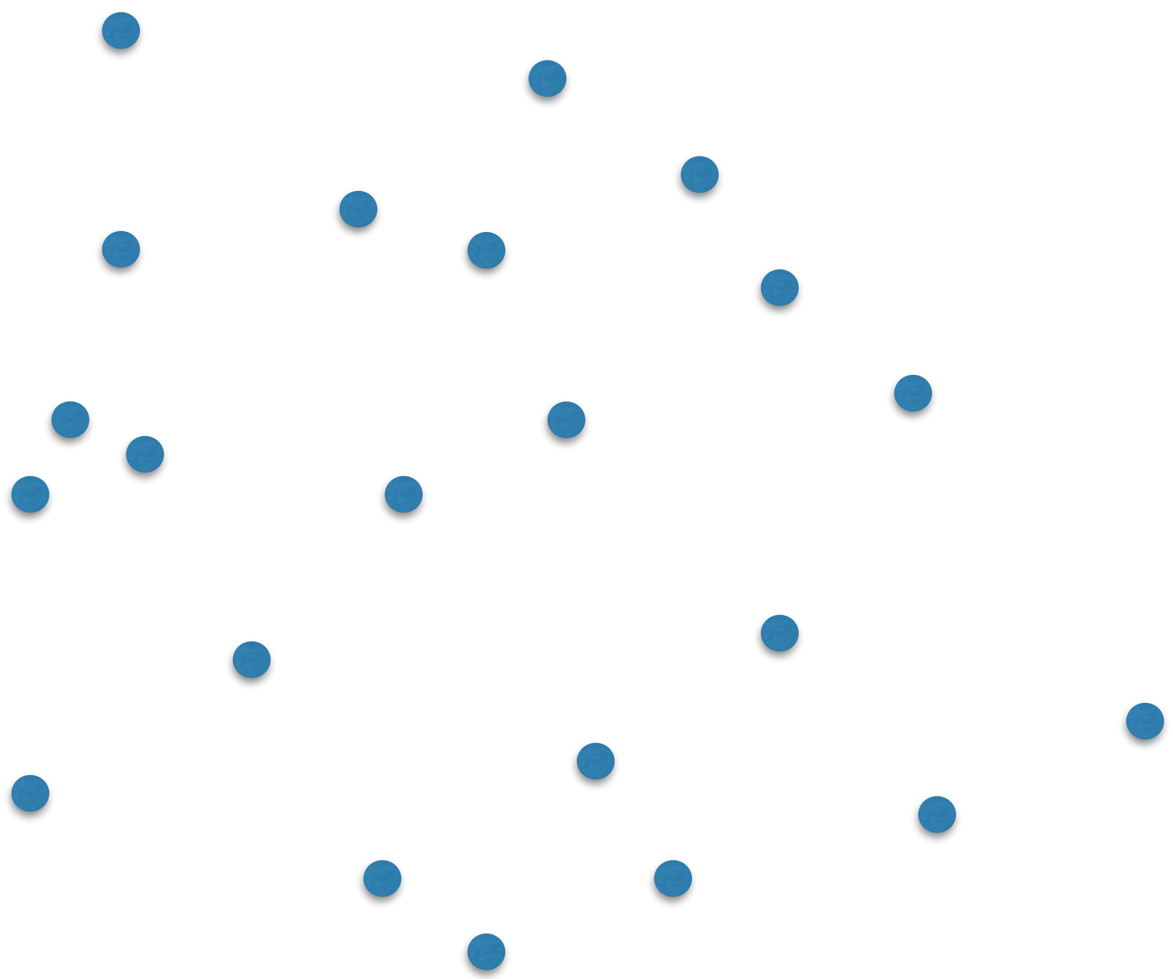
Computational Geometry

(csci3250)

Laura Toma
Spring 2017
Bowdoin College

Introduction

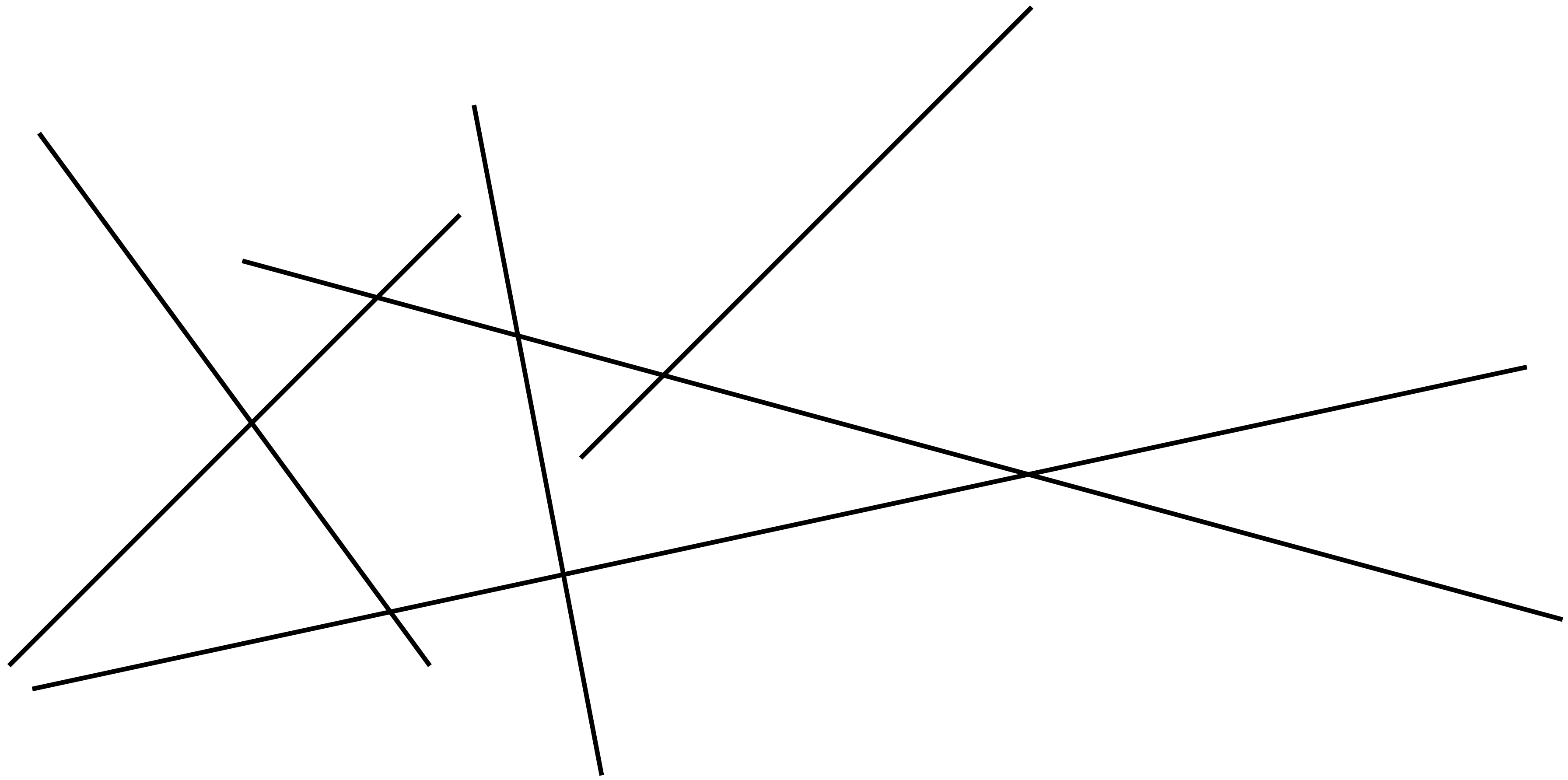
- CG deals with algorithms for geometric data



points

Introduction

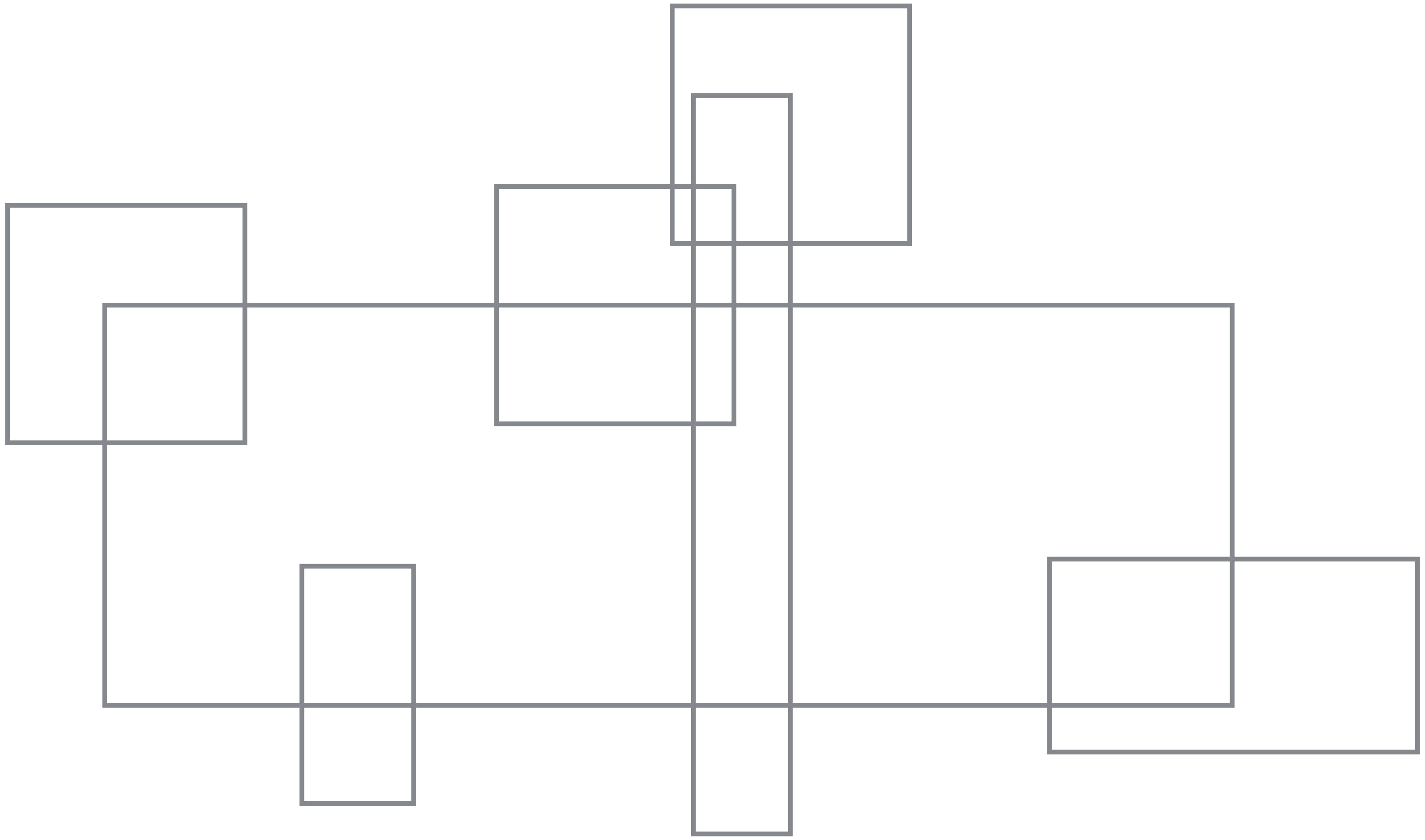
- CG deals with algorithms for geometric data



