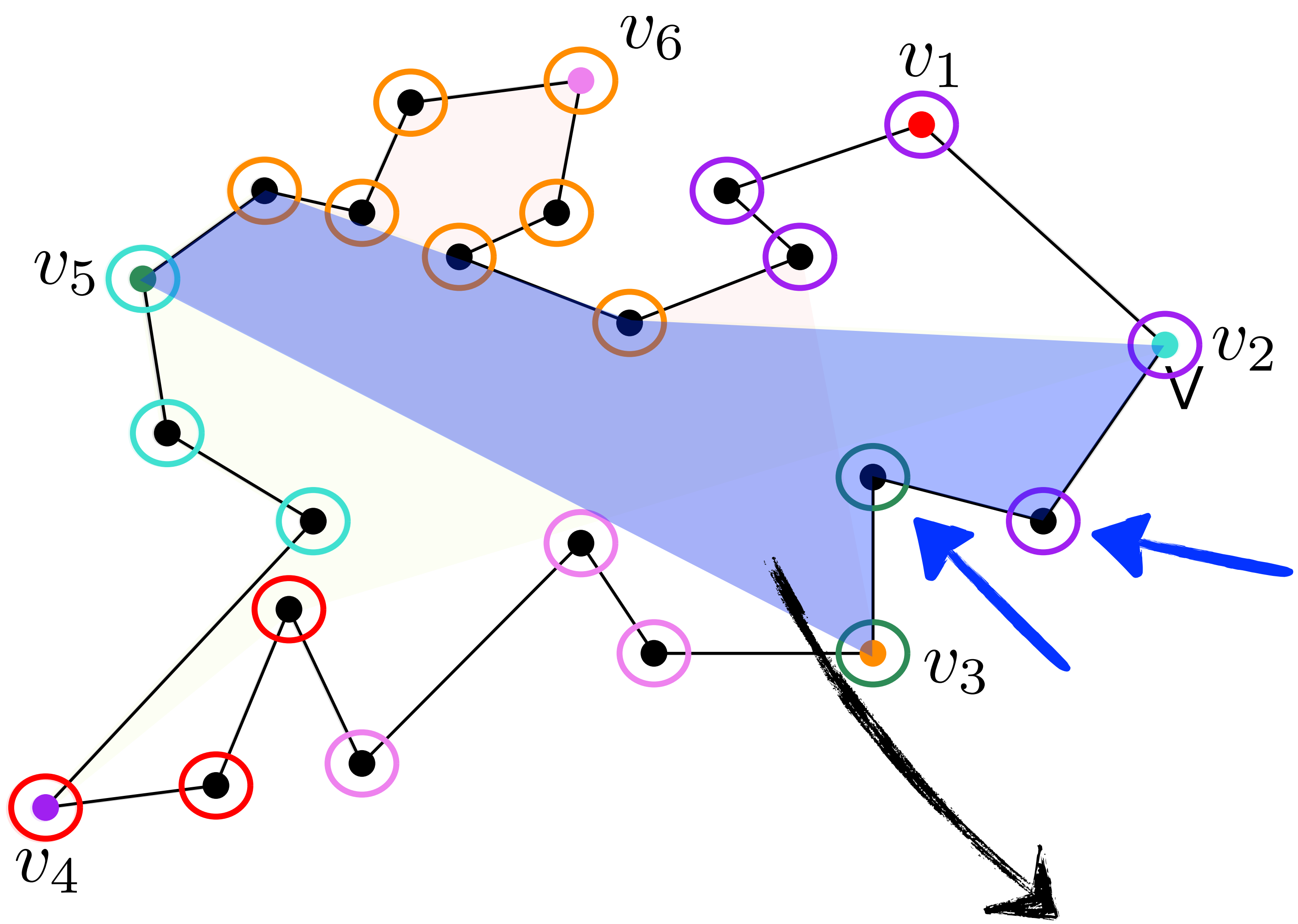
MARKED VERTICES



UNMARKED VERTICES



UNMARKED VERTICES



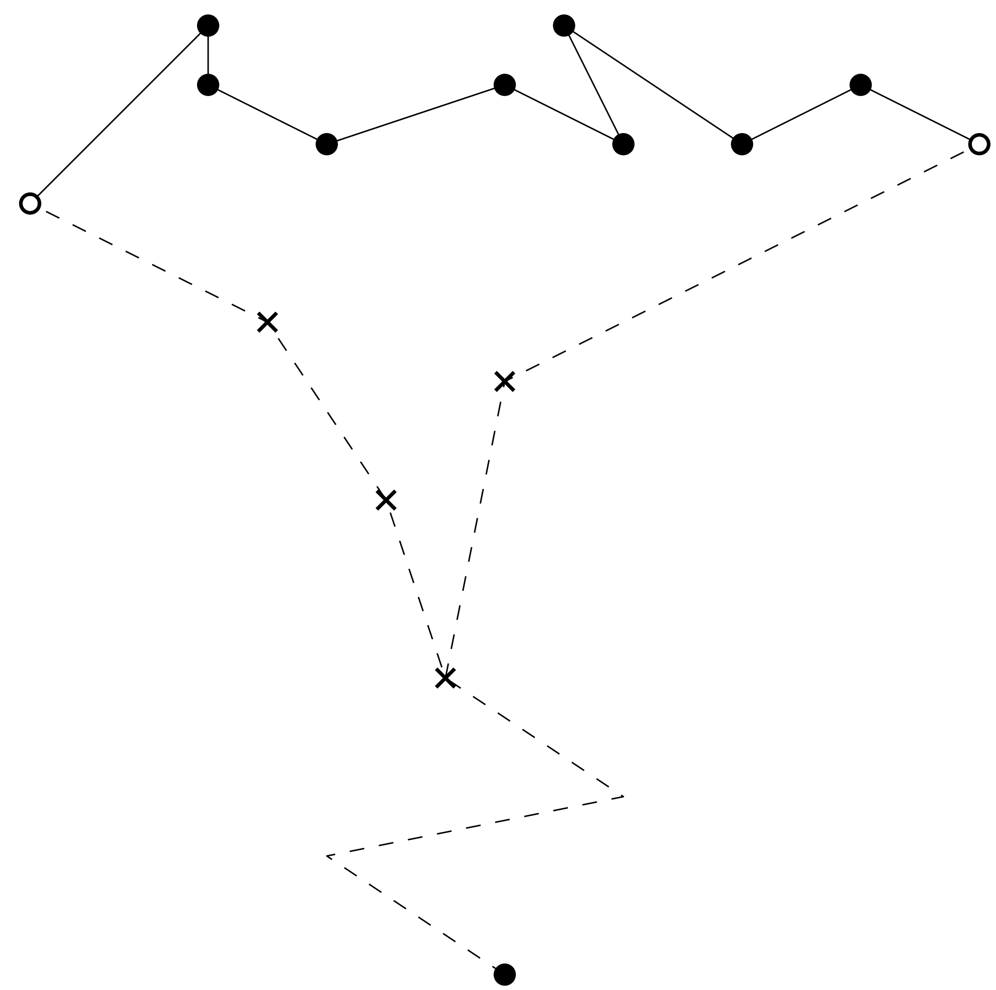
- Two adjacent u,v such that f(u)≠f(v) define an **Hourglass** H(v)
  - Upper chain may contain unmarked vertices

PROPERTIES OF THE DECOMPOSITION

Lemma This decomposition is great!

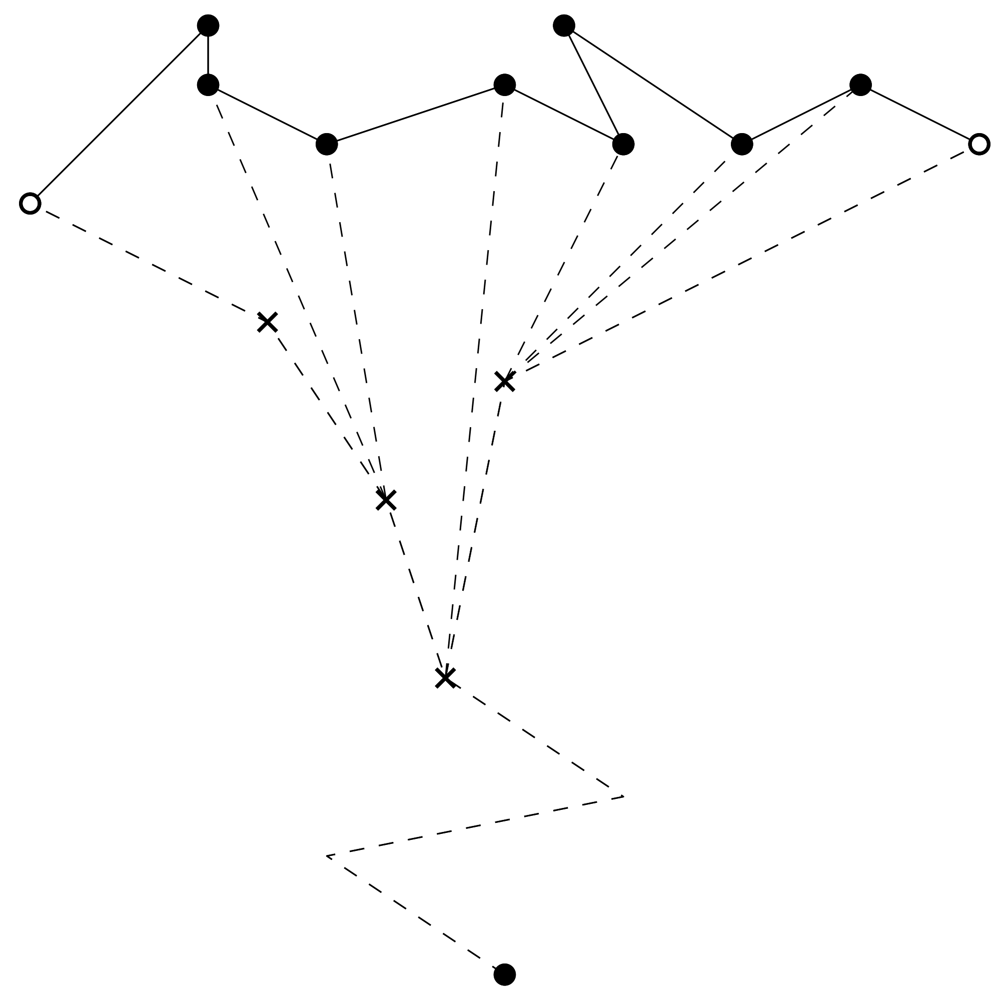
- All hourglasses and funnels have overall linear complexity
- Computed in linear time
  - Linear space
- Covers P
- Each point and its farthest neighbor are in an hourglass/funnel

