Computational Geometry
csci3250

Laura Toma

Bowdoin College

Today

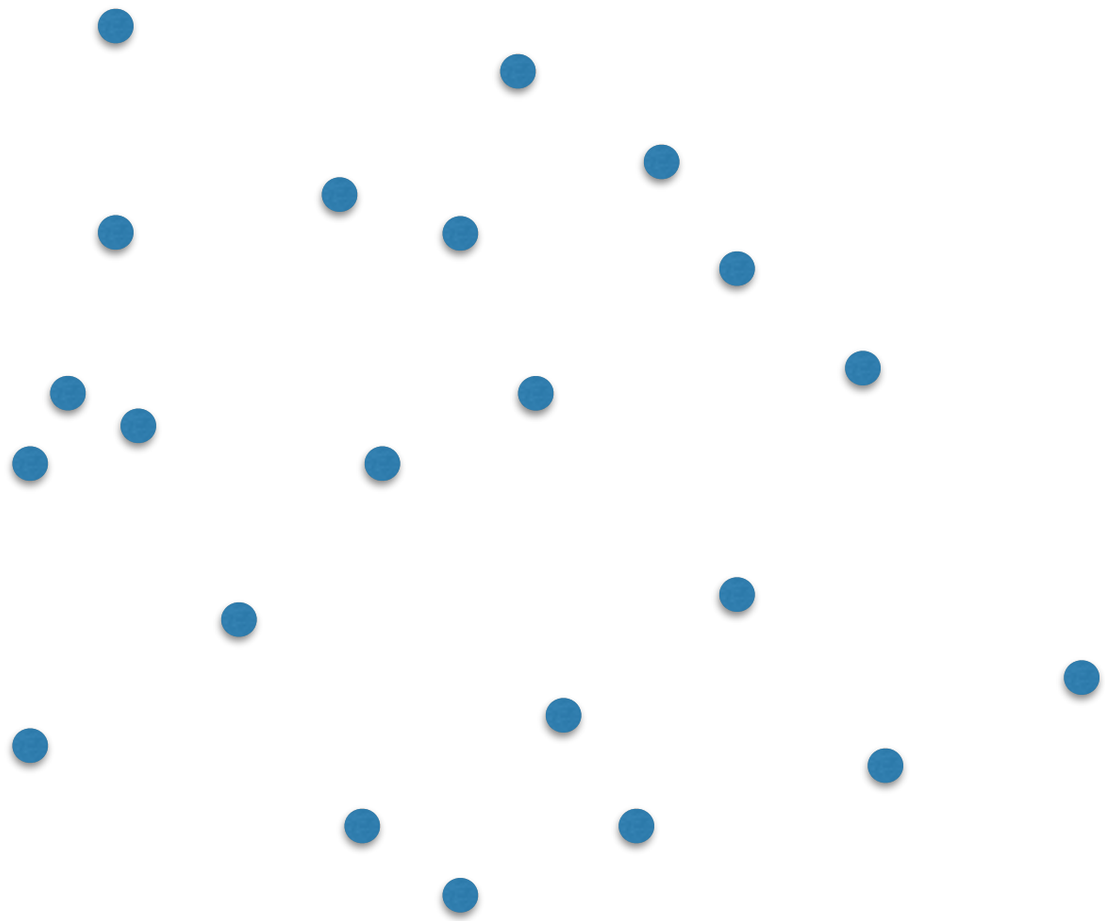
- Introduction
- Warmup
  - finding collinear points
  - finding closest pair of points

Introduction

- CG deals with algorithms for geometric data
  - Focuses on discrete geometry
    - input is a discrete set of points/lines/polygons/..
    - different emphasis than (continuous) geometry

Introduction

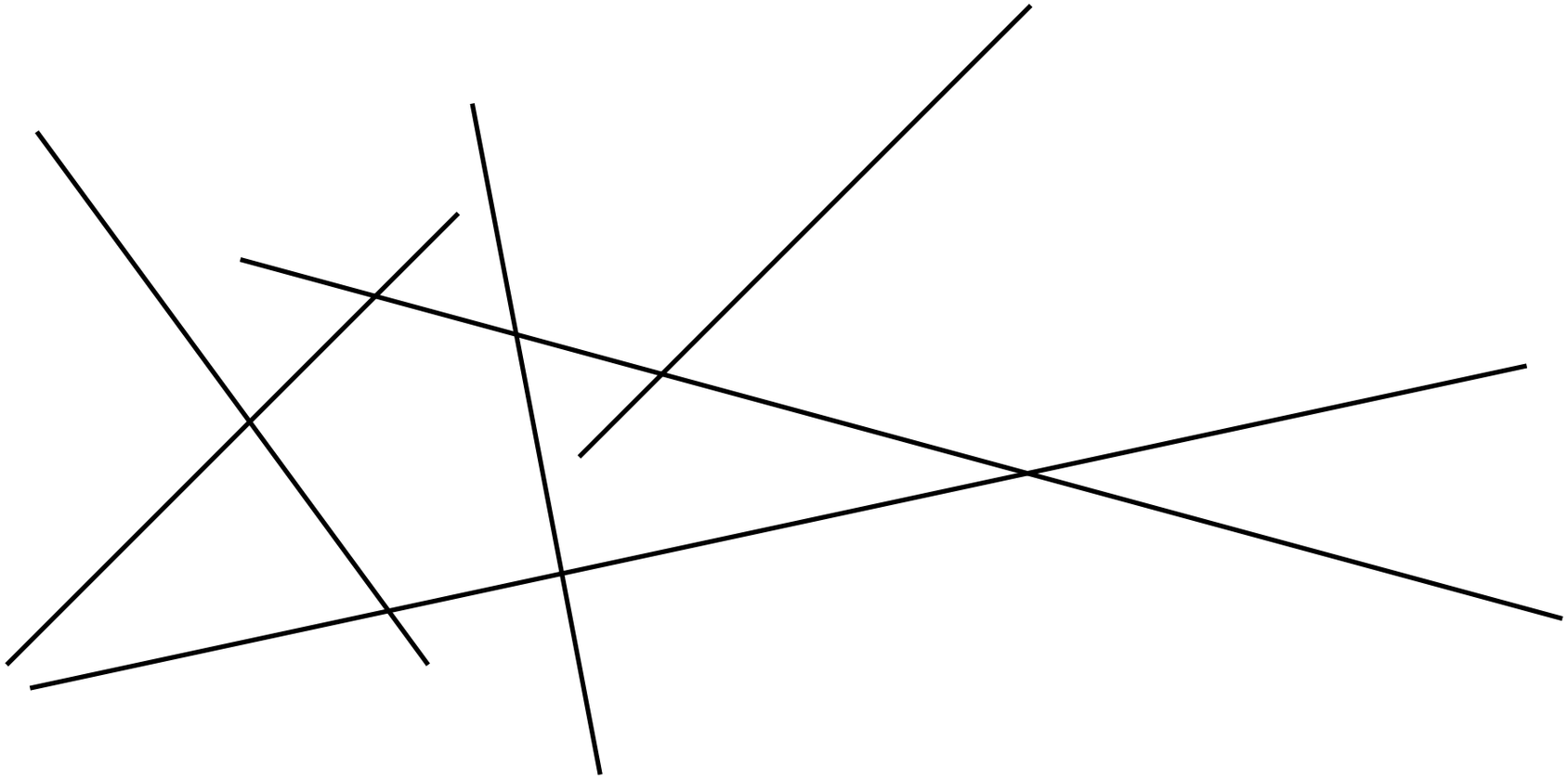
- CG deals with algorithms for geometric data