lines and line segments

Introduction

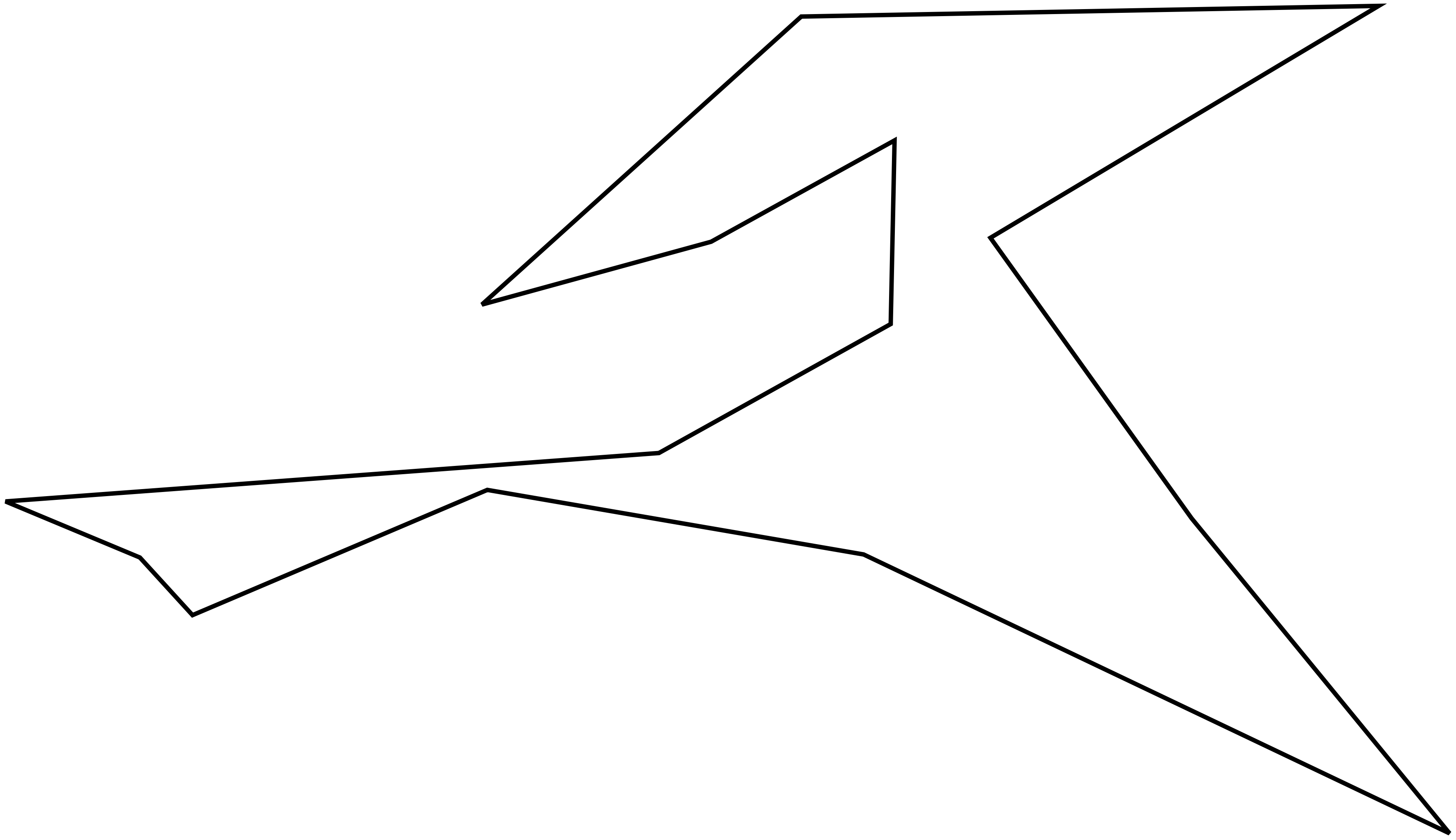
- CG deals with algorithms for geometric data



polygons

Introduction

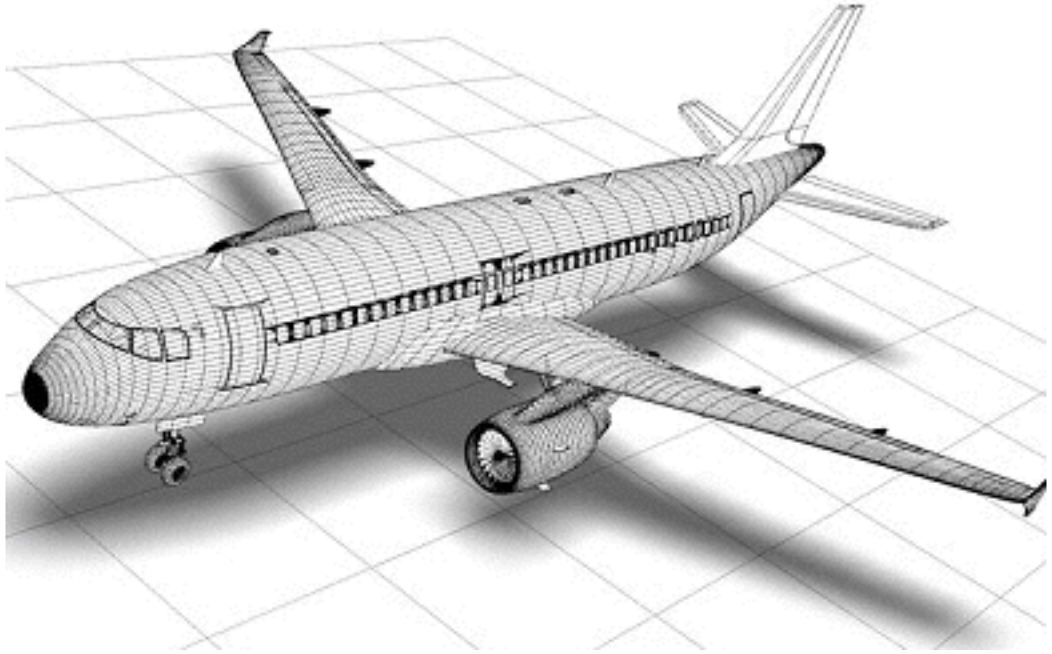
- CG deals with algorithms for geometric data