DECOMPOSING A FUNNEL



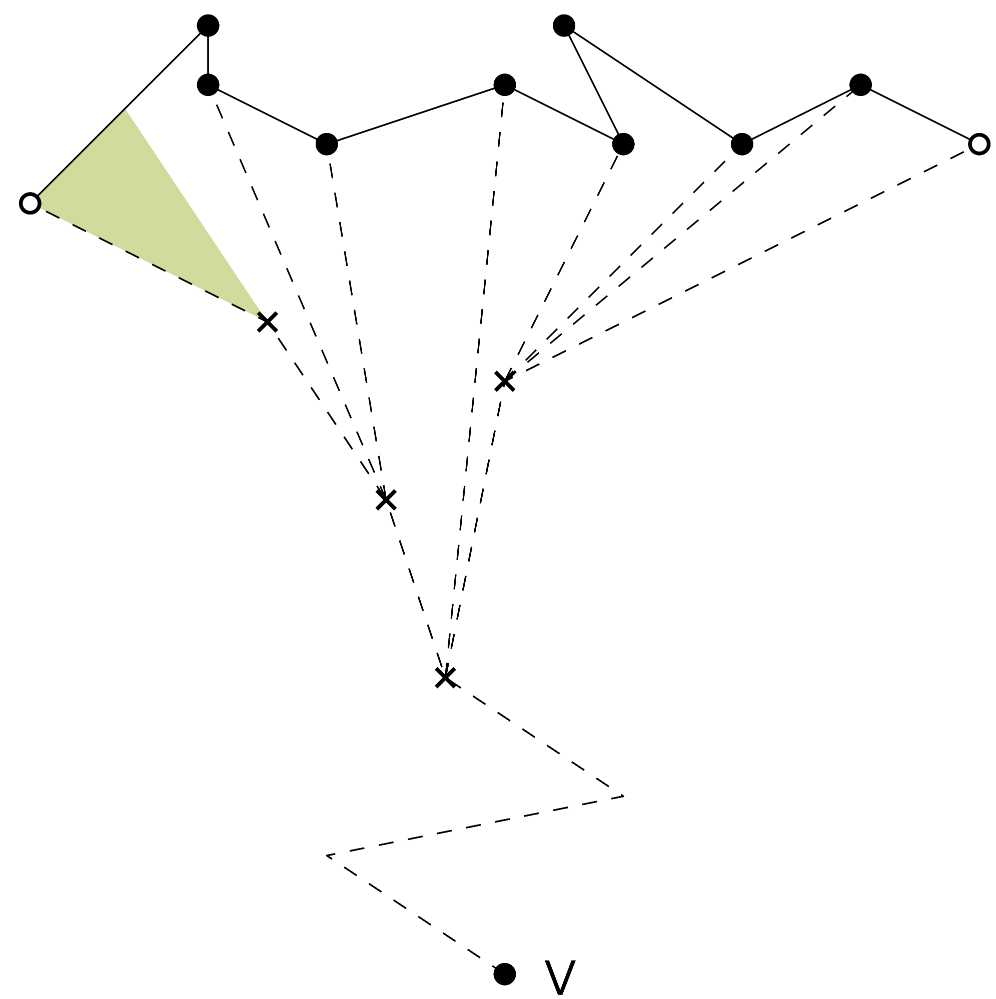
- Vertices (except endpoints) share farthest neighbor
  - Want to encode distance to it

DECOMPOSING A FUNNEL



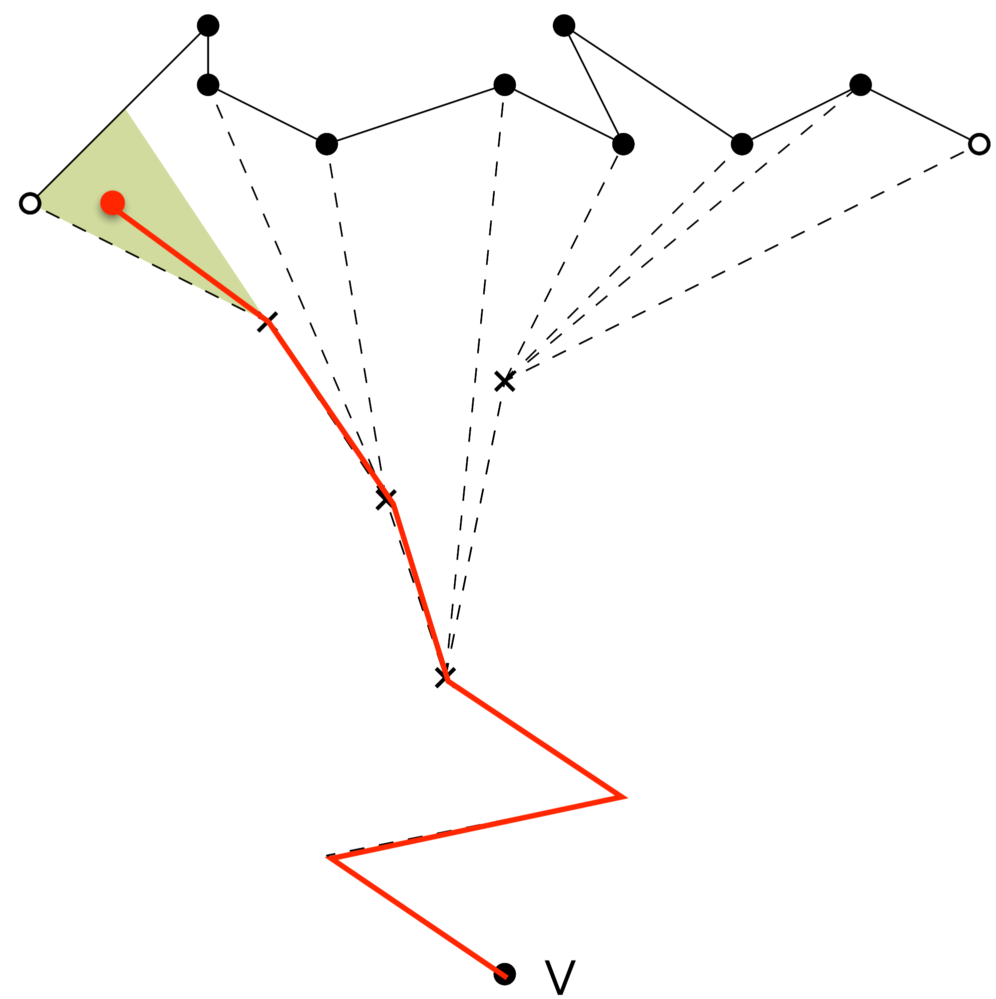
- Look for topologically equivalent shortest paths
  - Similar paths fall in the same apexed triangle

DECOMPOSING A FUNNEL



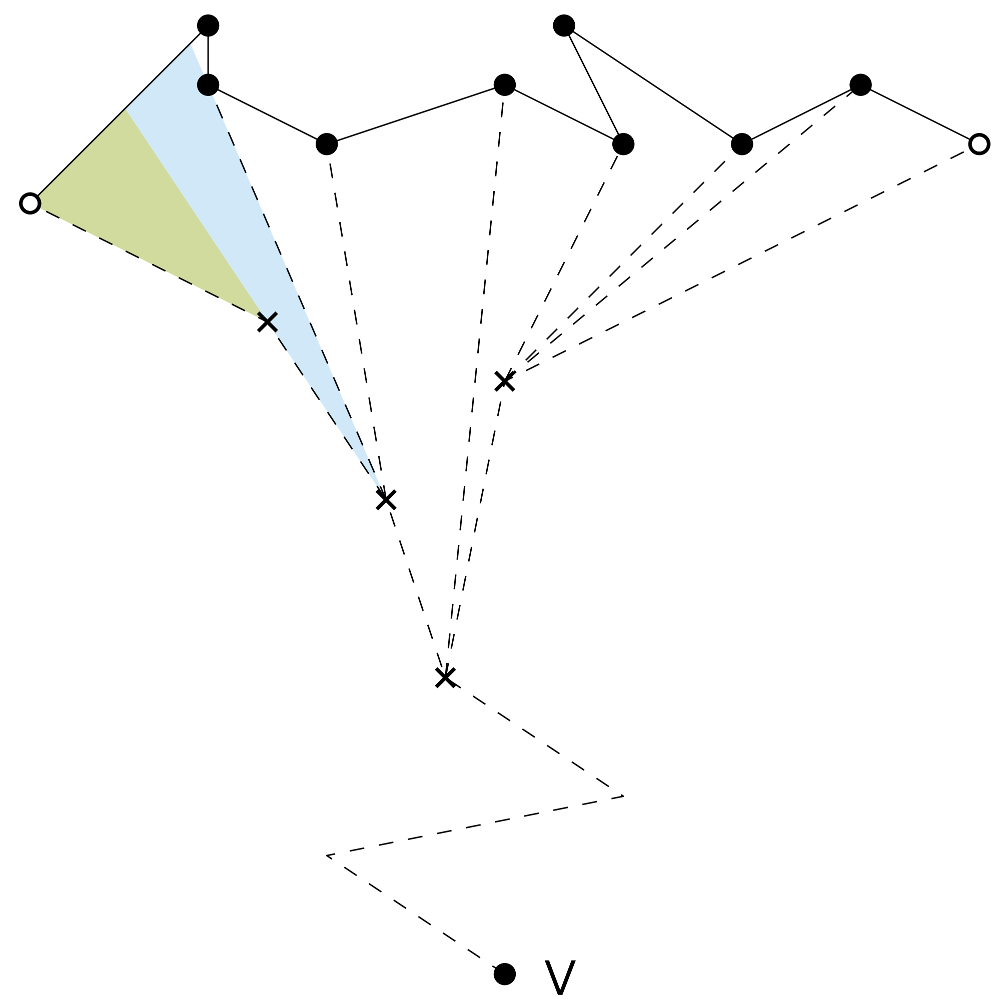
- Look for topologically equivalent shortest paths
  - Similar paths fall in the same apexed triangle