points

Introduction

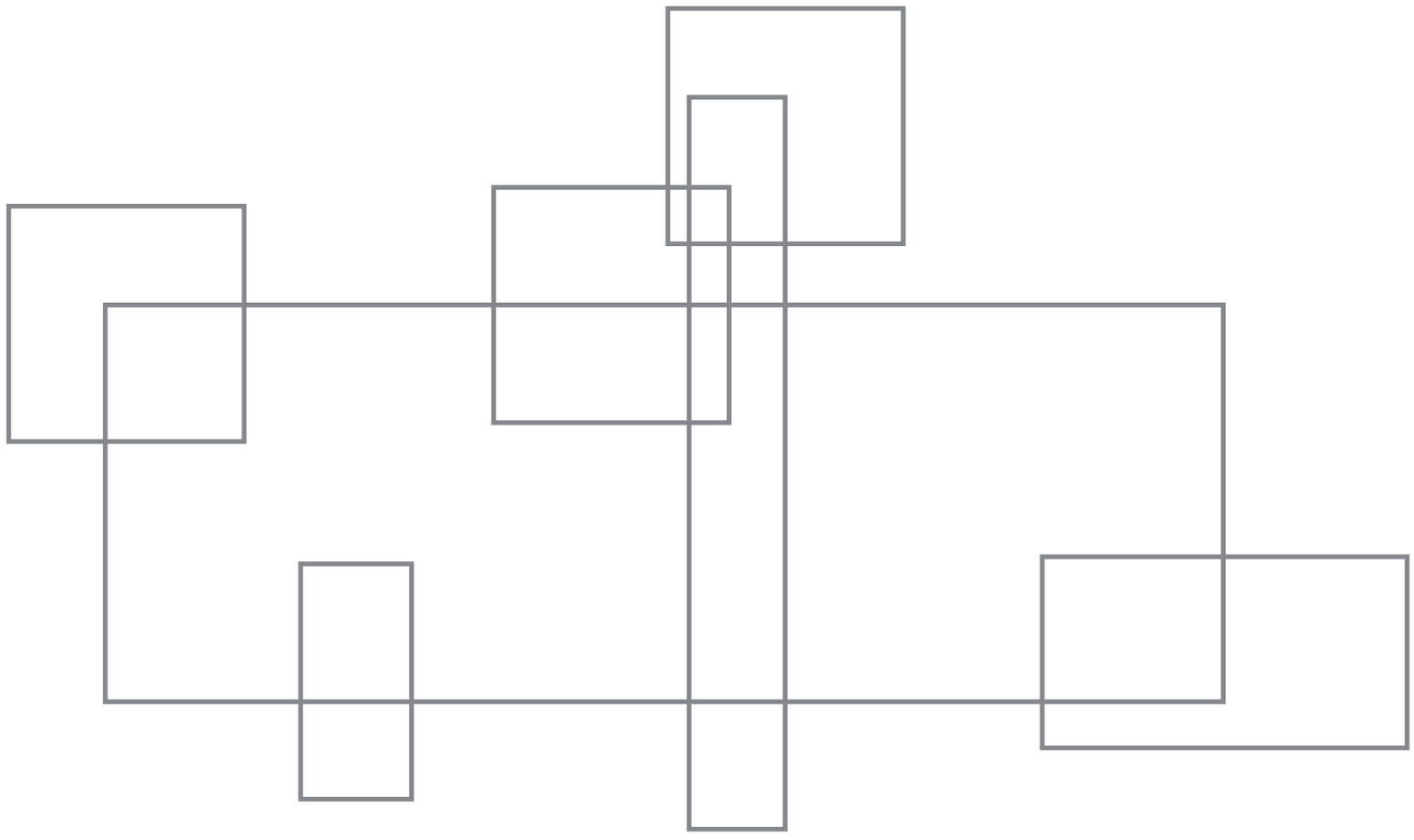
- CG deals with algorithms for geometric data



lines and line segments

Introduction

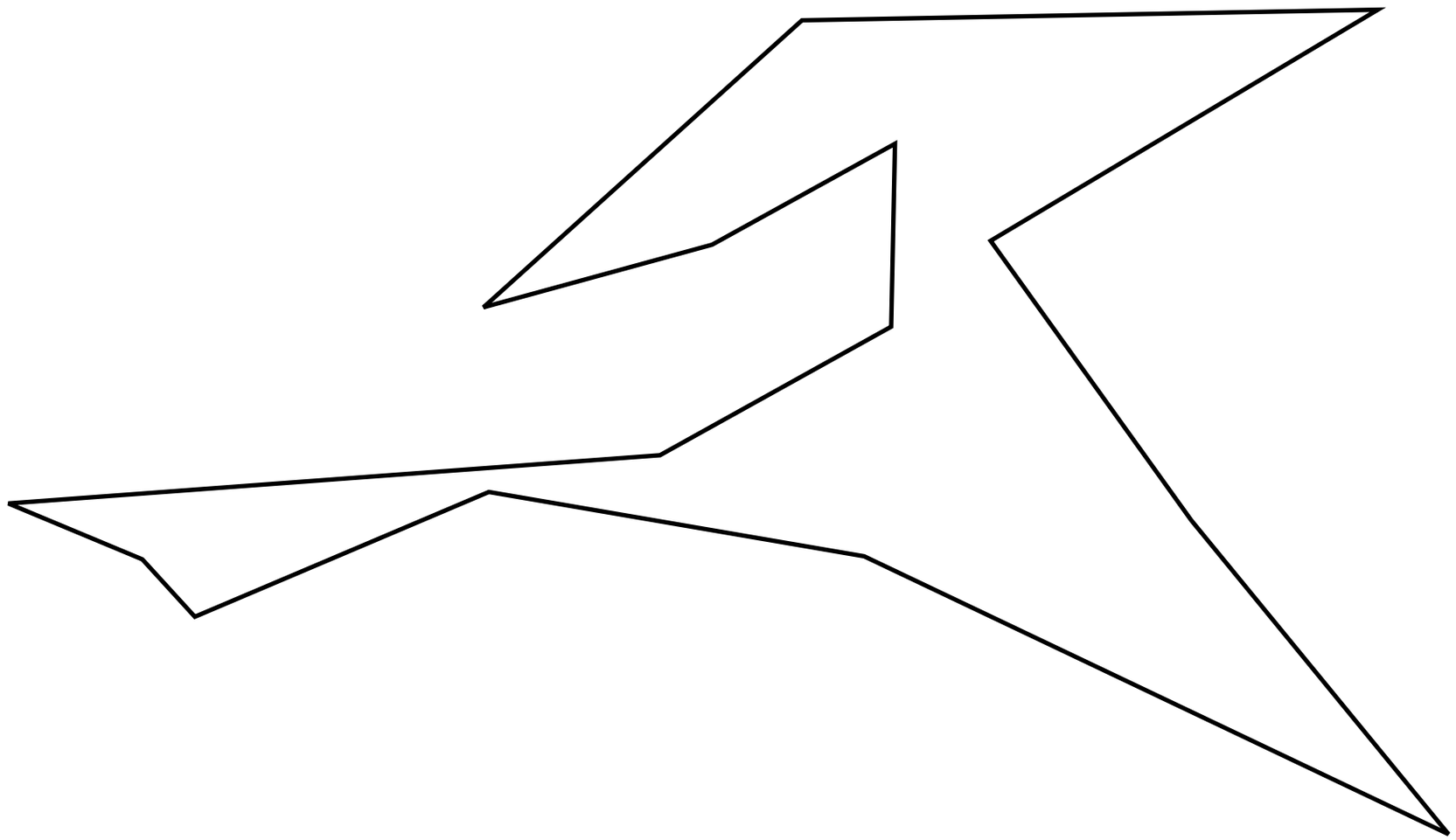
- CG deals with algorithms for geometric data