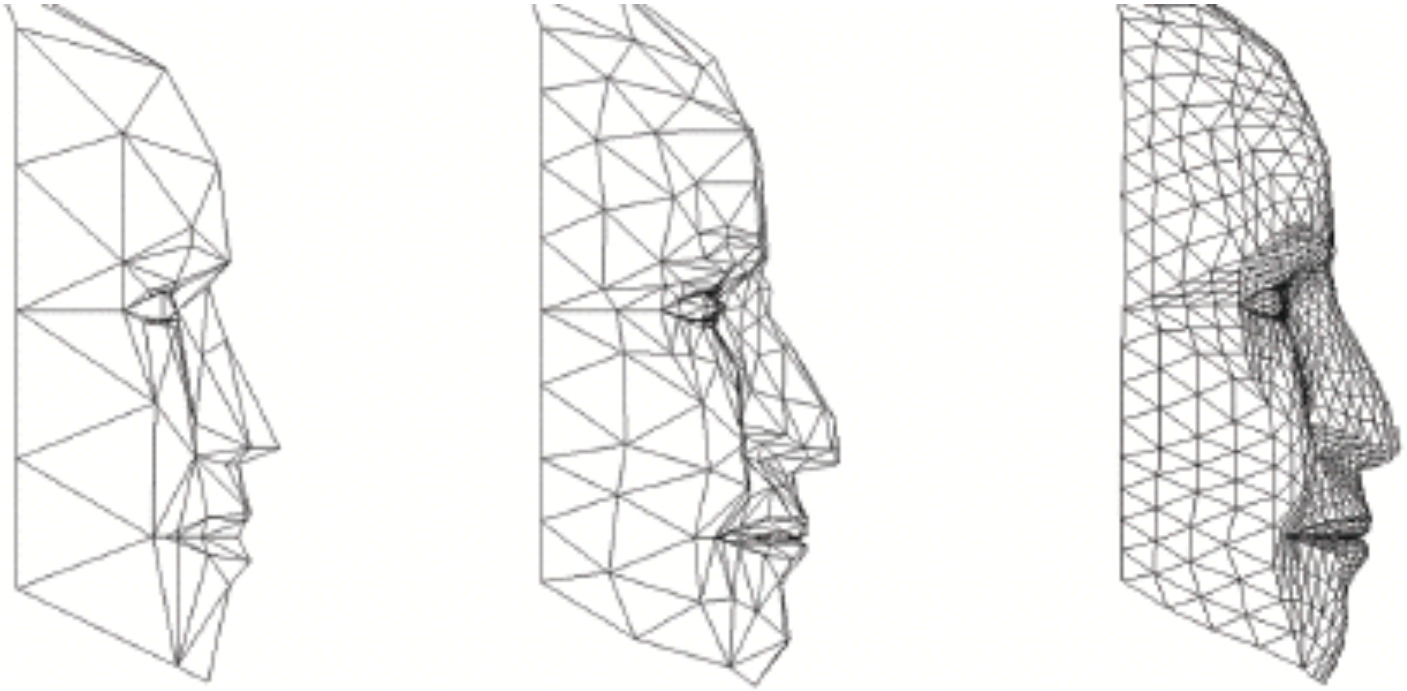


polygons

Introduction

- CG deals with algorithms for geometric data

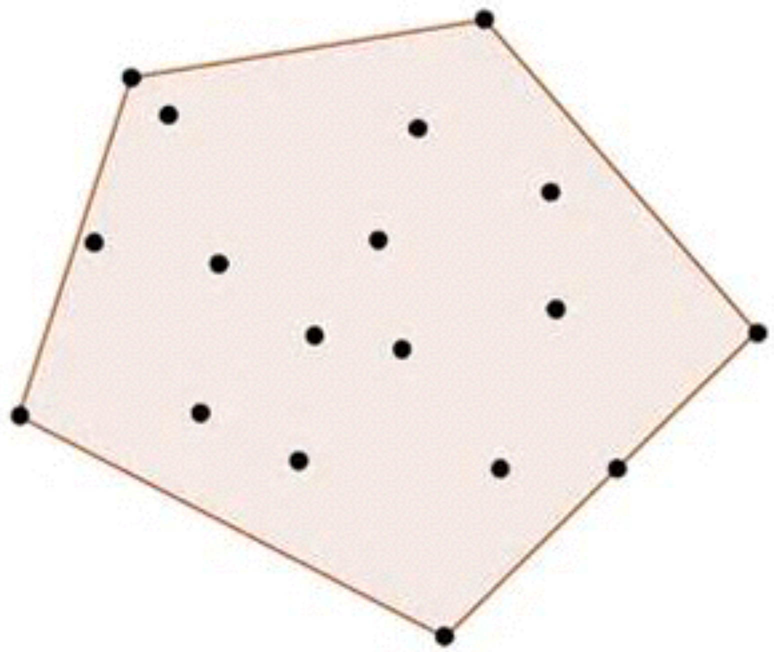


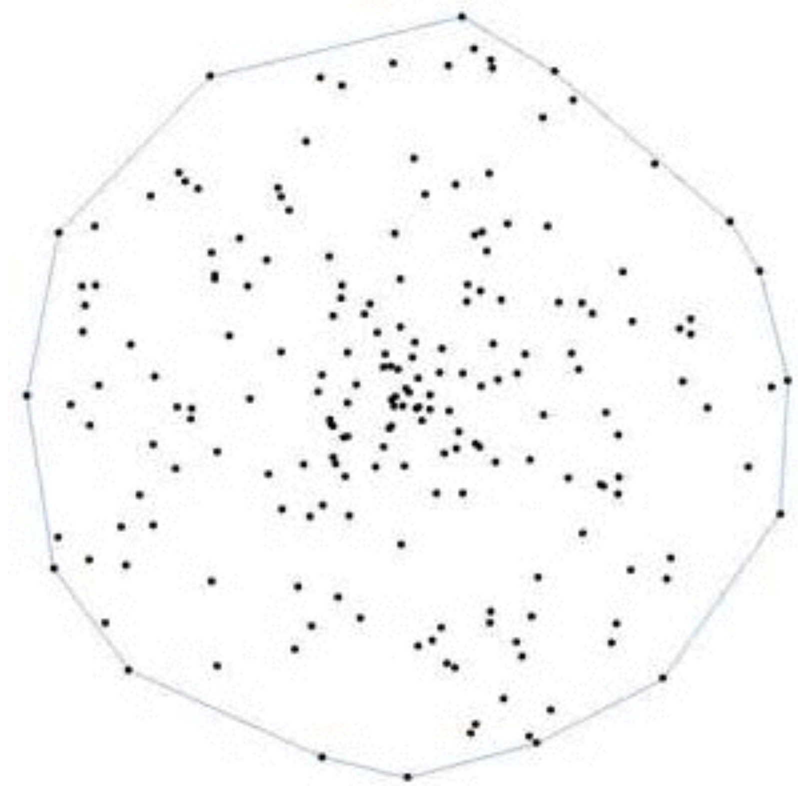


2D, 3D..

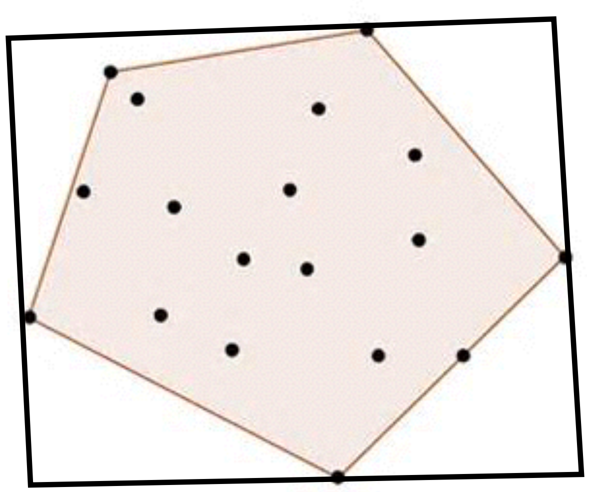
Class overview

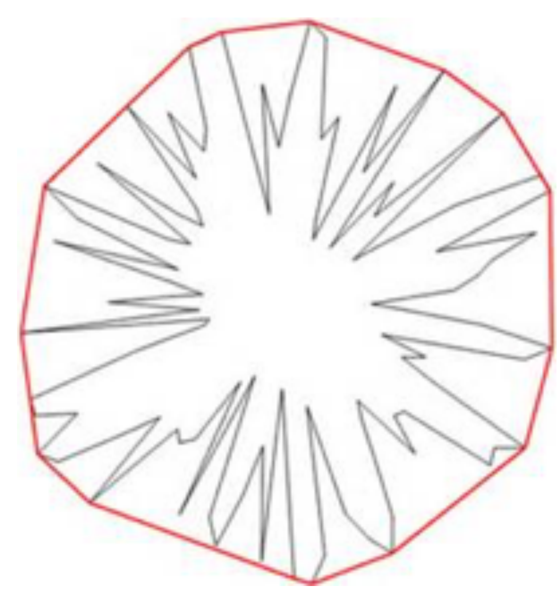
- Convex hull





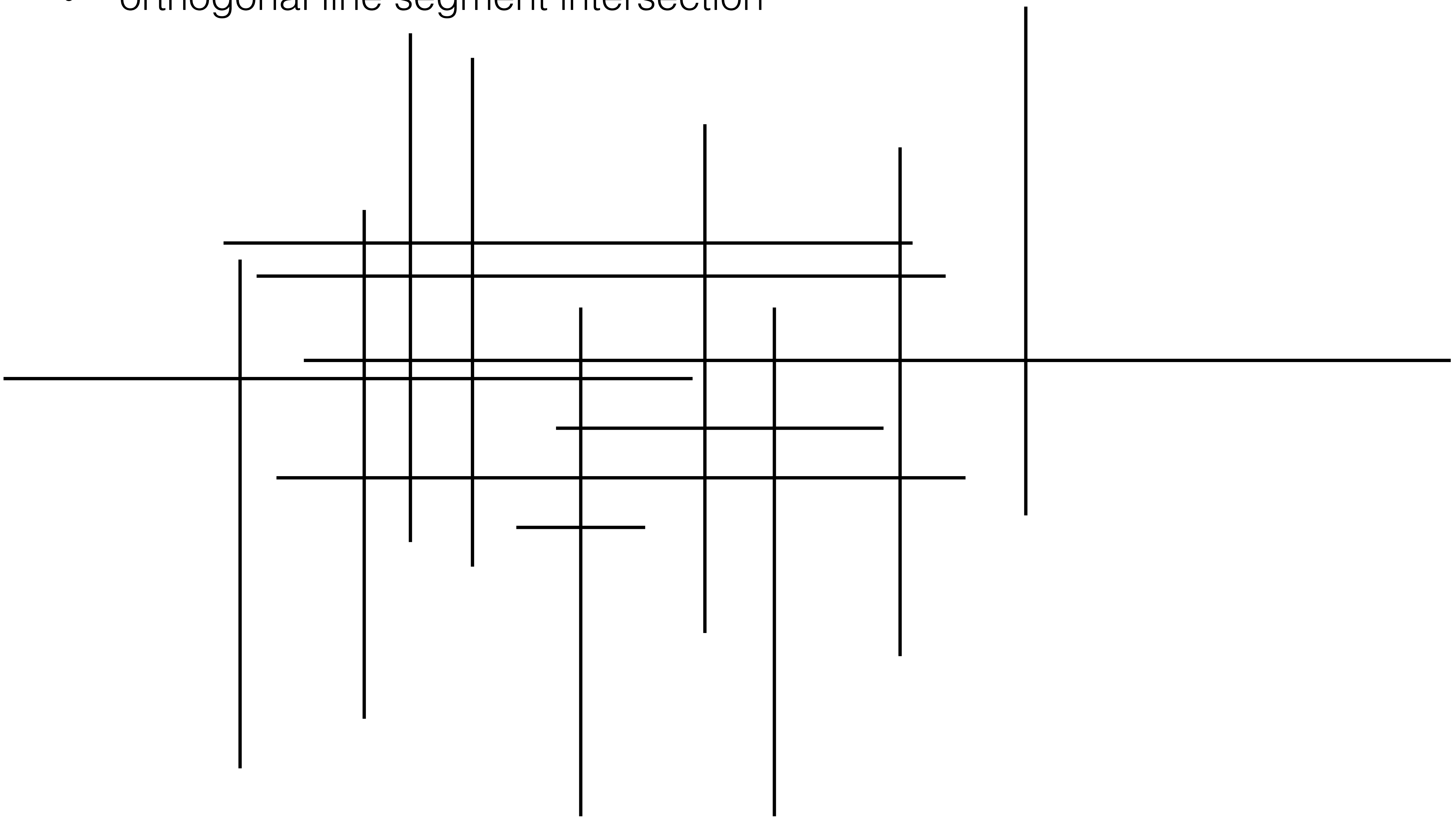
  - comes up in a lot of applications
  - objects are approximated by their CH shape