DECOMPOSING A FUNNEL



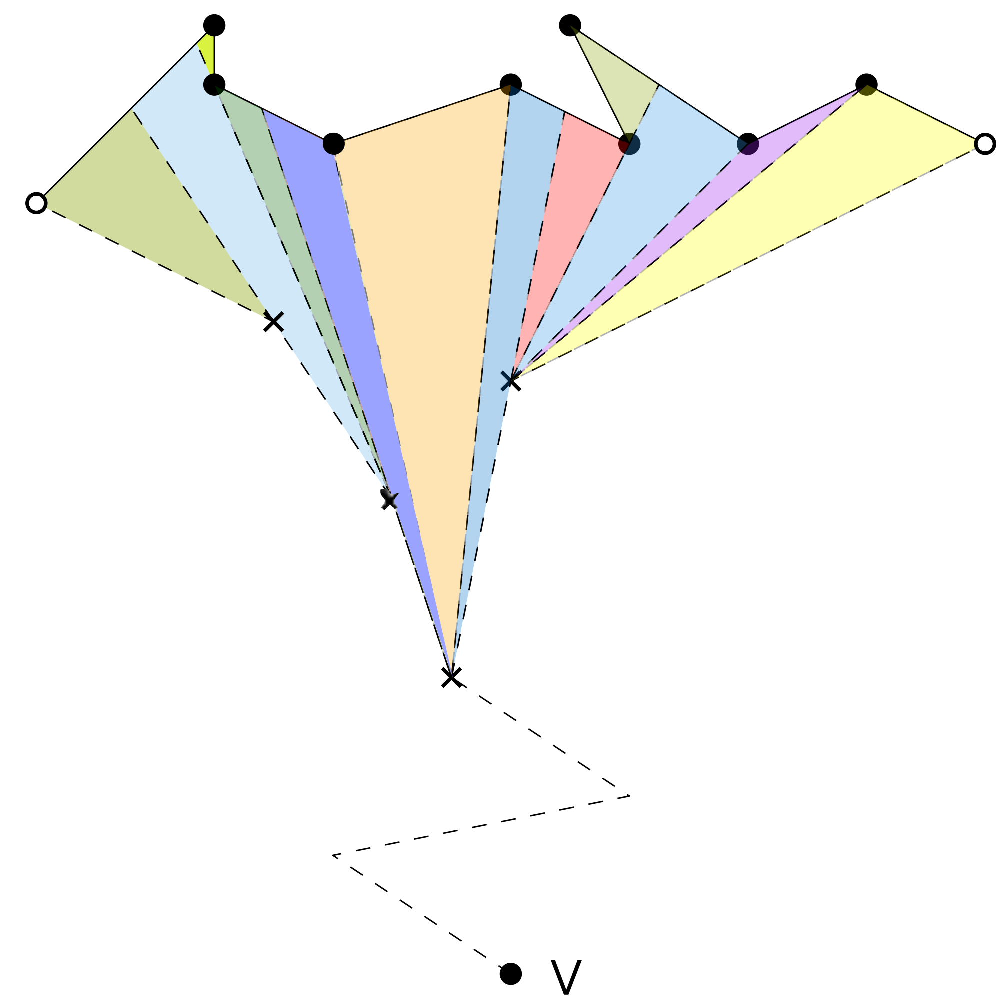
- Look for topologically equivalent shortest paths
  - Similar paths fall in the same apexed triangle

DECOMPOSING A FUNNEL



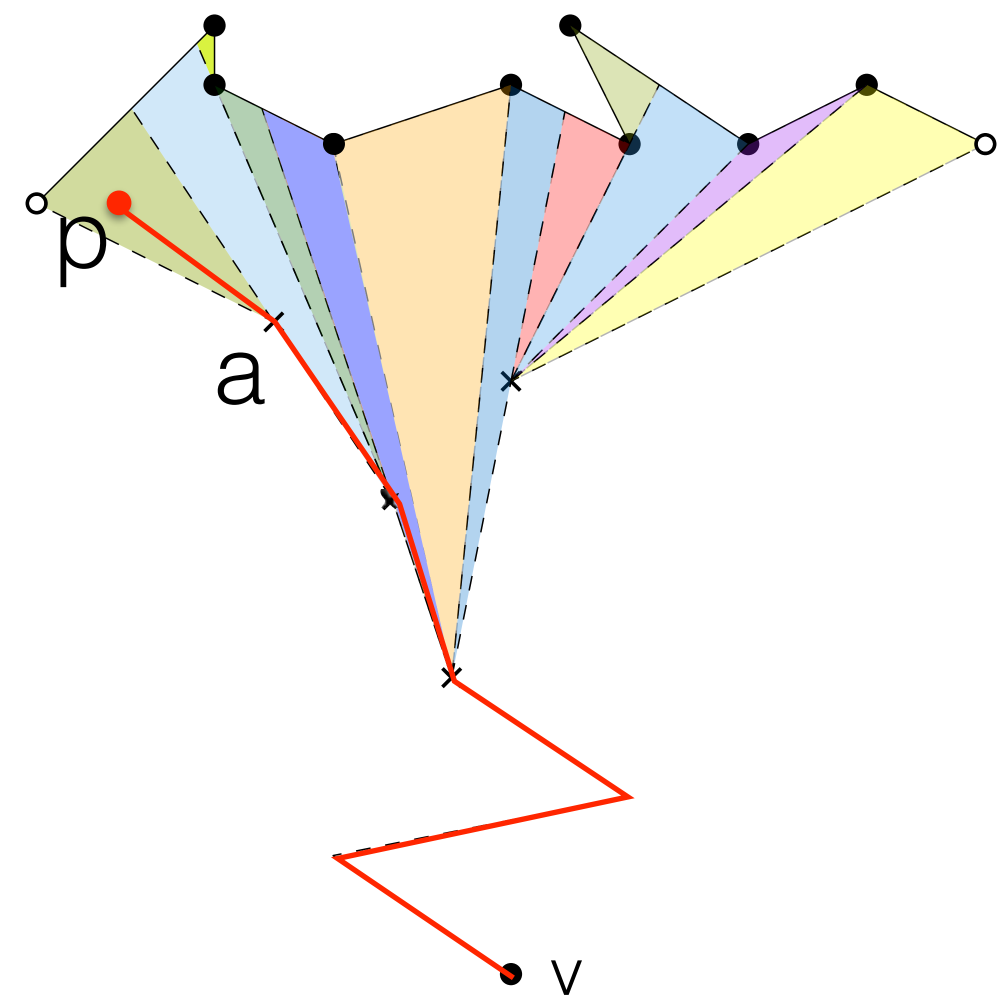
- Look for topologically equivalent shortest paths
  - Similar paths fall in the same apexed triangle

DECOMPOSING A FUNNEL



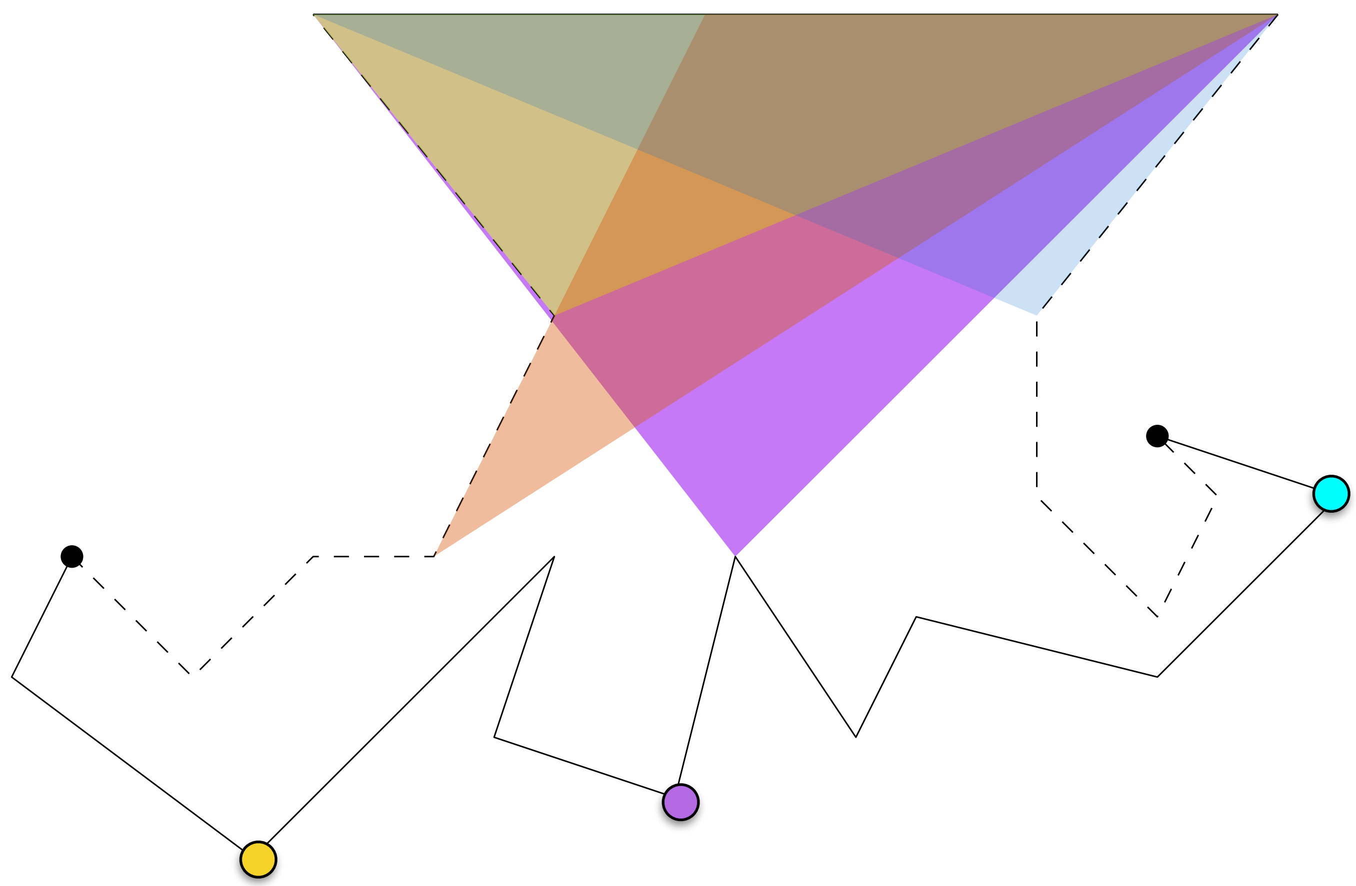
- Look for topologically equivalent shortest paths
  - Similar paths fall in the same apexed triangle