polygons

Introduction

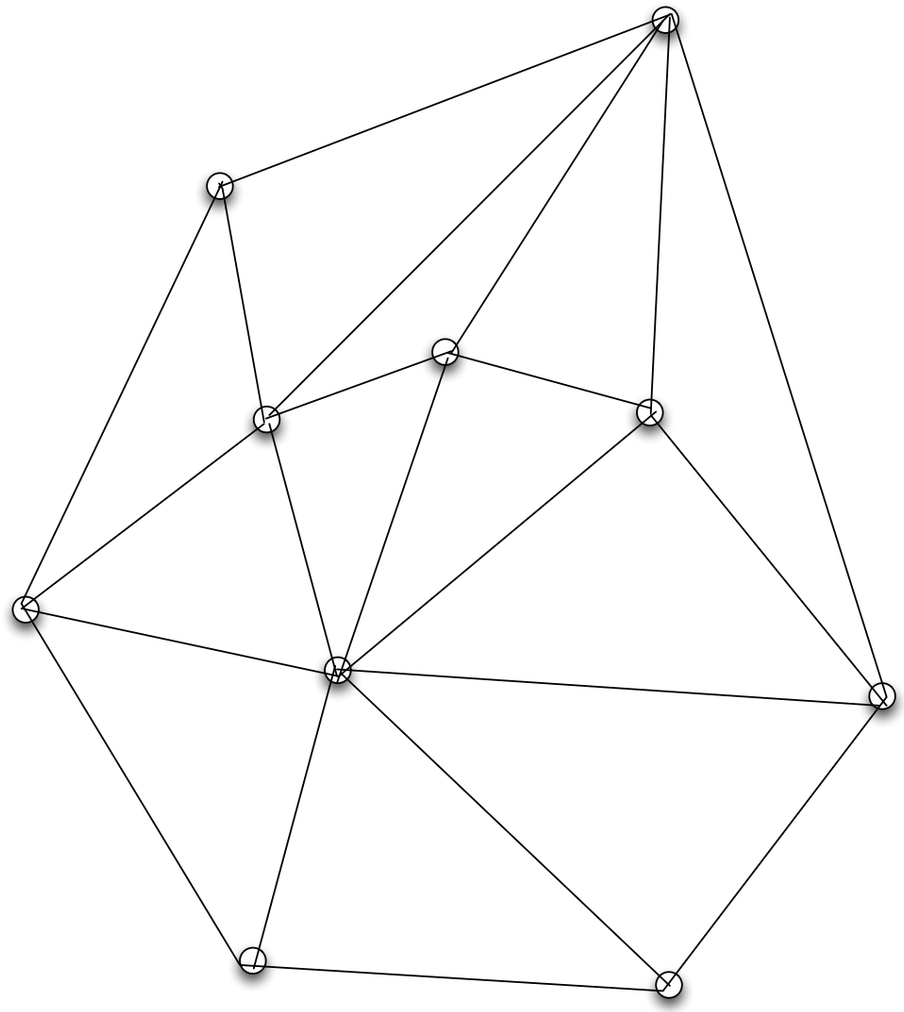
- CG deals with algorithms for geometric data



polygons

Introduction

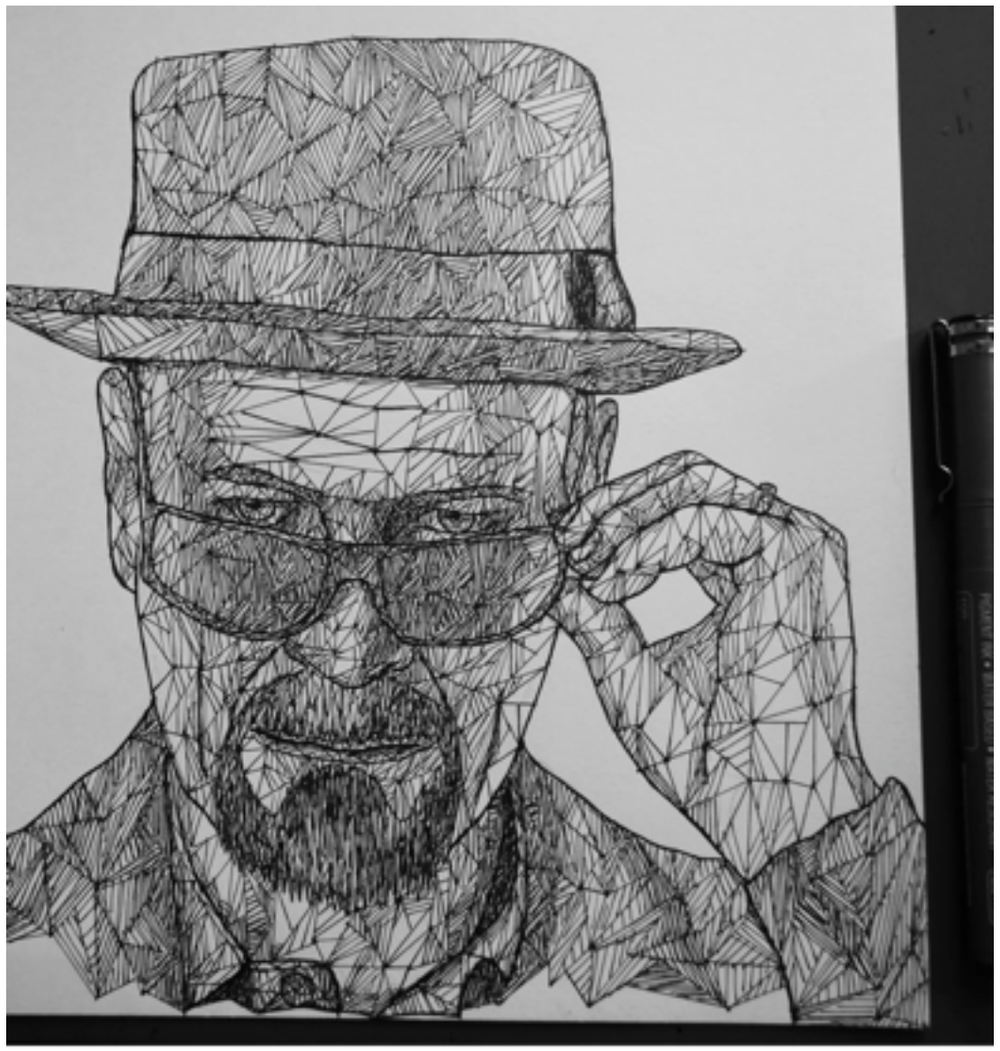
- CG deals with algorithms for geometric data