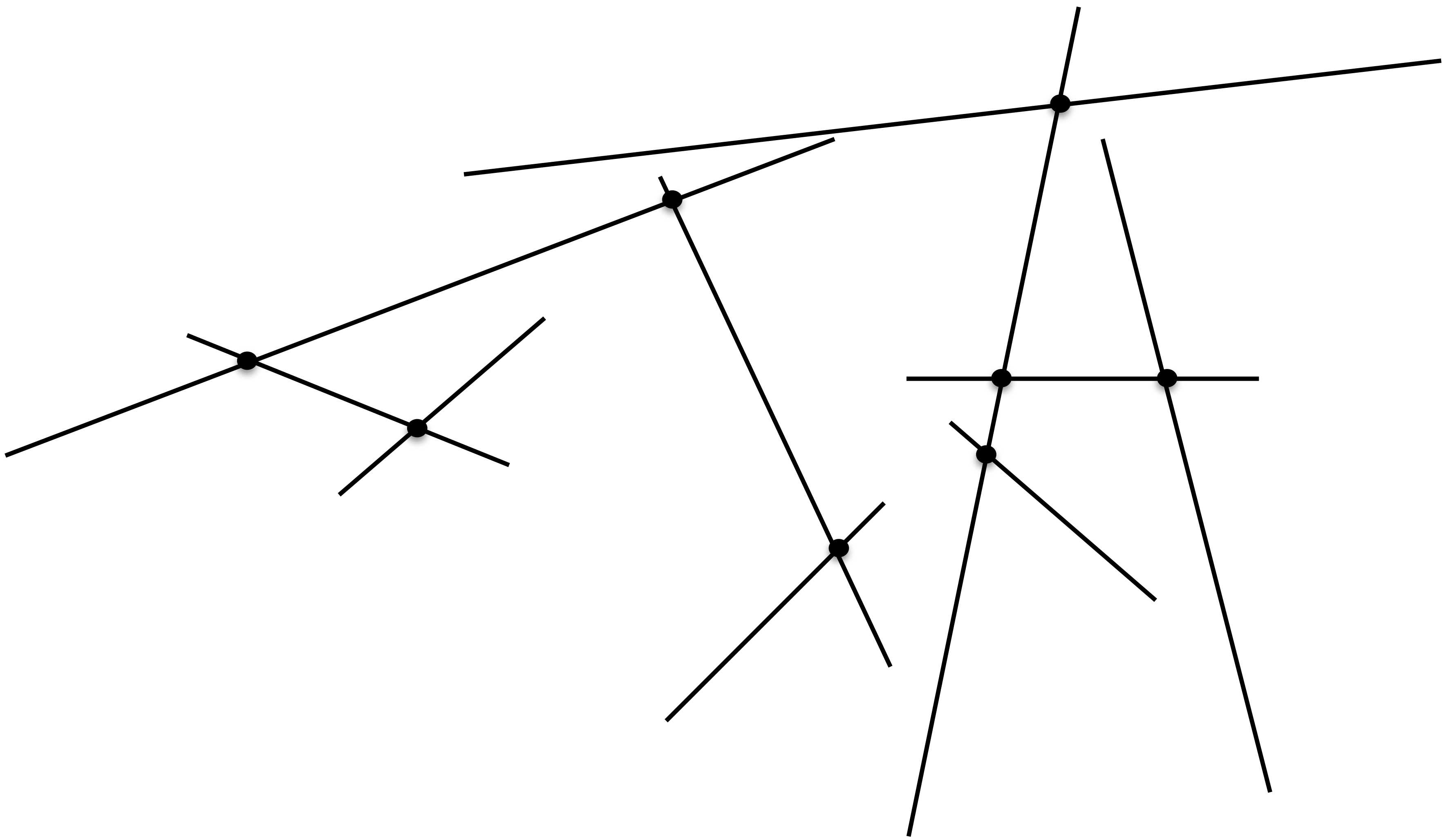
Class overview

- Intersections
  - orthogonal line segment intersection



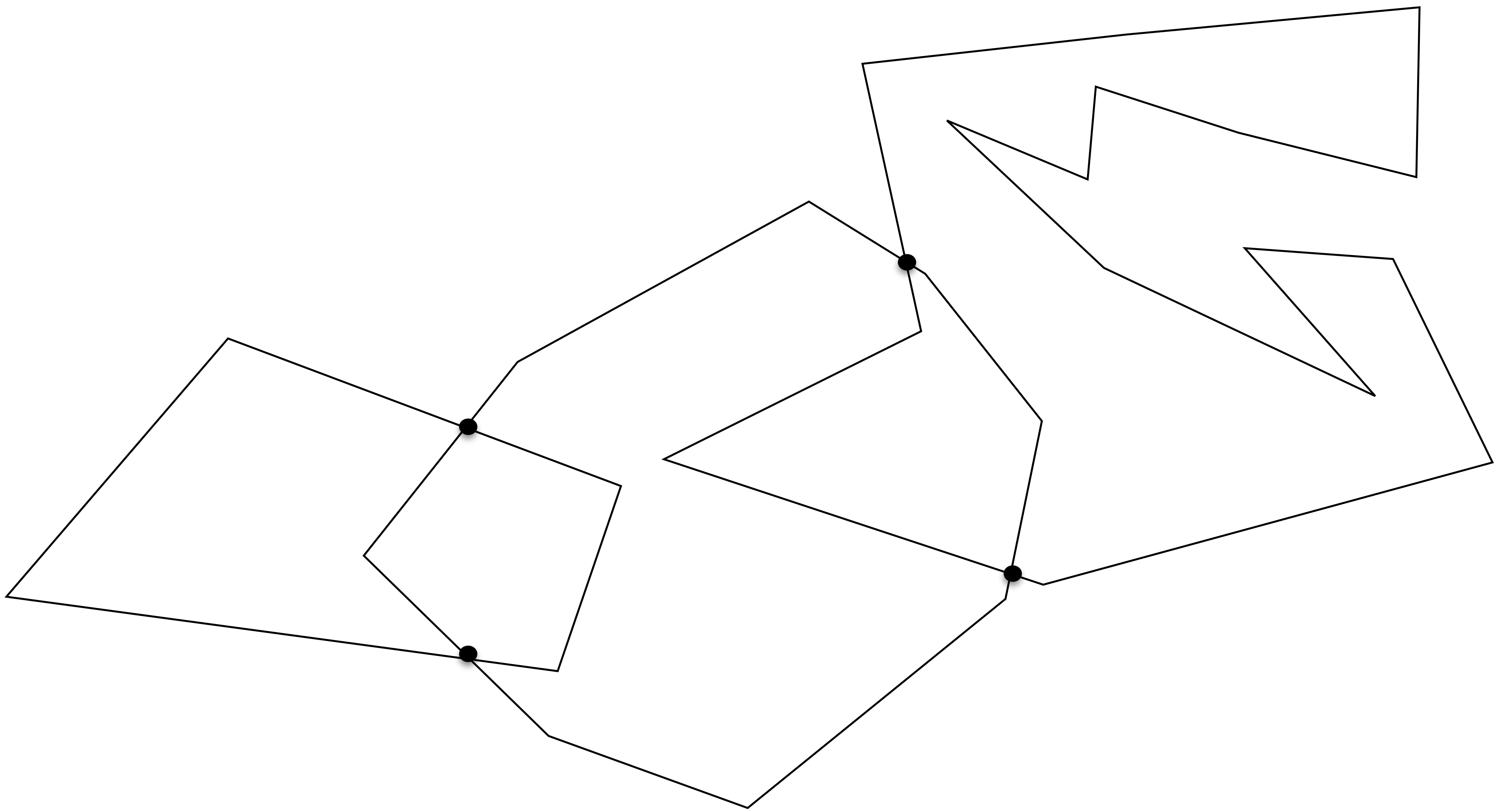
Class overview

- Intersections
  - general line segment intersection



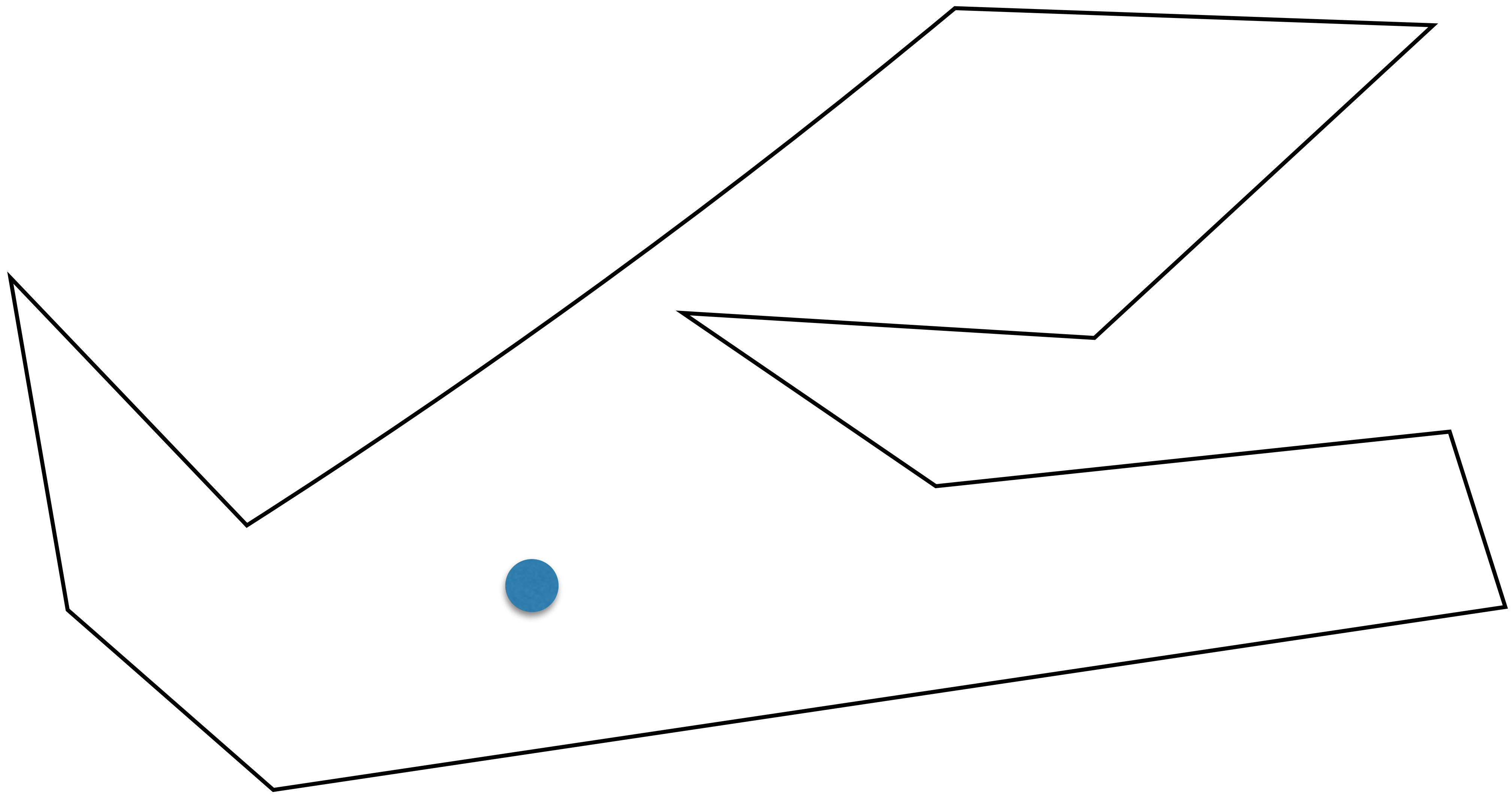
Class overview

- Intersections
  - general line segment intersection



Class overview

- Visibility
  - art gallery problem

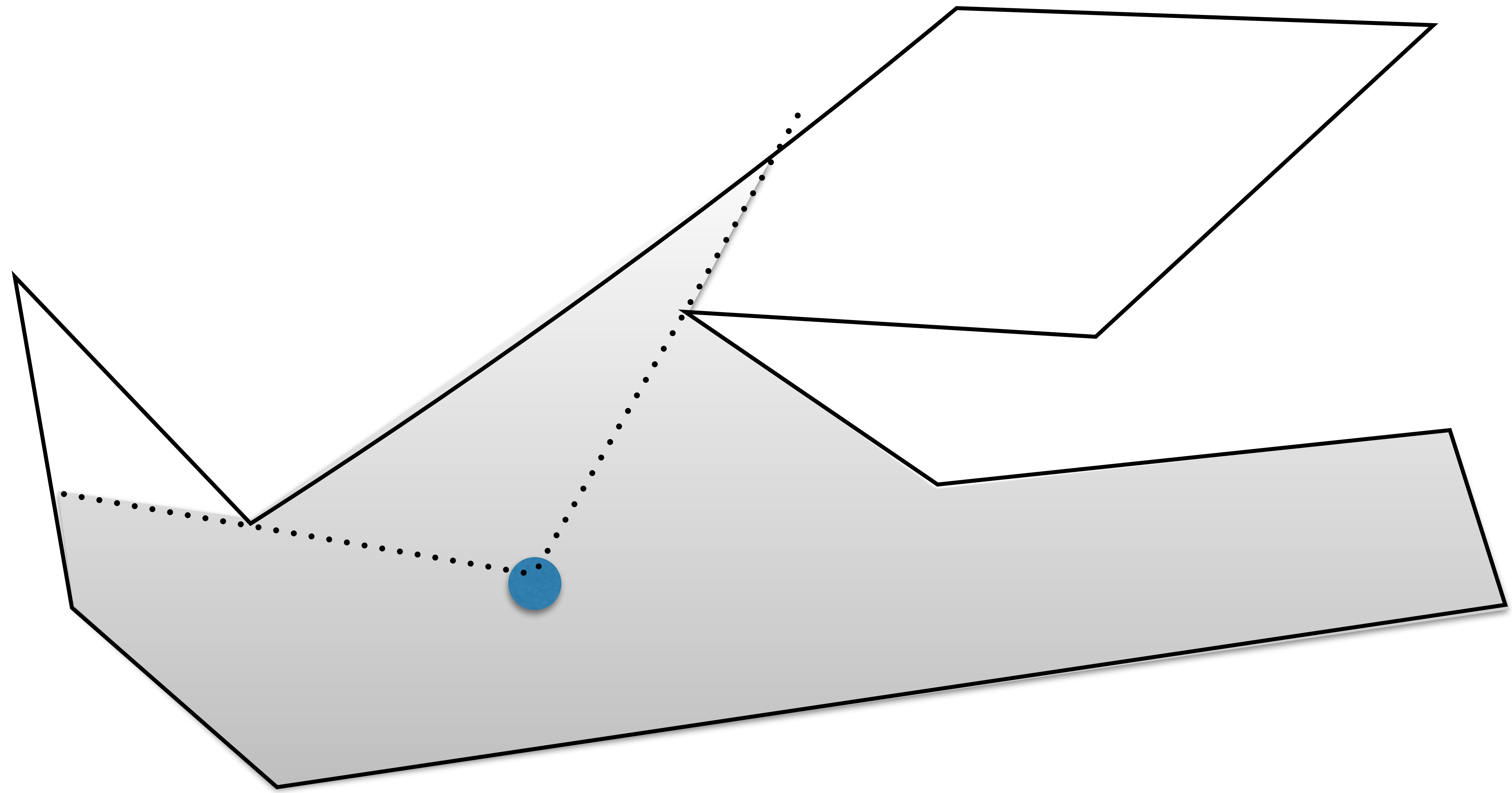


What part of the polygon can the guard see?