DECOMPOSING A FUNNEL



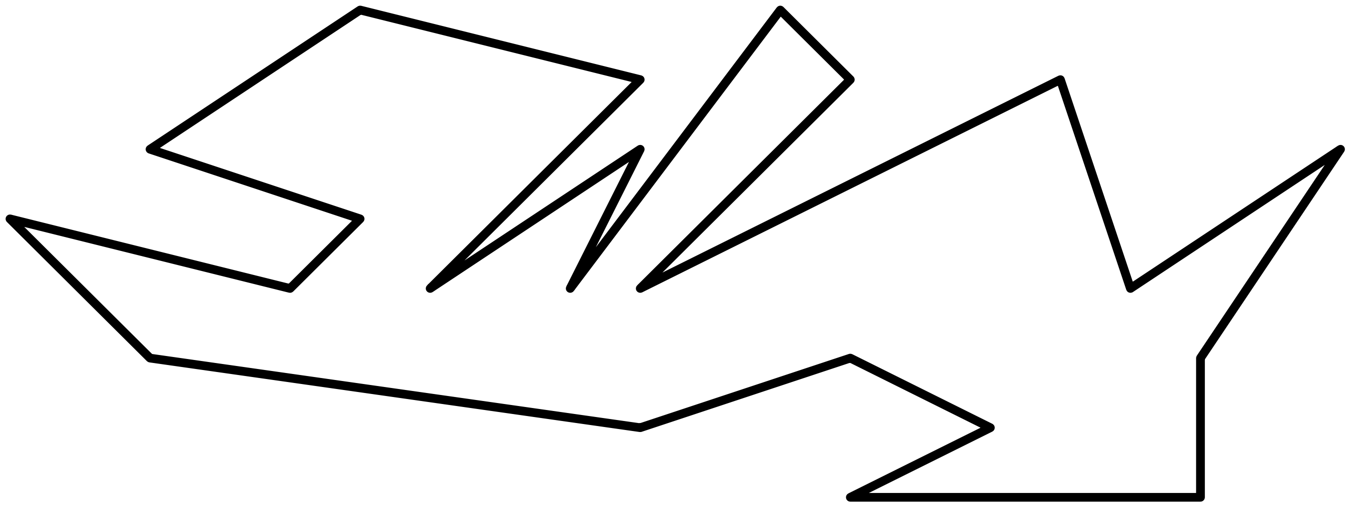
- For each triangle we define an **apexed function**
  - f(p)=d(p,a)+d(a,v)

HOURGLASSES



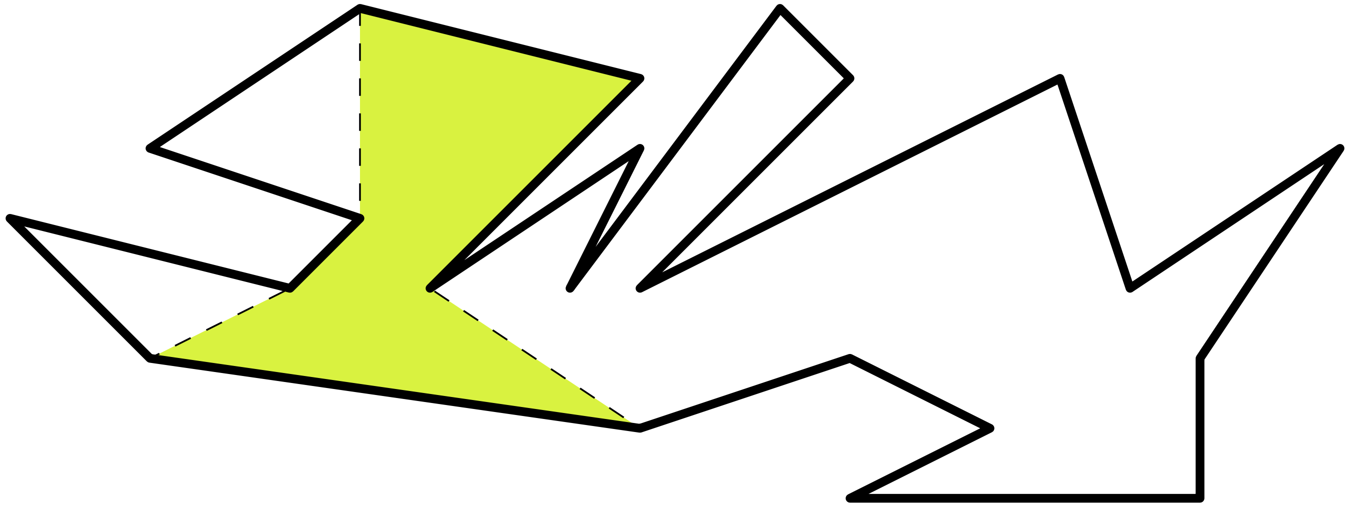
- We decompose into triangles (similar to funnel)
- The number of triangles is linear on the size of the hourglass

LET'S RECAP



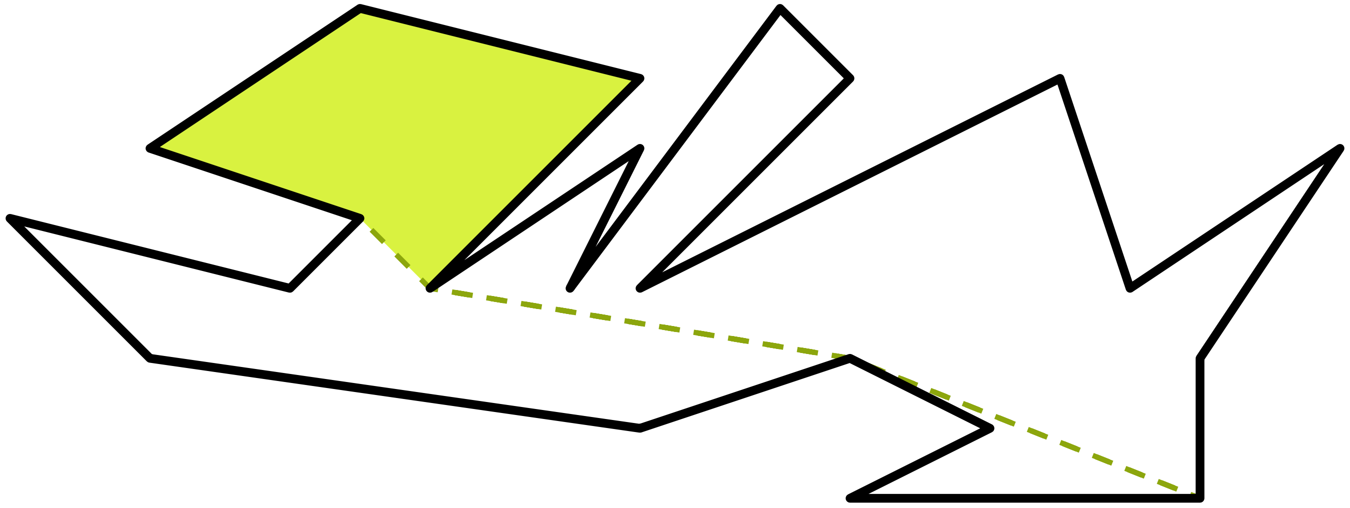
- Two step decomposition.
  - Compute a covering of P
  - Further cover regions with triangles