polygons

Introduction

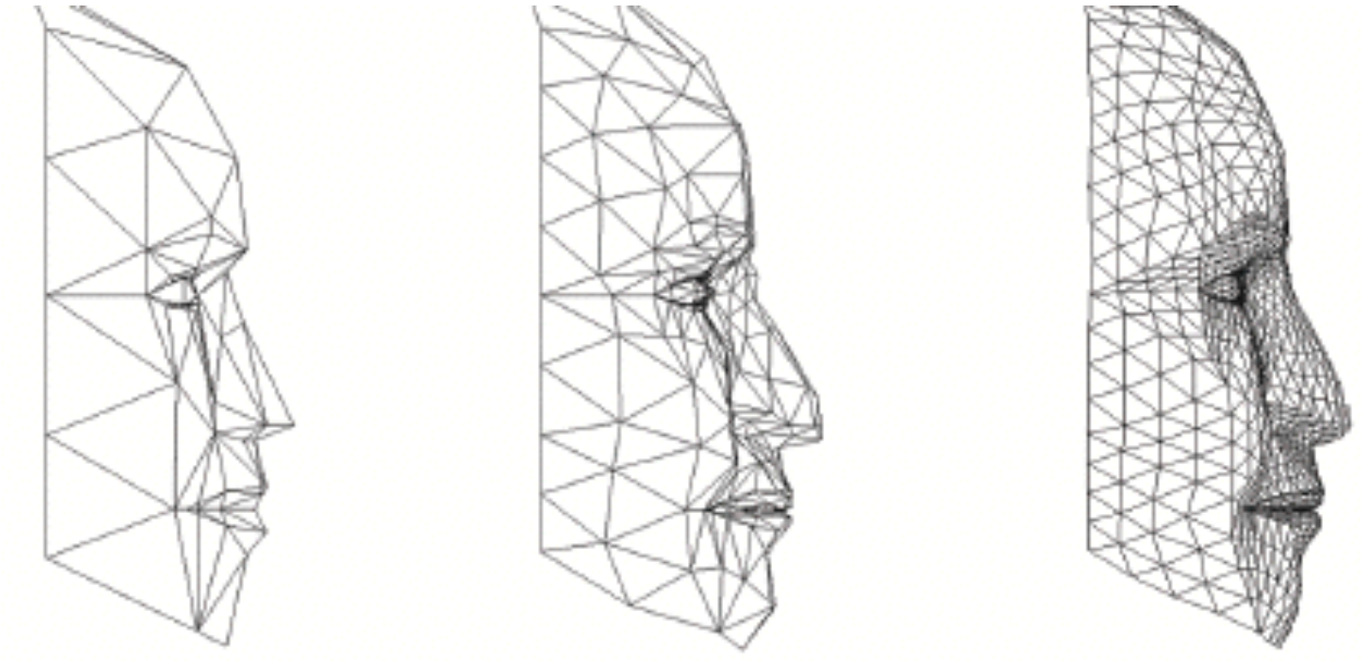
- CG deals with algorithms for geometric data



polygons

Introduction

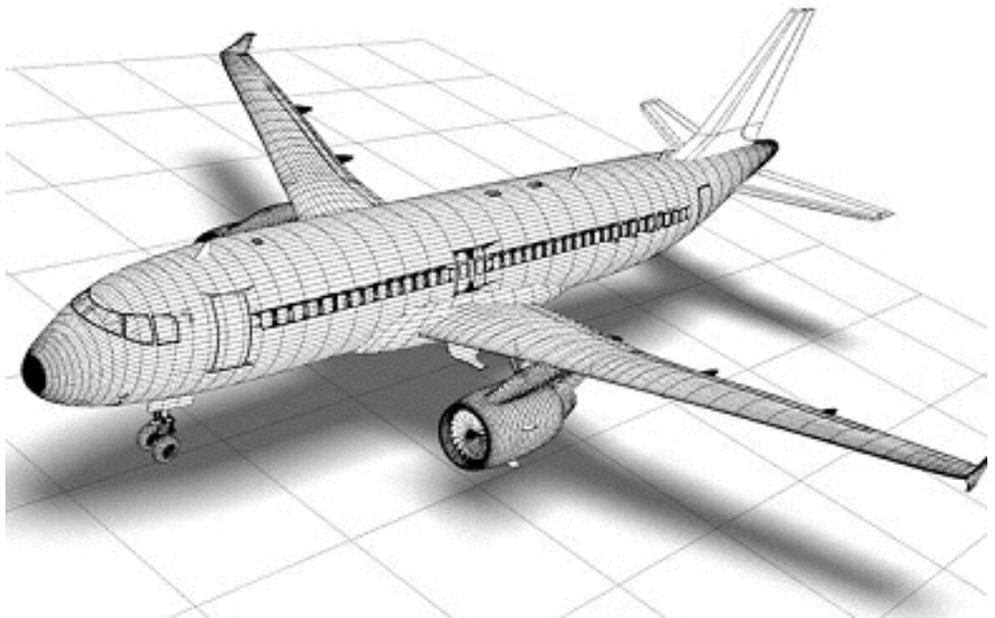
- CG deals with algorithms for geometric data



2D, 3D..

Introduction

- CG deals with algorithms for geometric data



2D, 3D..