How many guards necessary to cover this polygon?

Class overview

- Visibility
    - art gallery problem



What part of the polygon can the guard see?

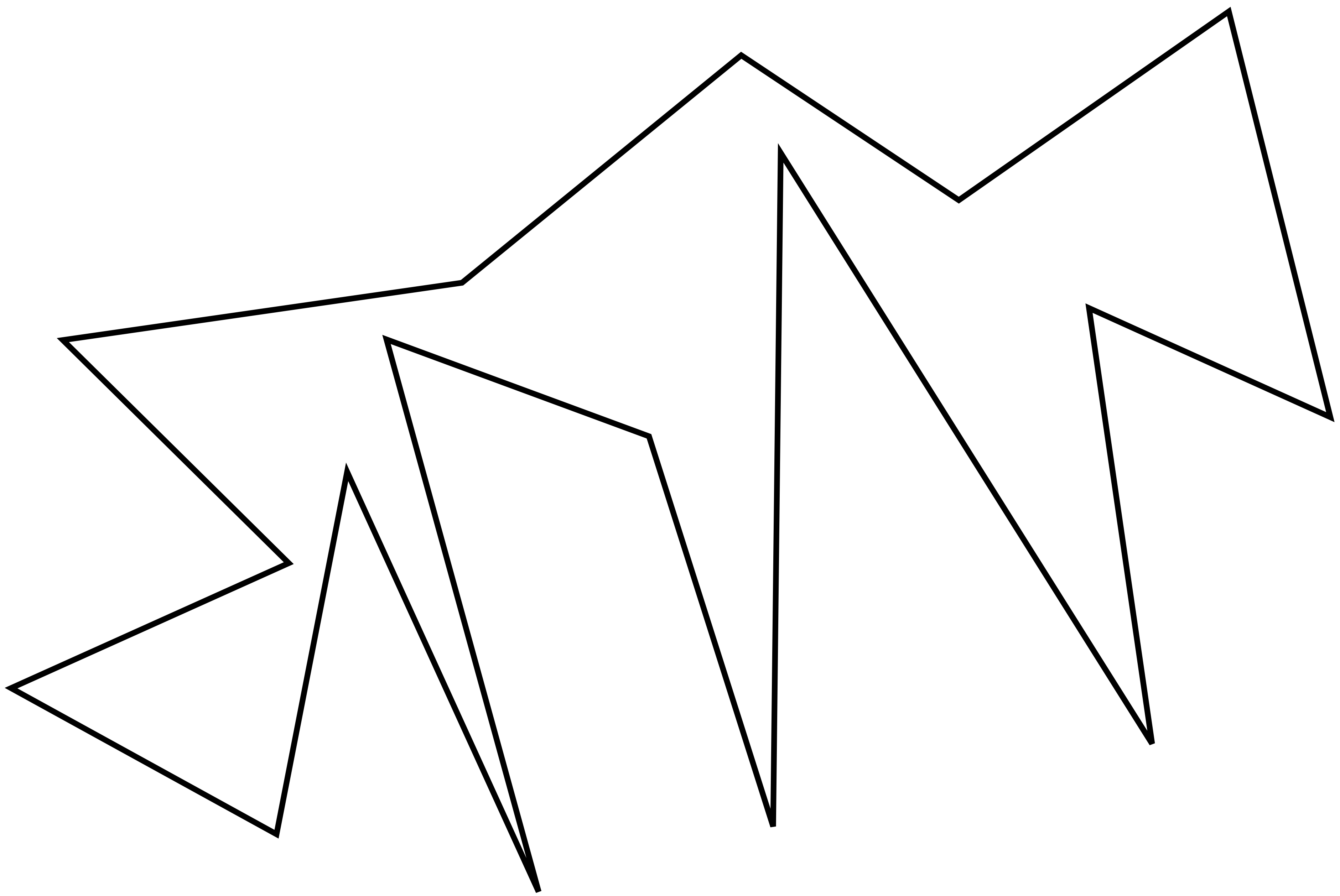
How many guards necessary to cover this polygon?

Class overview

- Triangulation and partitioning
  - subdivide a complex domain into simpler objects
  - simplest object: triangulation

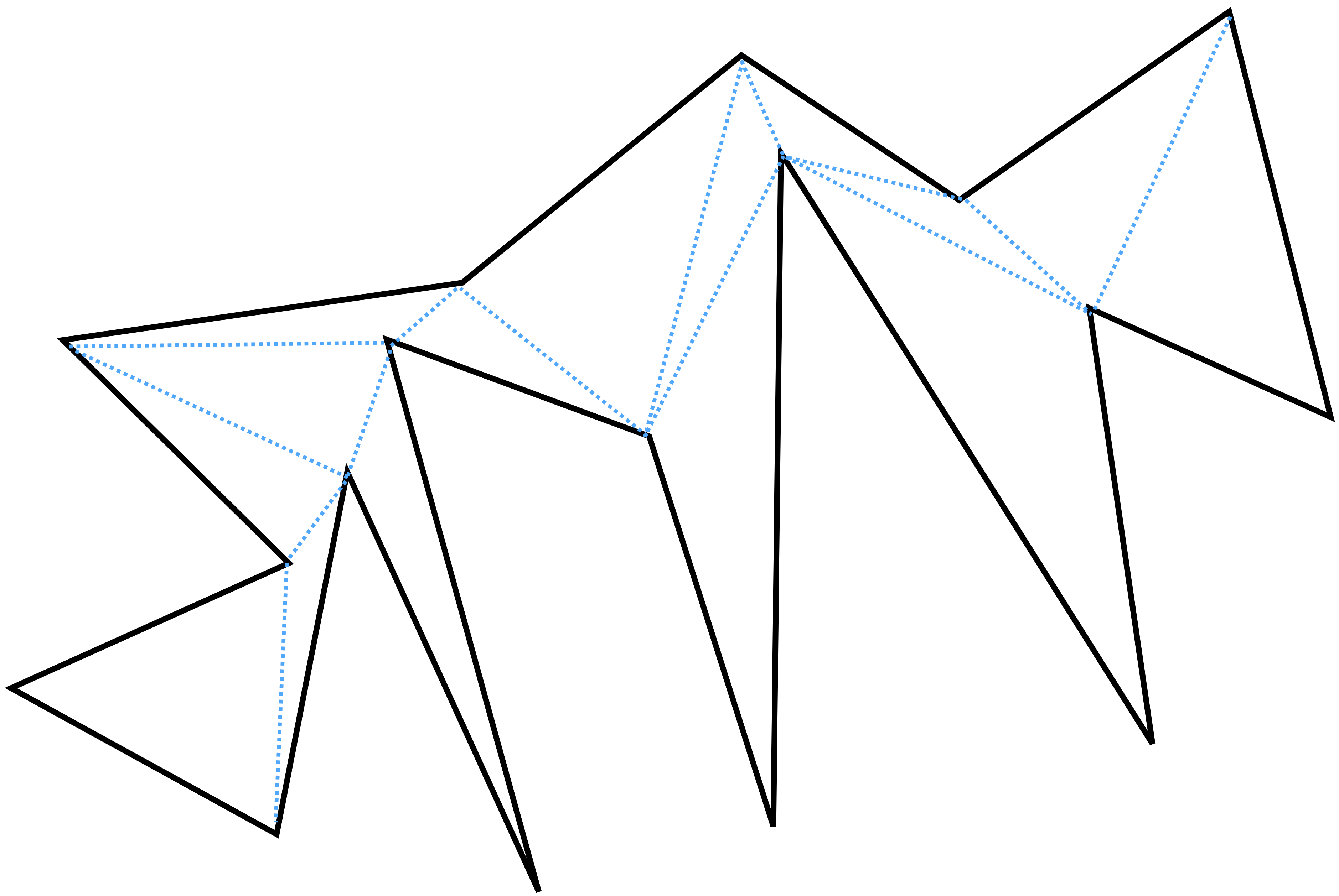
Class overview

- Polygon triangulation
  - output a set of diagonals that partition the polygon into triangles