LET'S RECAP



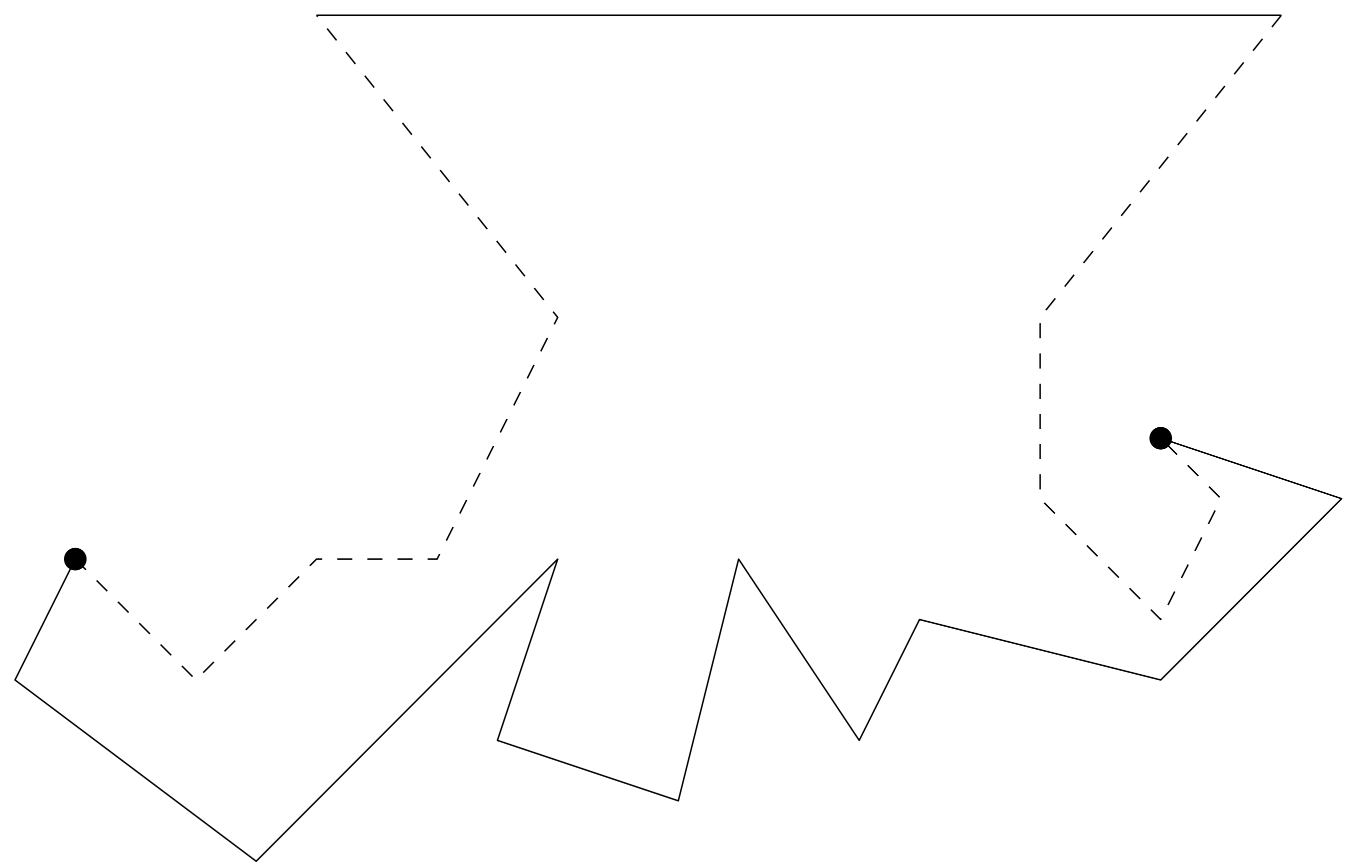
- Two step decomposition
  - Compute a covering of P
  - Further cover regions with triangles

LET'S RECAP



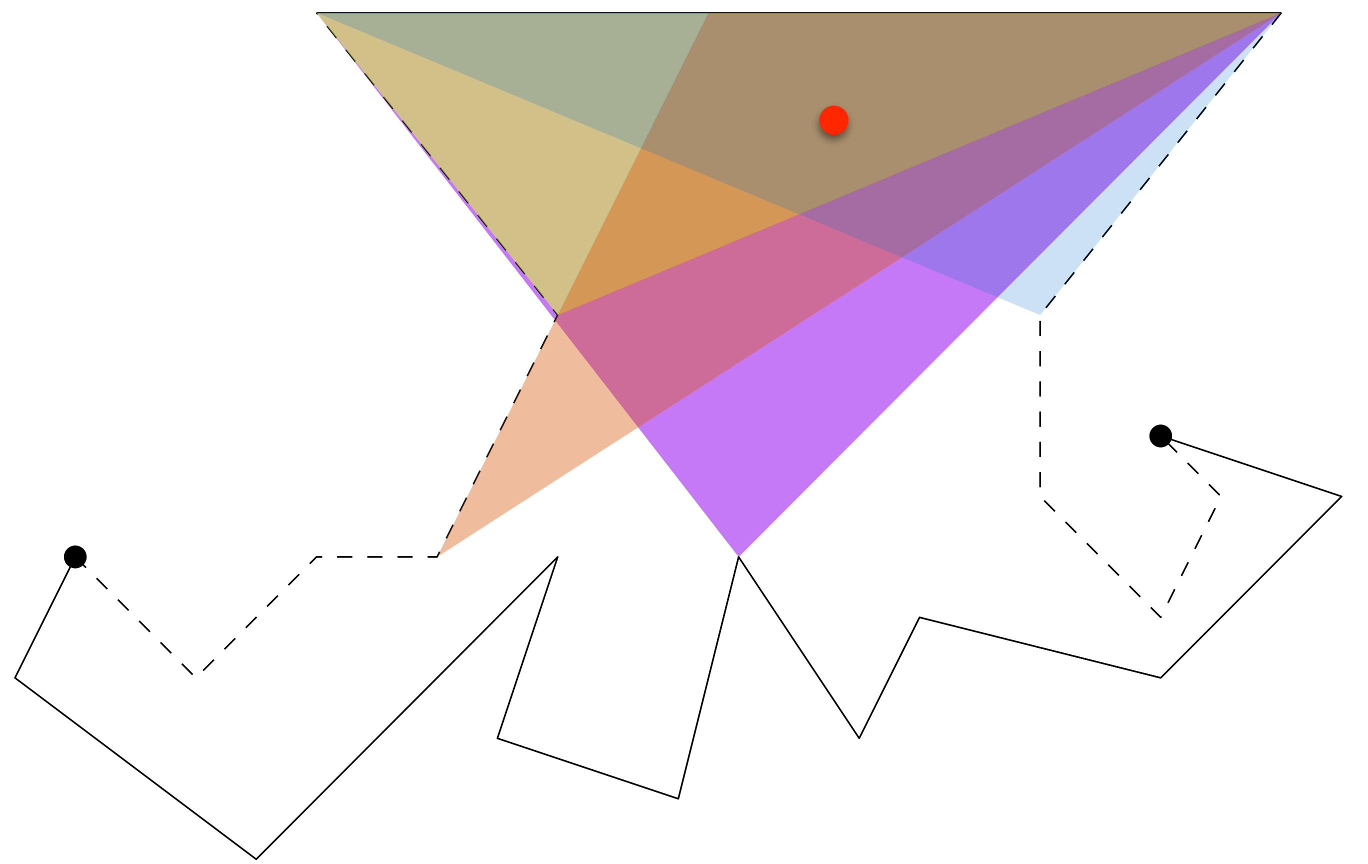
- Two step decomposition
  - Compute a covering of P
  - Further cover regions with triangles

OUR APPROACH



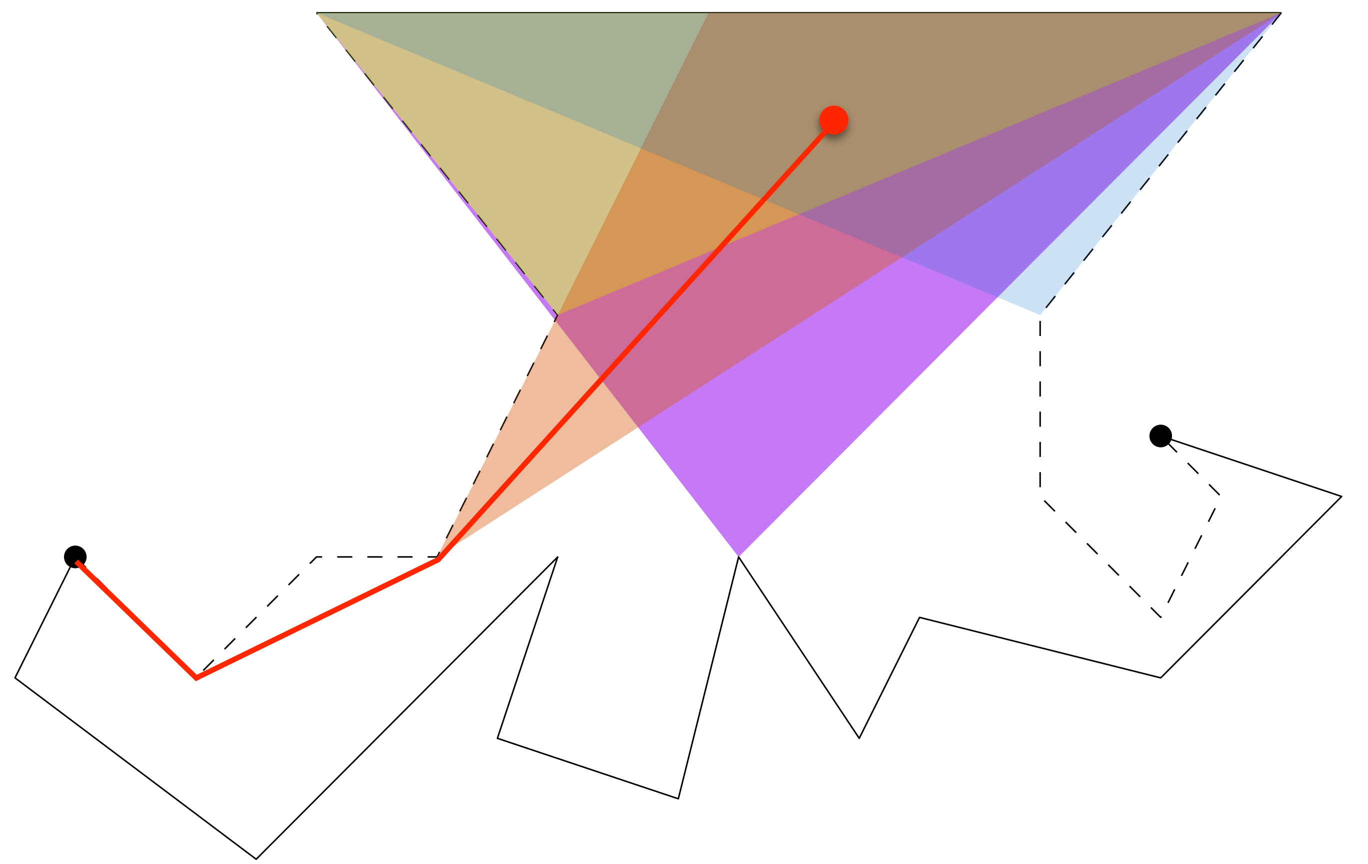
- Overall O(n) triangles
- For any point, there is a triangle containing him
    - associated vertex is a farthest neighbor