Computational geometry

- Questions
  - Come up with an algorithm to e.g. find the convex hull of a set of points
  - What is the complexity of the convex hull of a set of n points n the plane?
  - What is the worst-case running time (tight bound) for an 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)?
  - …

Applications

- Computer graphics
  - find what objects intersect others for rendering
  - hidden surface removal
  - lighting

Applications

- Robotics
  - path planning involves finding paths that avoid obstacles; this involves finding intersections
  - does this route intersect this obstacle?

Applications

- Spatial database engines
  - e.g. Oracle spatial
  - contain specialized data structures for answering queries on geometric data
  - e.g. find all intersections between two sets of line segments (road and rivers)

Applications

- Computational geosciences
  - geographical data has coordinates (geometry)
  - overlay county maps with vegetation types
  - meshes of triangles are used to model surfaces

Applications

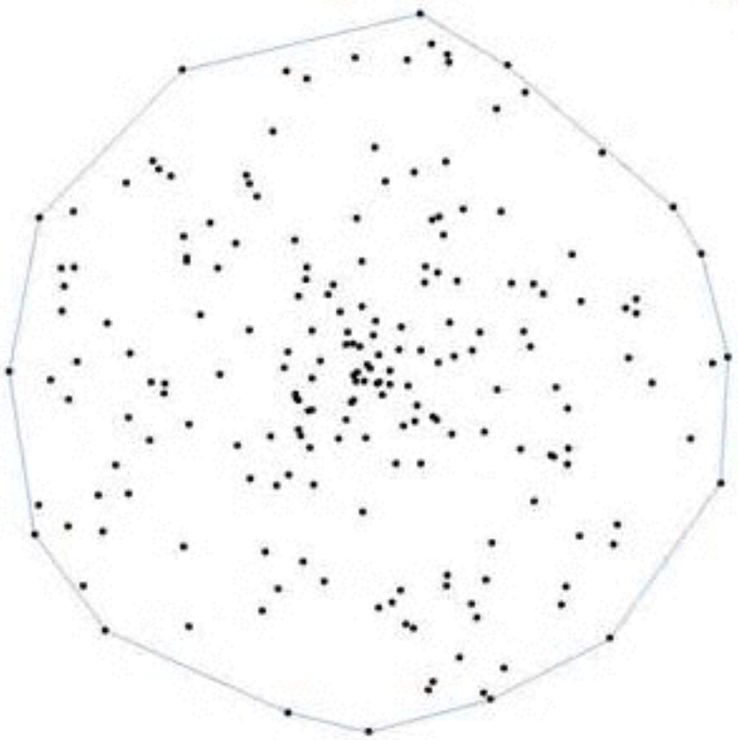
- Mobile networks / sensor networks
  - collections of devices that can communicate, and may be moving
  - e.g. keep track of their convex hulls as devices are moving
  - e.g. sending messages from one device to another requires e.g. planning a route for the message (combination of CG and graph theory)
  - chose a minimum set of sensors to turn on to assure that every point in the area is covered by a sensor.

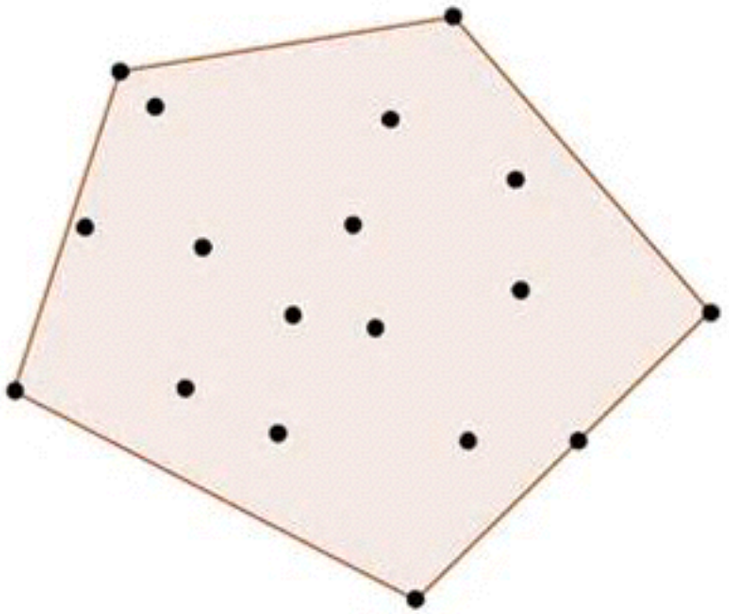
Applications

- Cell phone data
  - stream of coordinates
  - e.g. find congestion patterns, model real-time traffic conditions (done by cell phone apps)