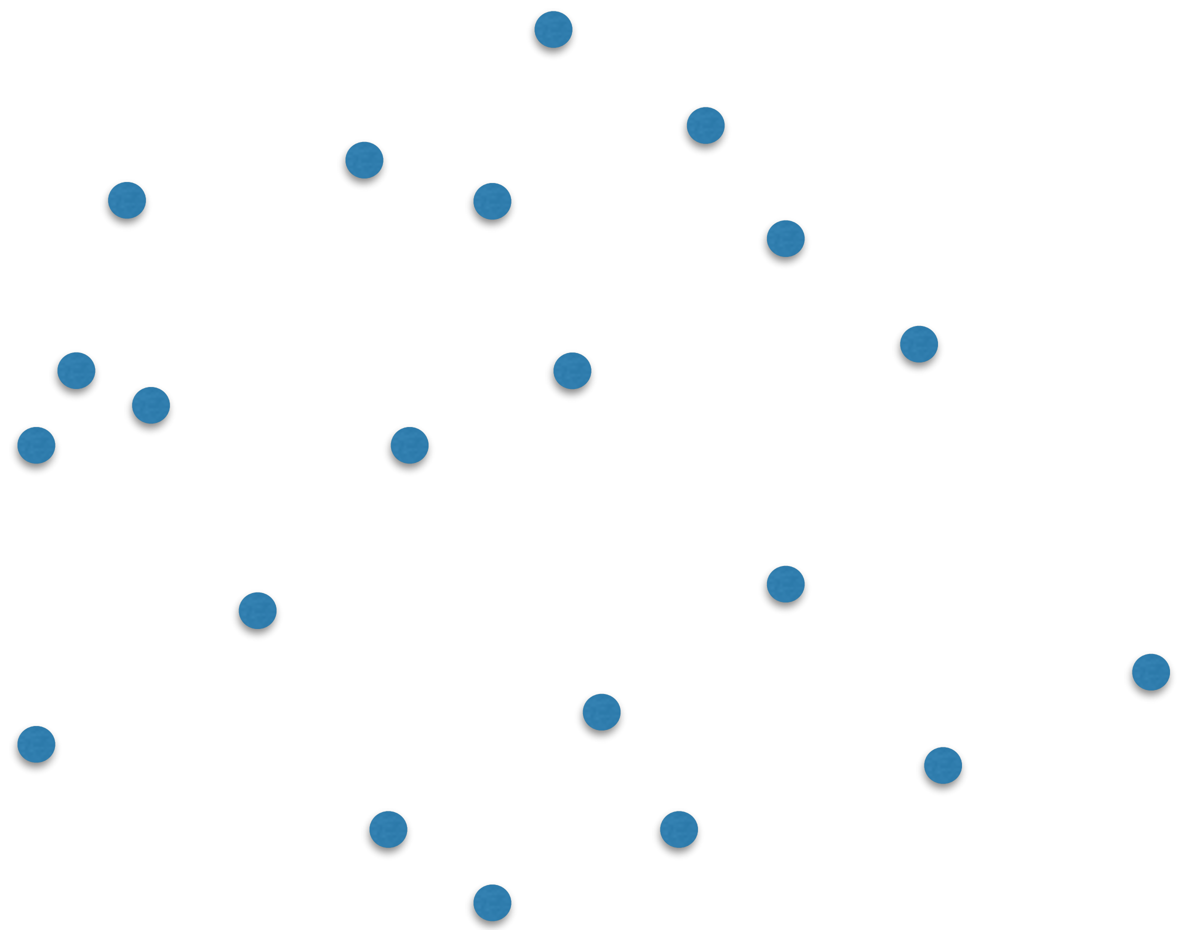
Class overview

- Polygon triangulation
    - output a set of diagonals that partition the polygon into triangles