LET'S RECAP



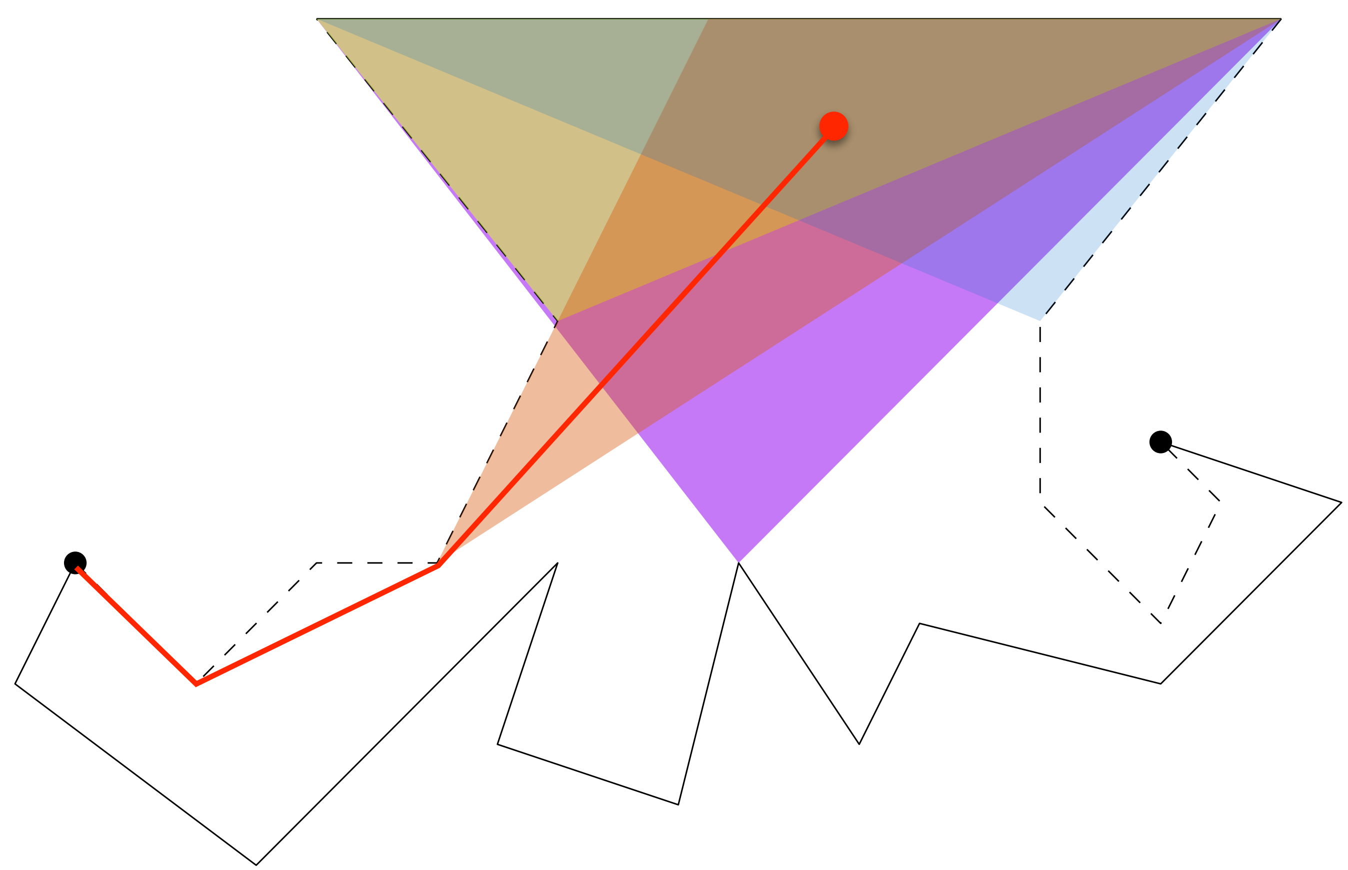
- Overall O(n) triangles
- For any point, there is a triangle containing him
    - associated vertex is a farthest neighbor

LET'S RECAP



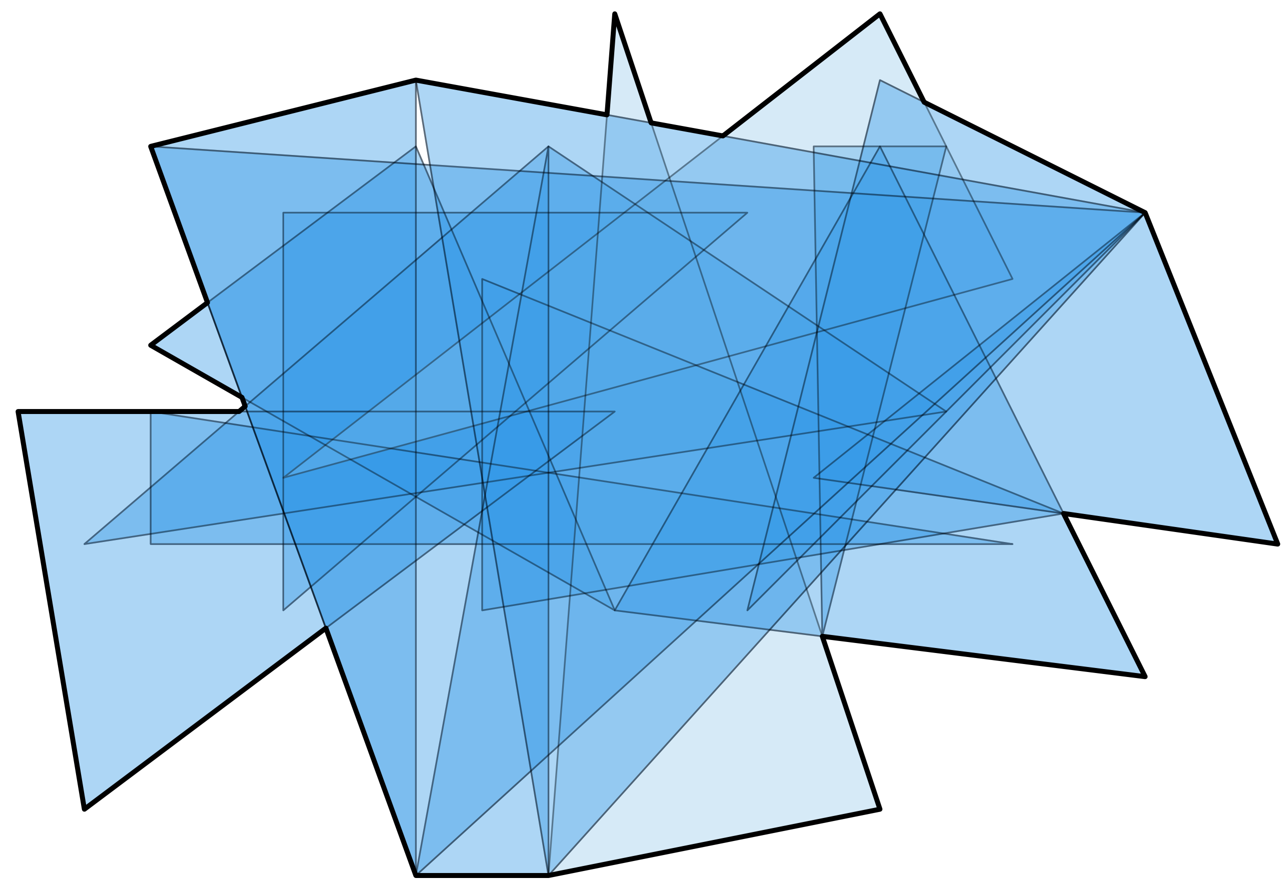
- Overall O(n) triangles
- For any point, there is a triangle containing him
  - associated vertex is a farthest neighbor