Class overview

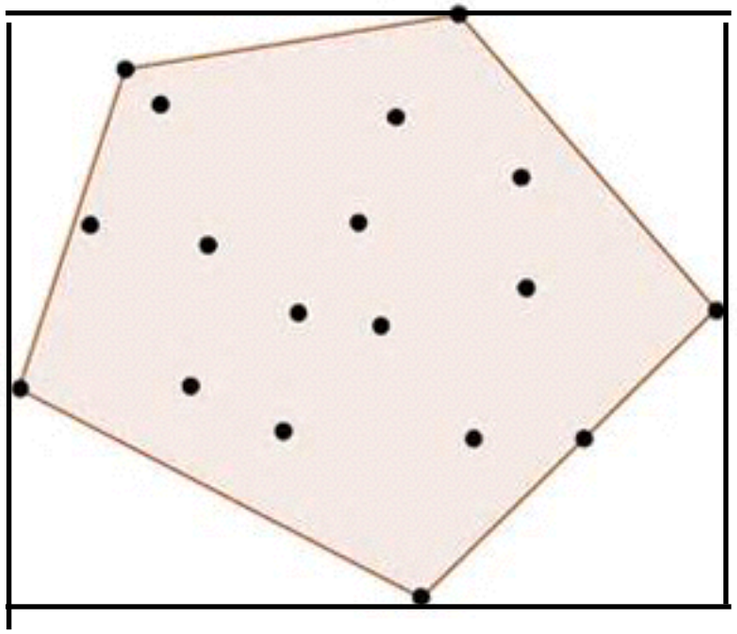
- Convex hull

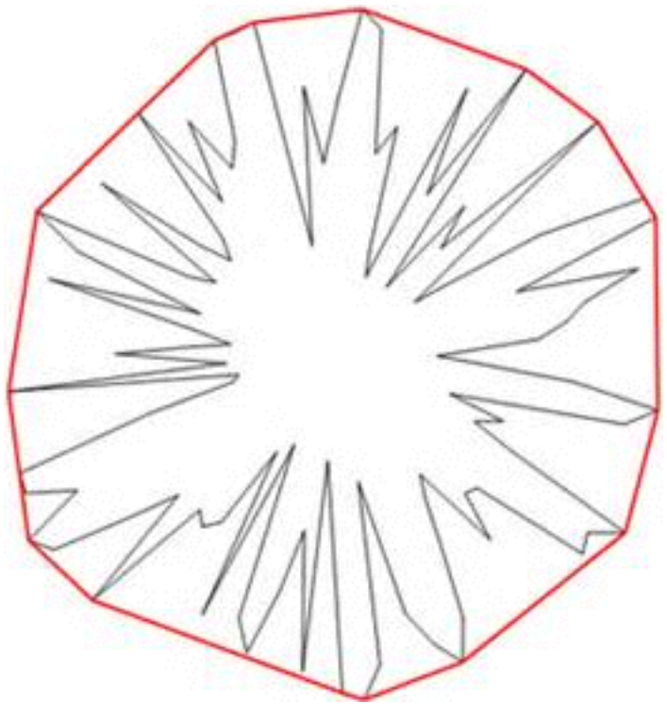




Class overview

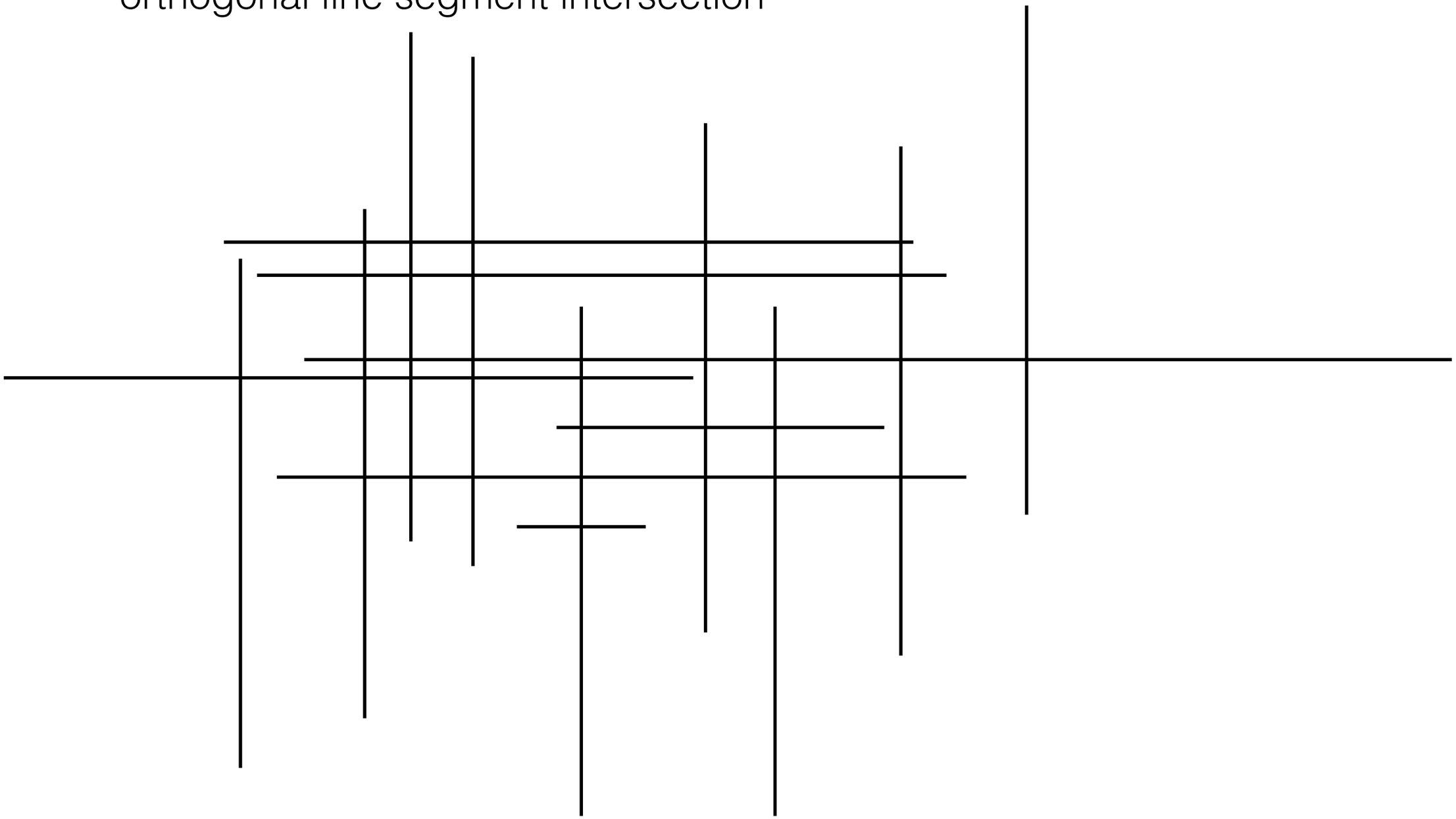
- Convex hull
  - comes up in a lot of applications
  - objects are approximated by their CH shape