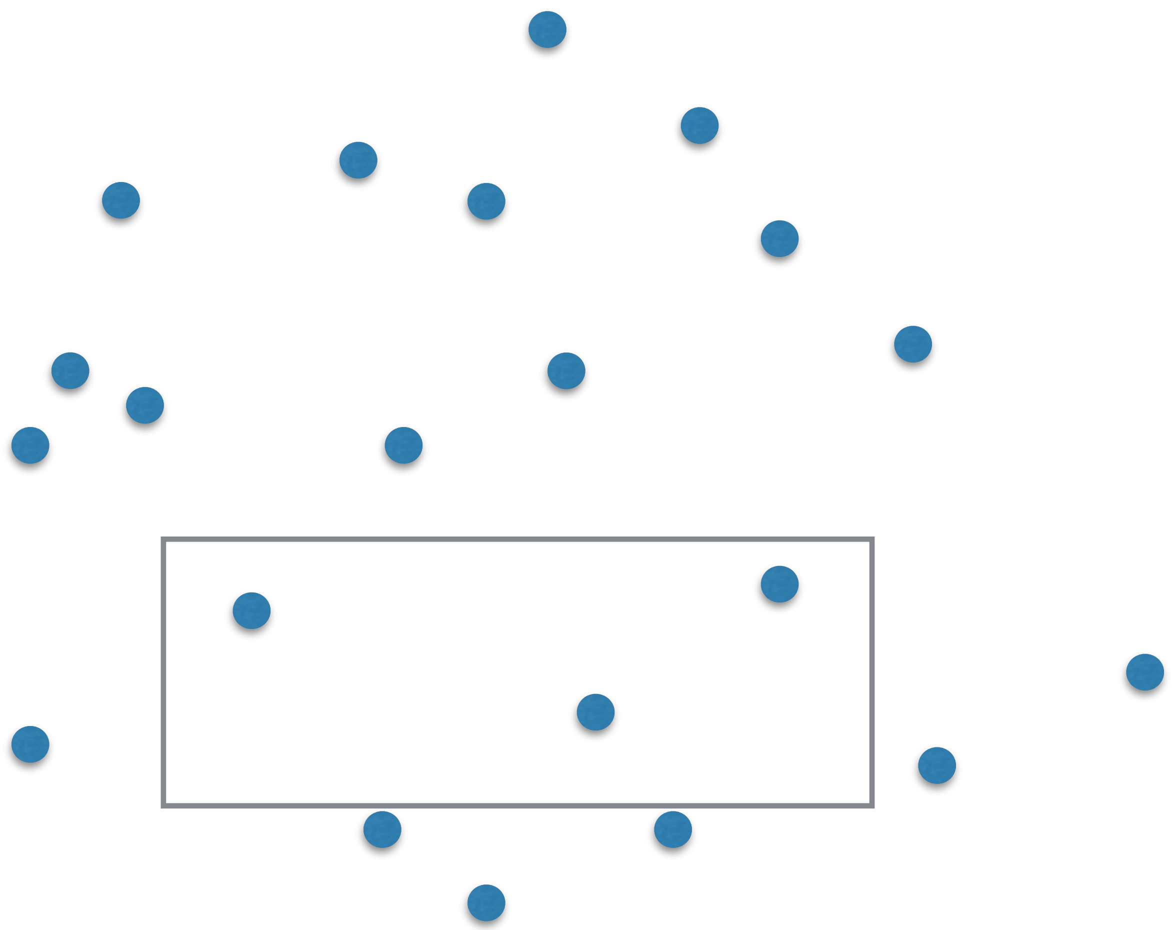
Class overview

- Range searching



Class overview

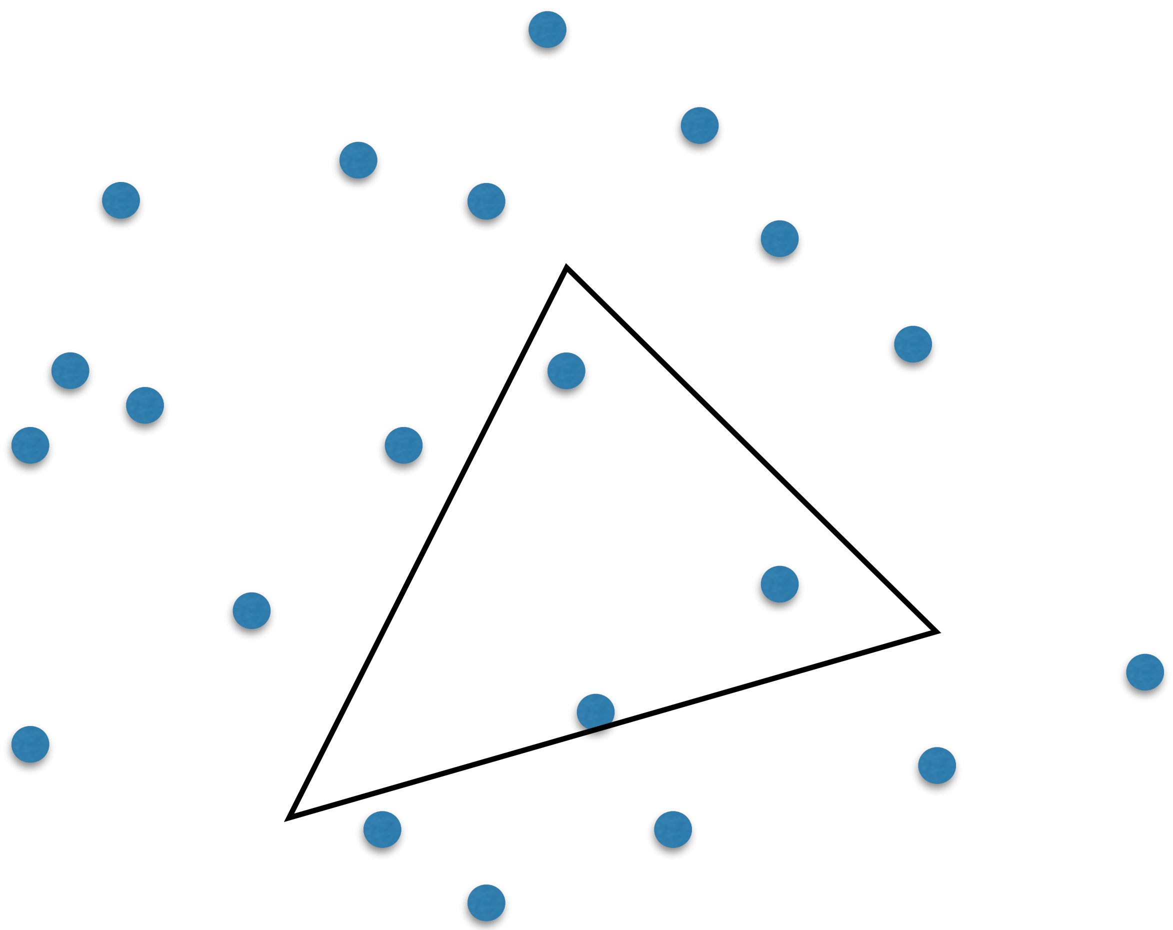
- Range searching



find all points in this range

Class overview

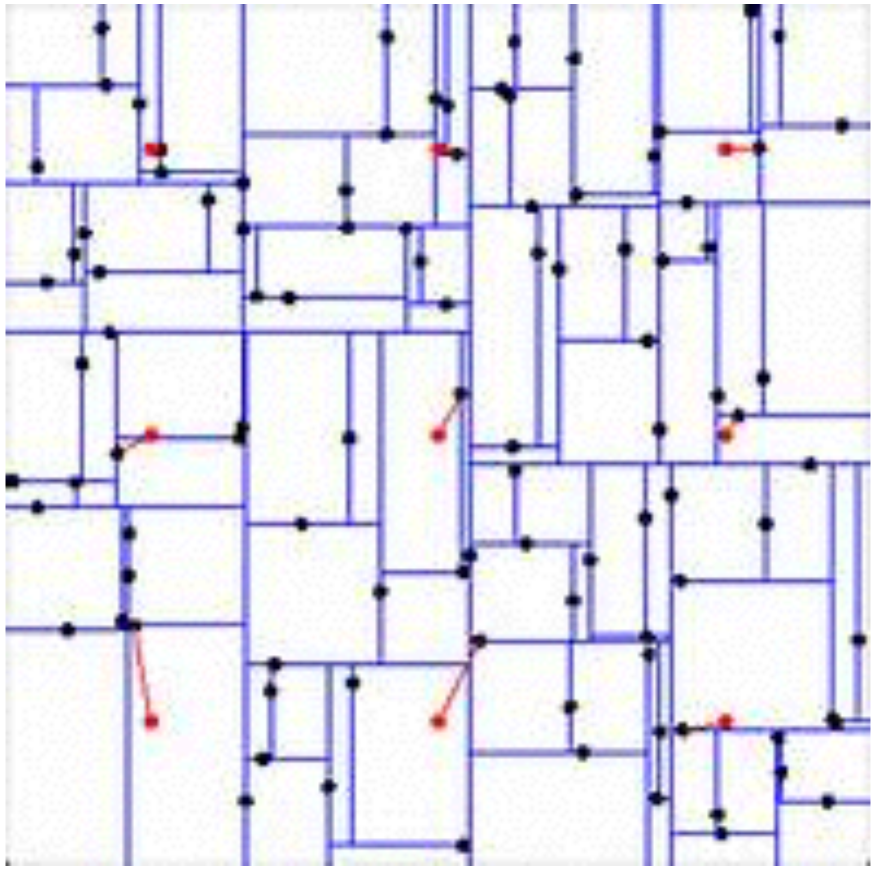
- Range searching



find all points in this range

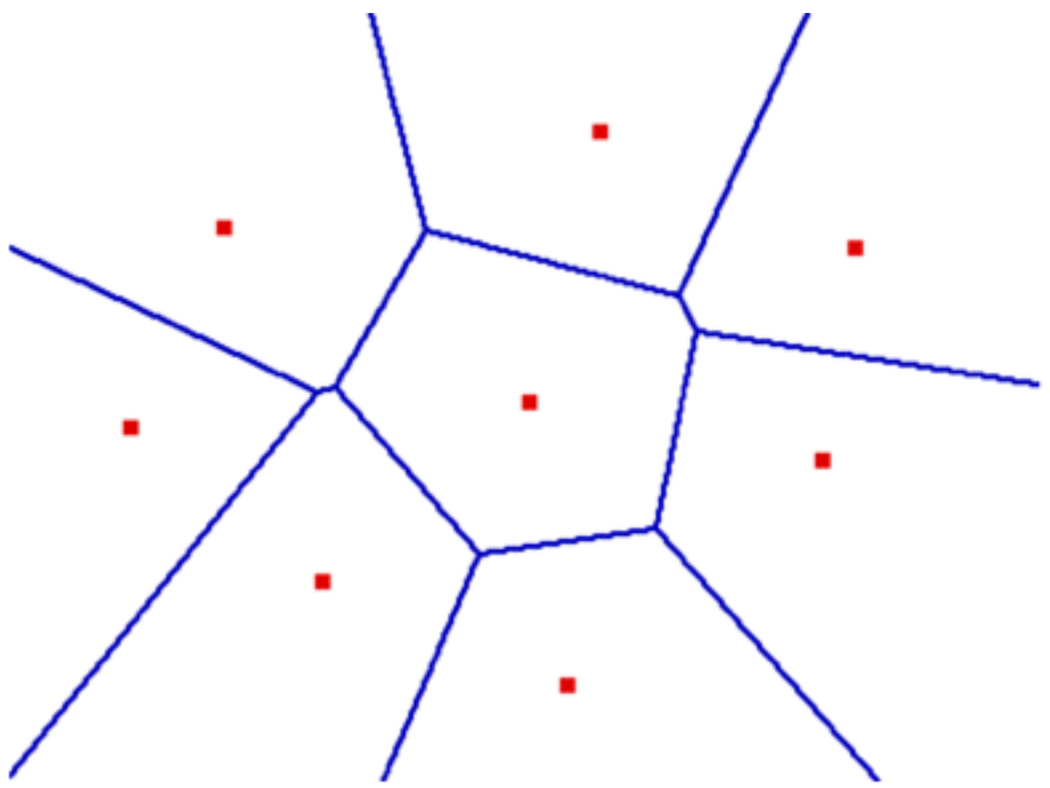
Class overview

- Range searching
  - range tree
  - kd-tree



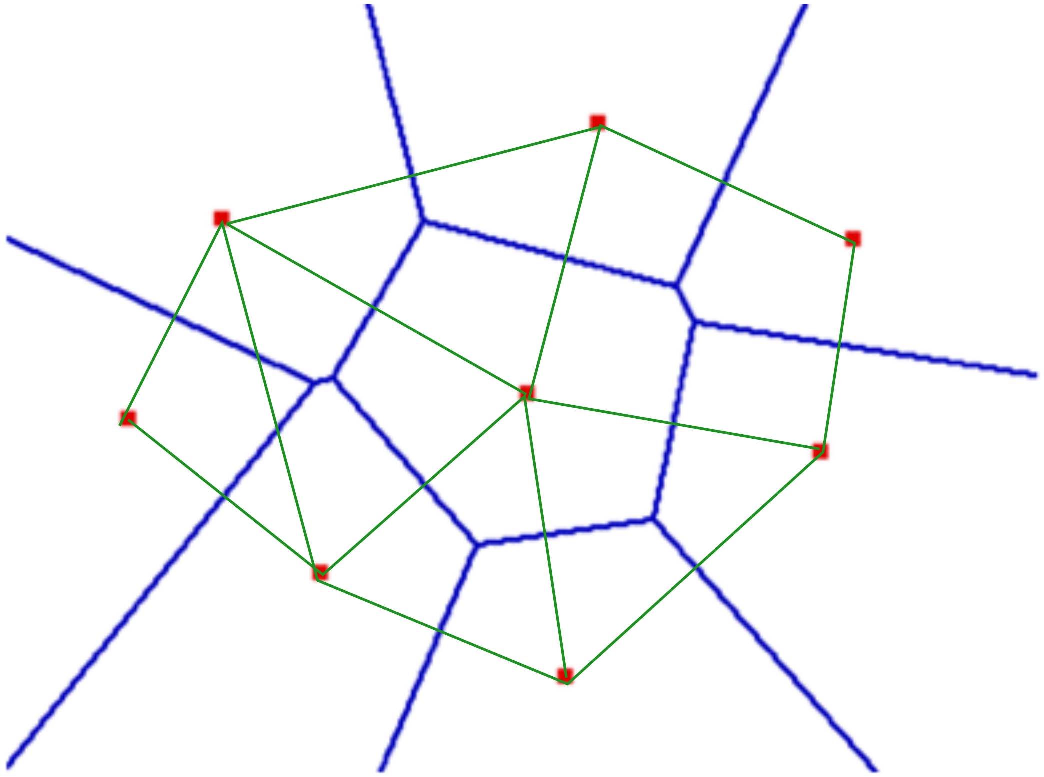
Class overview

- Proximity problems
    - Voronoi diagram

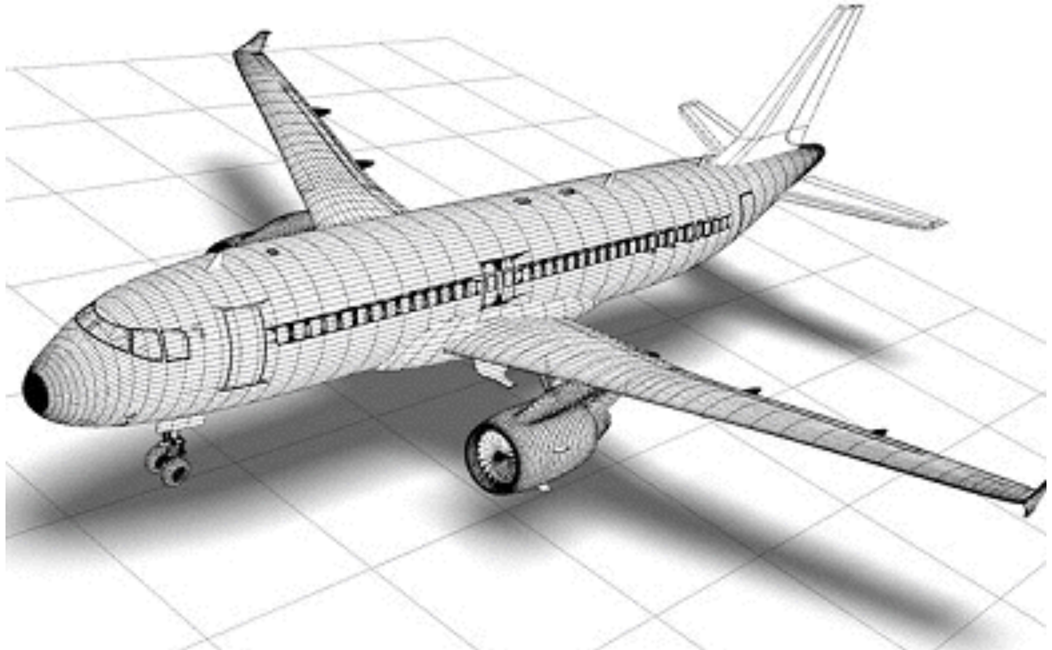


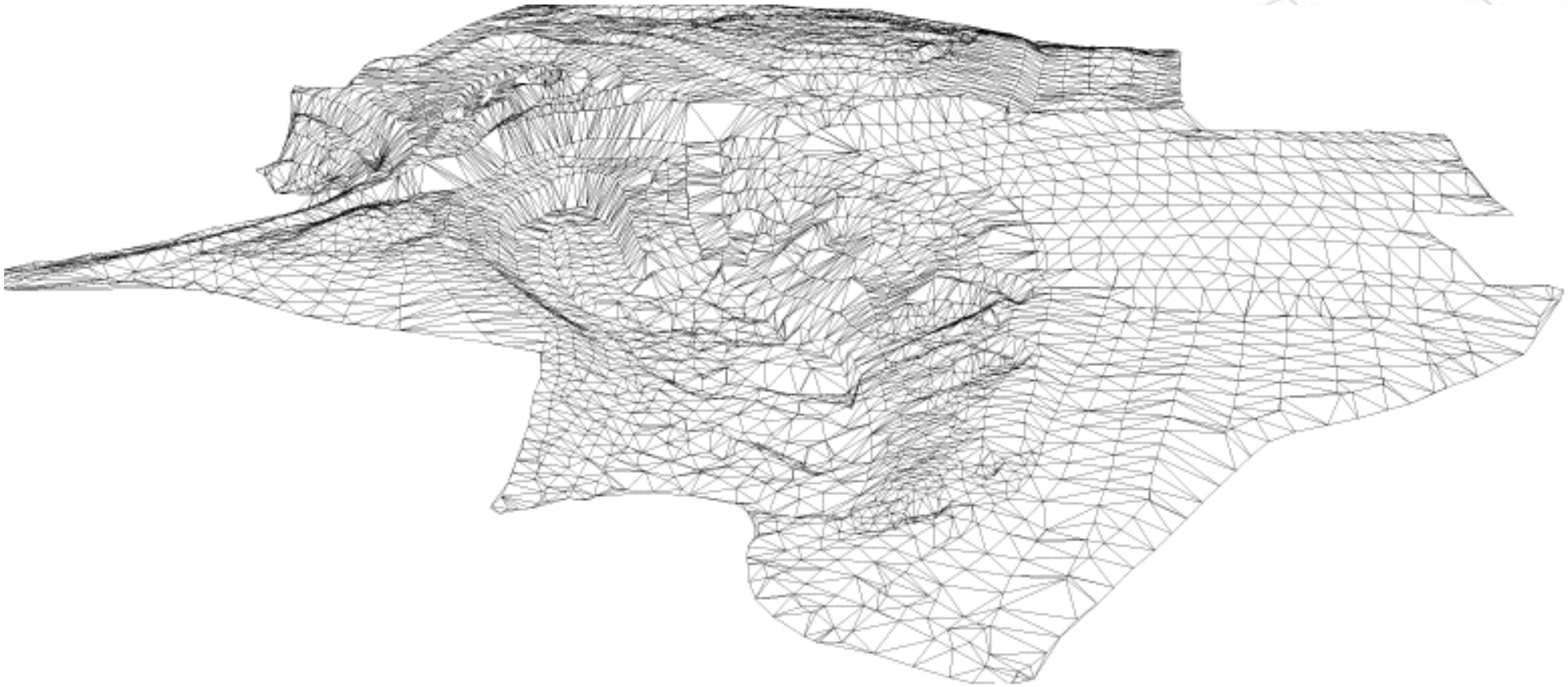
Class overview

- Proximity problems
    - Voronoi diagram



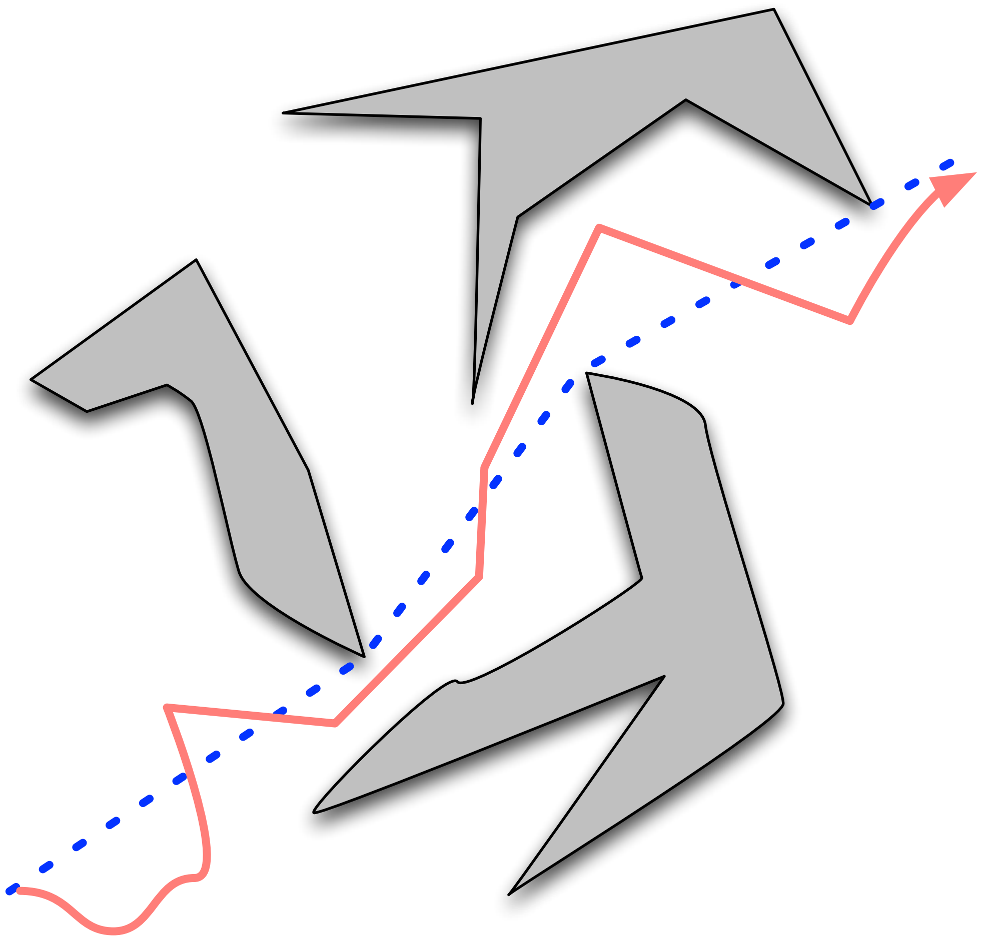
Delaunay Triangulations





Class overview

- Motion planning
  - find collision-free path from start to end moving among obstacles



Applications

- Computer graphics
  - rendering, hidden surface removal, lighting, moving and collision detection
- Robotics
  - path planning involves finding paths that avoid obstacles; this involves finding intersections
  - does this route intersect this obstacle?
- Cell phone data
  - stream of coordinates
  - e.g. find congestion patterns, model real-time traffic conditions (done by cell phone apps)
- Spatial database engines
  - e.g. Oracle spatial contains specialized data structures for answering queries on geometric data
  - e.g. find all intersections between two sets of line segments (road and rivers)

Computational geometry

- We'll talk about algorithms
- Example: the convex hull of a set of n points in the plane
    - Properties
    - Come up with an algorithm to …
        - e.g. find the convex hull of a set of points
    - What is the complexity of the problem/result?
        - e.g. the convex hull of a set of n points n the plane?
    - What is the worst-case running time for the algorithm?
    - Can we do better? What is a lower bound for the problem?
    - Is the algorithm practical? Can we speed it up by exploiting special cases of data (that arise in practice)?

Logistics

- Lectures and in-class group work
- Material is theoretical
- All work comes from programming assignments
  - expect 5-7 assignments
  - in C/C++ (but I'm open to Python)
  - can be open-ended
  - teams of 2 people
- Textbooks
- TAs and office hours

Today: warmup

Problem:

Given a set of n points in 2D, determine if there exist three that are collinear