LET'S RECAP



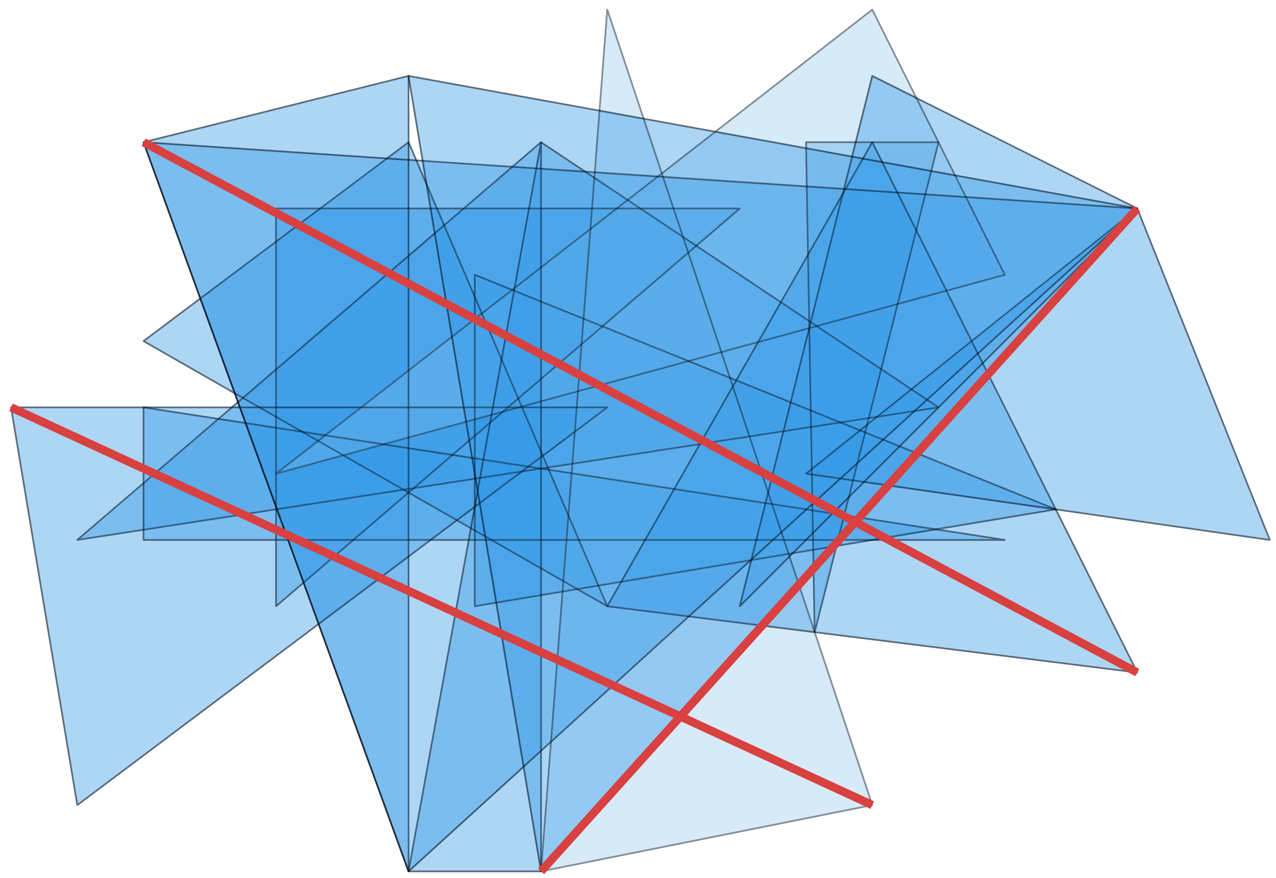
We ignore P and use **prune&search** on triangles

PRUNE AND SEARCH



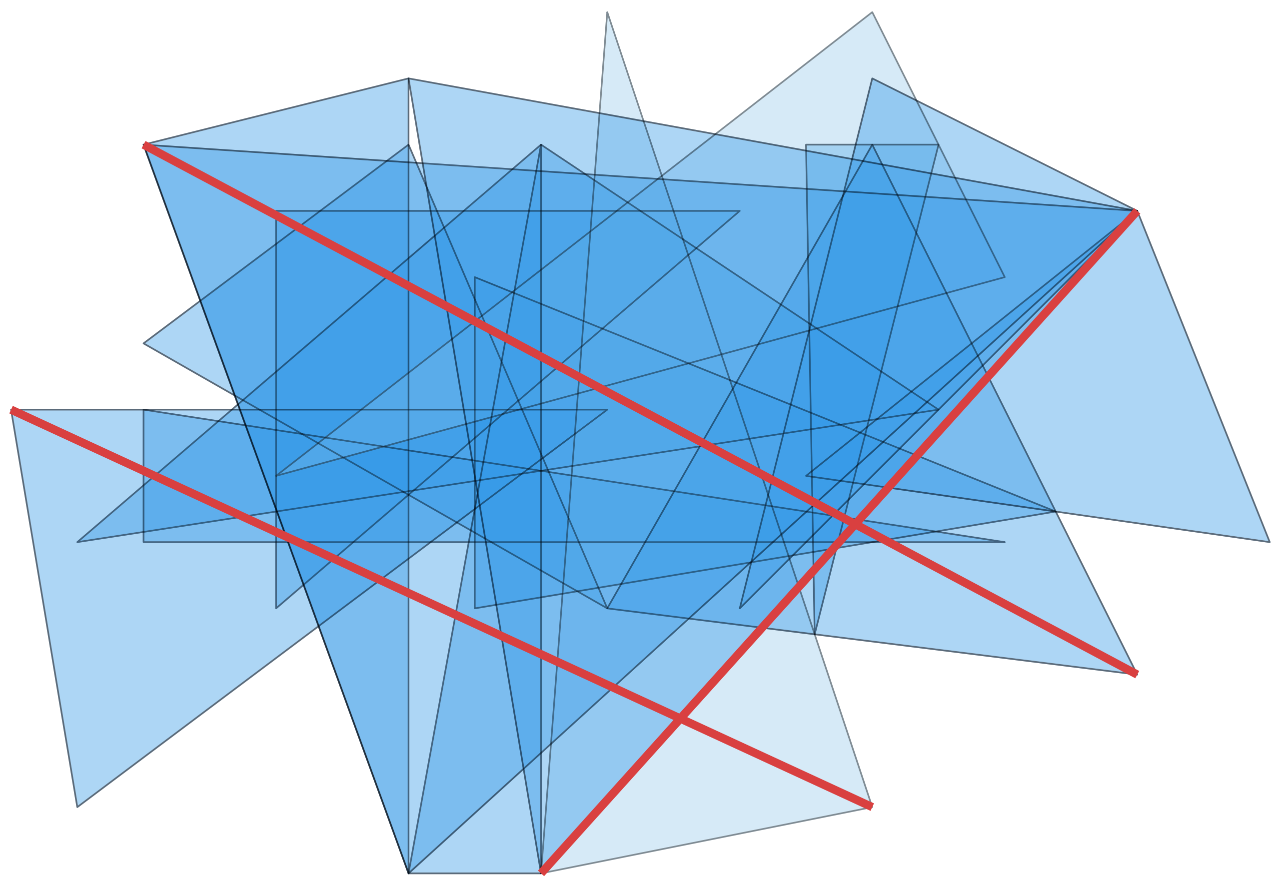
- Covered P with O(n) triangles
  - Each triangle associated with a distance function
- We ignore P and work on the triangles

CUTTINGS



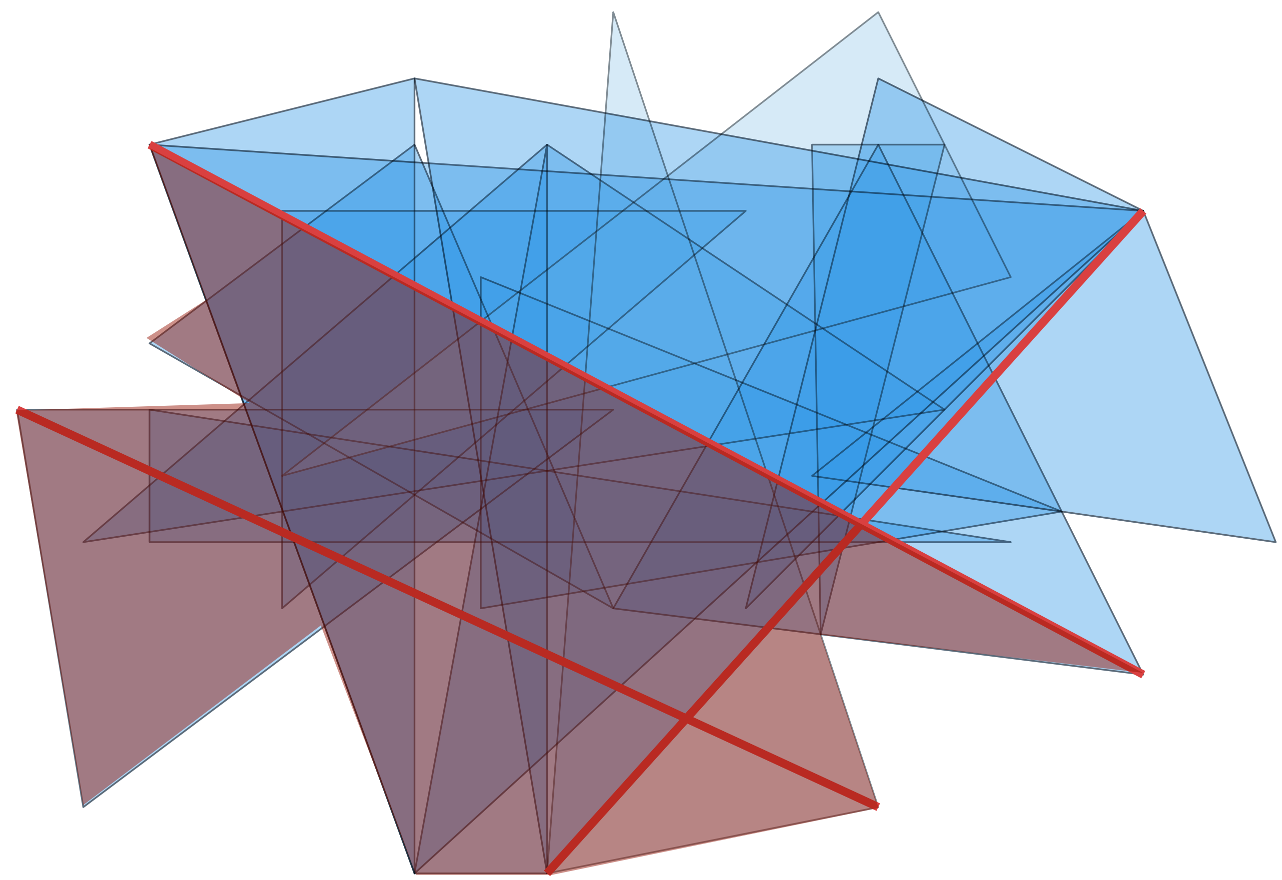
- We construct a cutting of the triangles chords (sides)
  - Partition into O(1) cells
  - Each cell intersecting the boundary of **at most** εn triangles