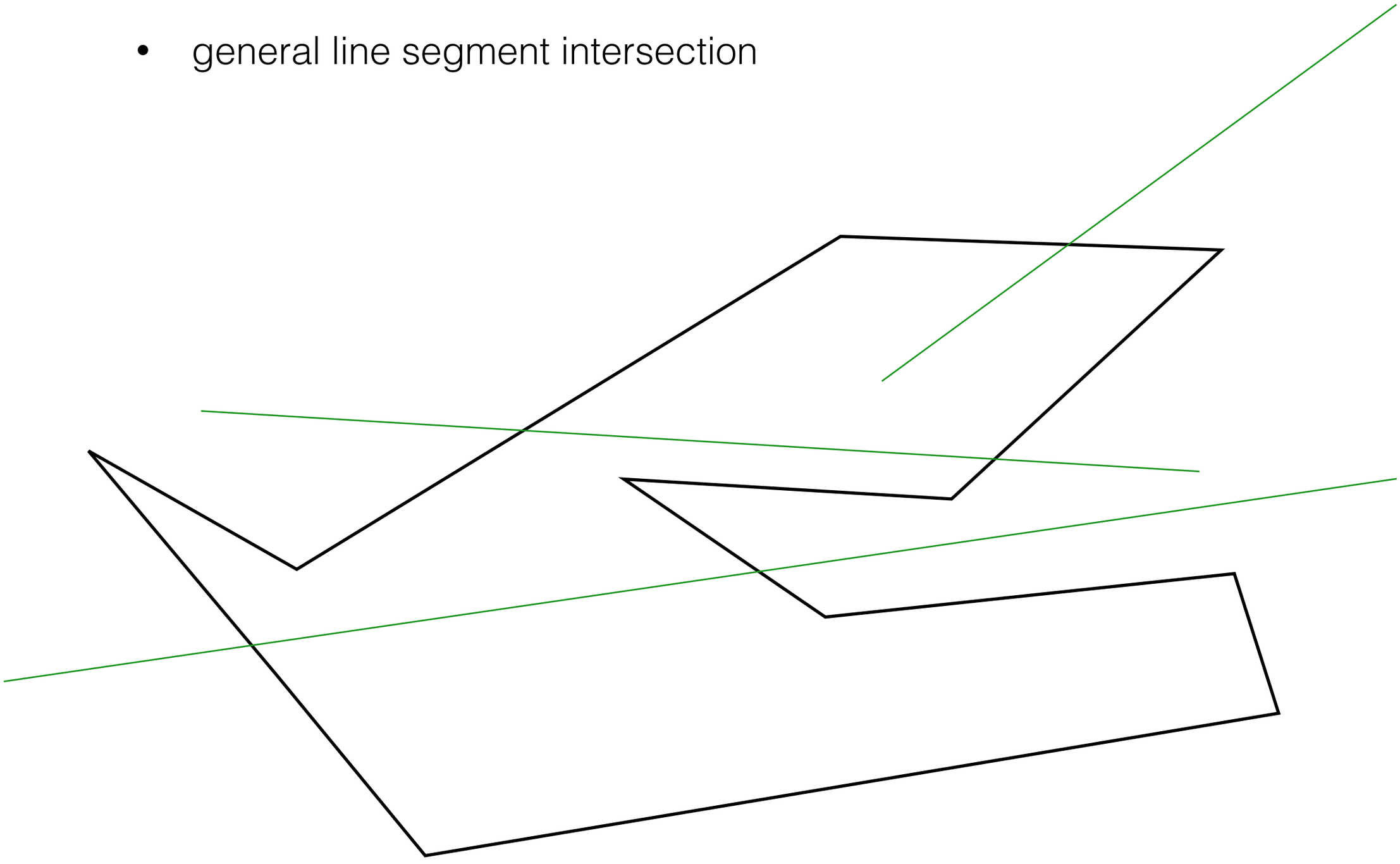
Class overview

- Intersections
  - orthogonal line segment intersection



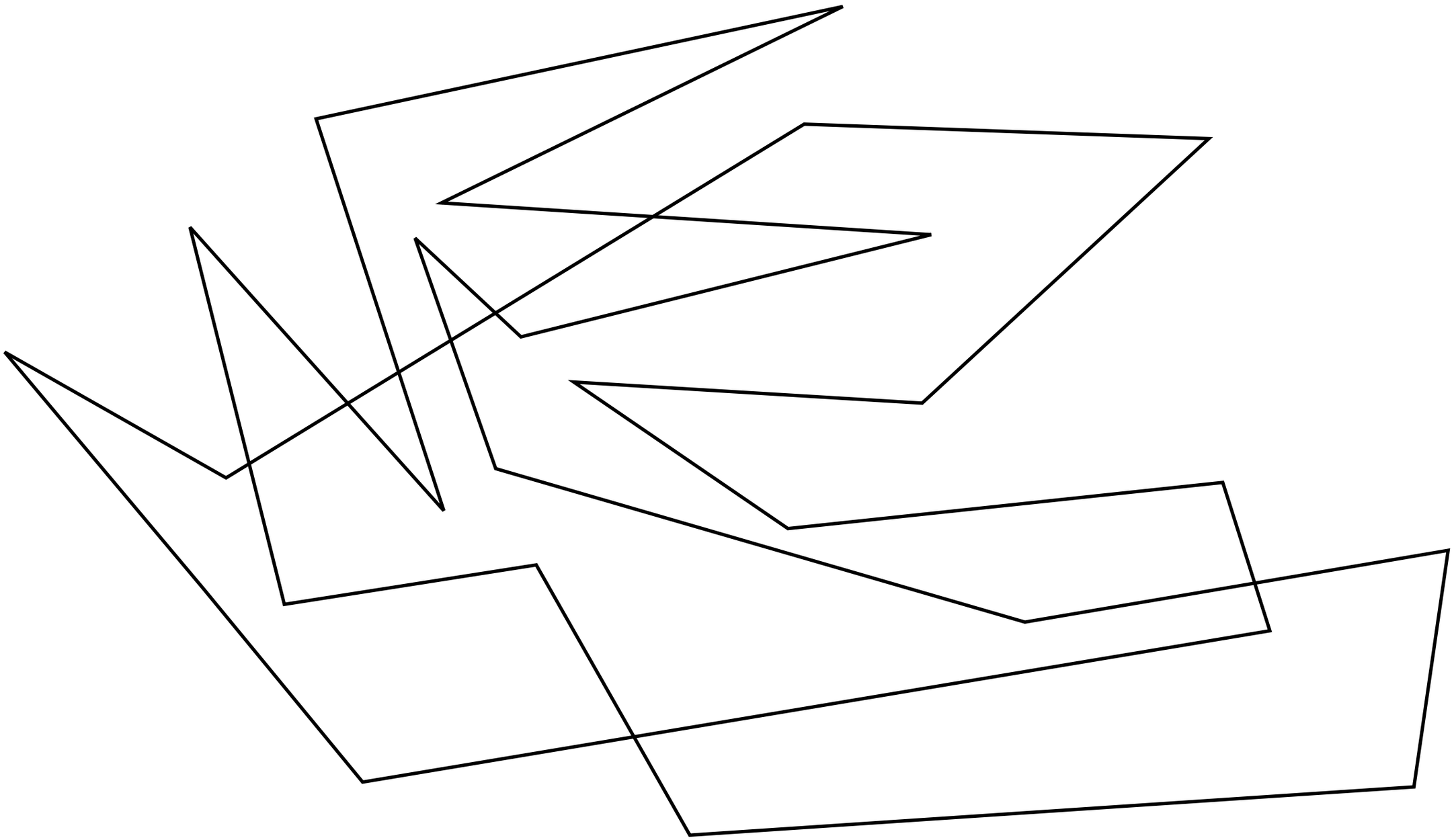
Class overview

- Intersections
  - general line segment intersection



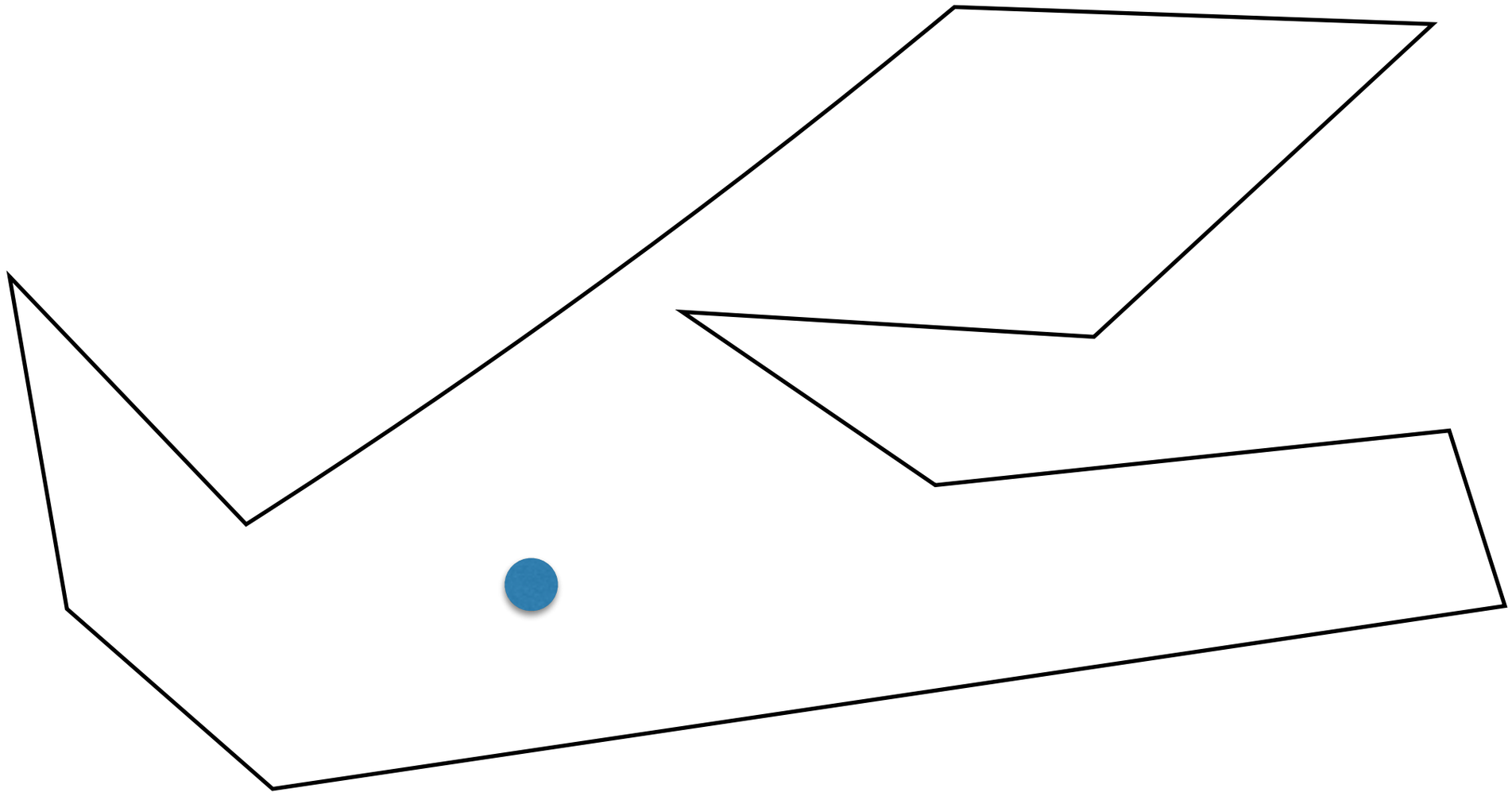
Class overview

- Intersections