- What is the brute force solution?
- Can you refine it?

Finding collinear points

Brute force:

- for all distinct triplets of points p_i, p_j, p_k
  - check if they are collinear

- Analysis:
  - n chose 3 = $O(n^3)$ triplets
  - checking if three points are collinear can be done in constant time

==> $O(n^3)$ algorithm

Finding collinear points

Improved idea 1:

- initialize array L = empty
- for all distinct **pairs** of points p_i, p_j
  - compute their line equation (slope, intercept) and store in an array L
- sort array L //note: primarily by slope, secondarily by intercept
- //invariant: identical lines will be consecutive in the sorted array
- scan array L, if find any identical lines ==> there exist 3 collinear points

- Analysis:
  - $O(n^2)$ pairs
  - time: $O(n^2 \lg n)$
  - space: $O(n^2)$

Finding collinear points

Improved idea 2:

- initialize BBST = empty
- for all distinct **pairs** of points p_i, p_j
  - compute their line equation (slope, intercept)
  - insert (slope, intercept) in BBST; if when inserting you find that (slope, intercept) is already in the tree, you got 3 collinear points

Note: for this to work, you need to make sure that the key for the BBST is both the slope and the intercept

- Analysis:
  - n chose 2 = $O(n^2)$ pairs
  - time: $O(n^2 \lg n)$
  - space: $O(n^2)$

Finding collinear points

Algorithms

- brute force: $O(n^3)$ time, $O(1)$ space
- refined: $O(n^2 \lg n)$ time, $O(n^2)$ space

Questions

- Can you find a solution that runs in $O(n^2 \lg n)$ time with only linear space?
- Can you improve your solution, for example by making some assumption about the input?

  e.g.: integer coordinates

Integer coordinates

- If points have integer coordinates, we can immediately think of using **hash table** instead of BBST

- Hash table:
    - insert, delete, search
    - O(1) for families of universal hash functions

- Hashing integers
  - families of *universal hash functions* are known for integers which guarantee no collision with high probability
  - O(1) insert/search/delete

- Hashing chars and strings

Integer coordinates

Improved idea 3:

- initialize HT = empty
- for all distinct **pairs** of points p_i, p_j
  - compute their line equation (slope, intercept)
  - check HT to see if already there => if yes, you got 3 collinear points

Time?

Space?