PRUNE AND SEARCH



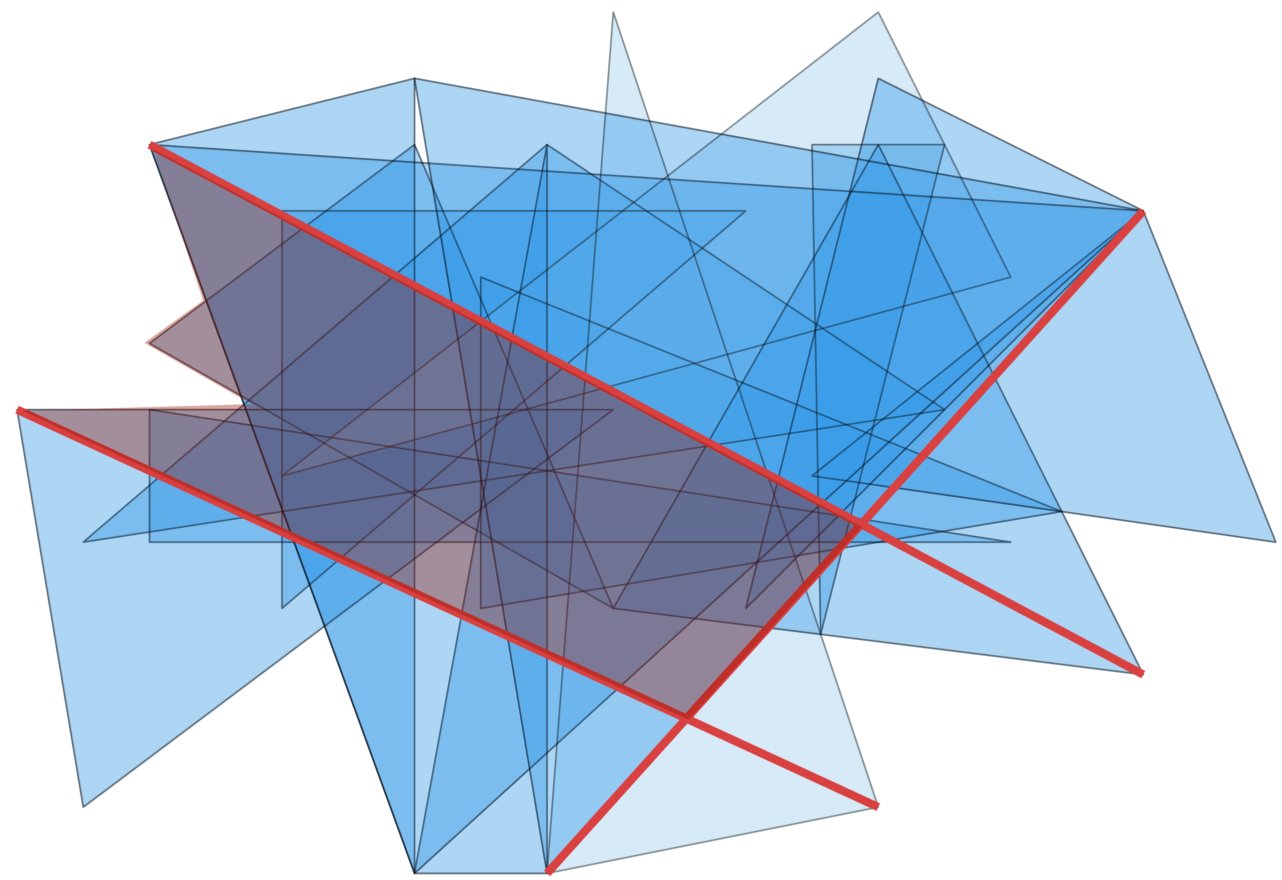
- Use the chord oracle of Pollack-Sharir-Rote
  - Reduce the problem to a sub-cell
  - We discard a fraction of the triangles and iterate

PRUNE AND SEARCH



- Use the chord oracle of Pollack-Sharir-Rote
  - Reduce the problem to a sub-cell
  - We discard a fraction of the triangles and iterate

PRUNE AND SEARCH



- Use the chord oracle of Pollack-Sharir-Rote
  - Reduce the problem to a sub-cell
  - We discard a fraction of the triangles and iterate

SUMMARY

- Cover P with funnels/hourglasses
  - Further cover the regions with triangles
  - Ignore P and apply prune and search to triangles
- Finish a triangular region
  - Can be solved in linear time using **cuttings in R^3**

SUMMARY

- Cover P with funnels/hourglasses
  - Further cover the regions with triangles
  - Ignore P and apply prune and search to triangles
- Finish a triangular region
  - Can be solved in linear time using **cuttings in R^3**

Theorem We can compute the center of a simple polygon in linear time

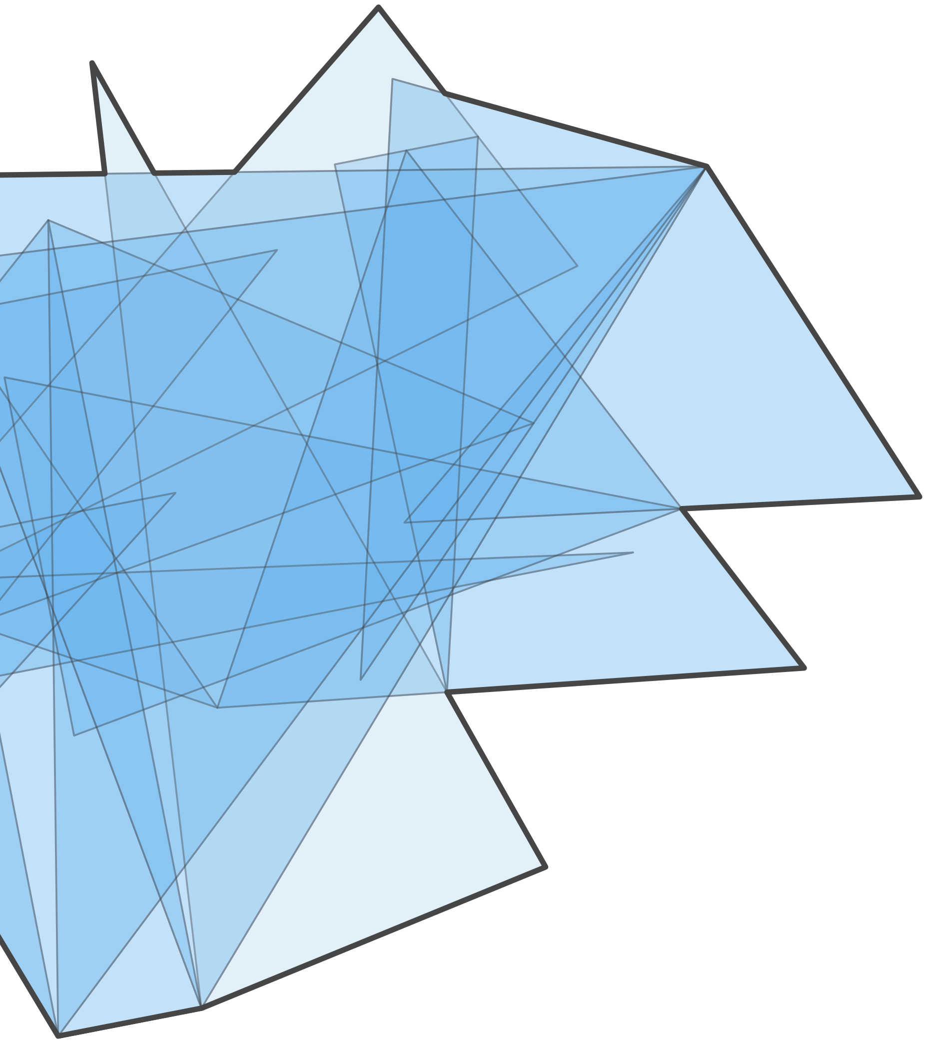
OPEN QUESTIONS

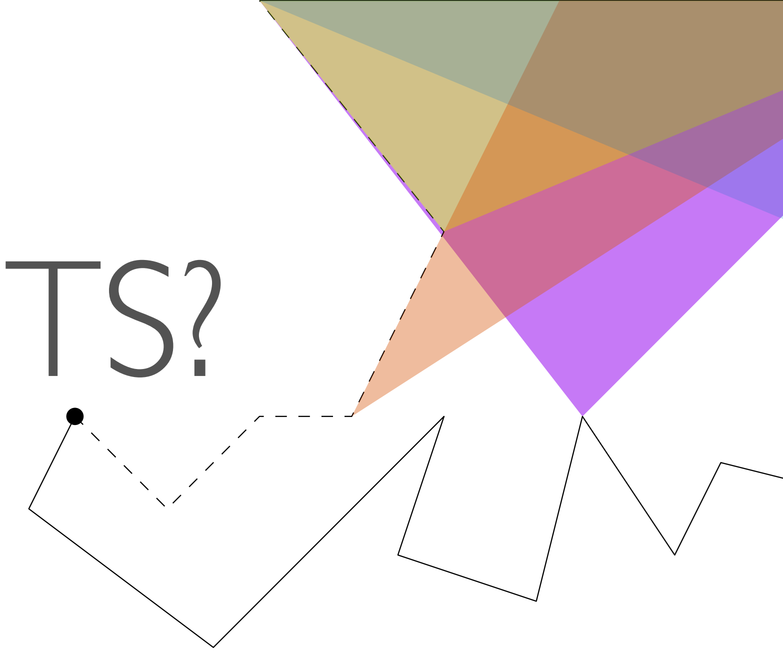
- Can we compute the center of a set of sites S in the interior of the polygon in linear time?

Polygon	Diameter	Center
Simple	O(n) [HS'93]	**O(n) [ABBCKO'15]**
General	$O(n^{7.73})$ [BKO'10]	$O(n^{12+\varepsilon})$ [BKO'14]

- For the case of polygonal domains (polygons with holes), the running times are polynomial but unfeasible. Can we improve them? What about lower bounds?

QUESTIONS? COMMENTS?





Thanks for your attention!

Summary

Polygon	Diameter	Radius / Center	Farthest Voronoi	Closest Voronoi
Simple	O(n) [HS'93]	**O(n) [ABBCKO'15]**	O((n+m) log (n+m)) [AFG'93]	O((n+m) log (n+m)) [A'89]