  - general line segment intersection



Class overview

- Visibility
  - art gallery problem

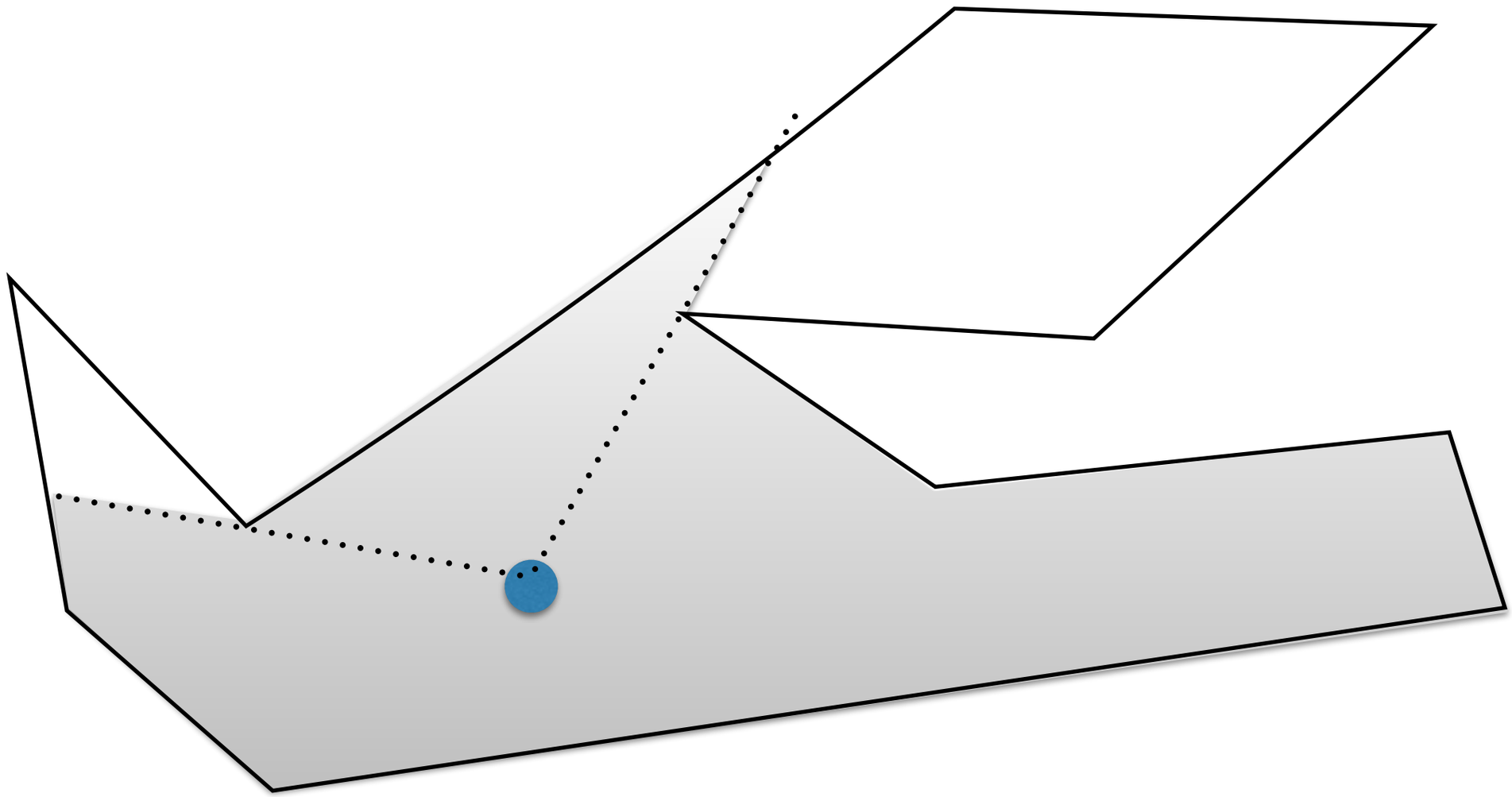


What part of the polygon can the guard see?

How many guards necessary to cover this polygon?

Class overview

- Visibility
  - art gallery problem



What part of the polygon can the guard see?

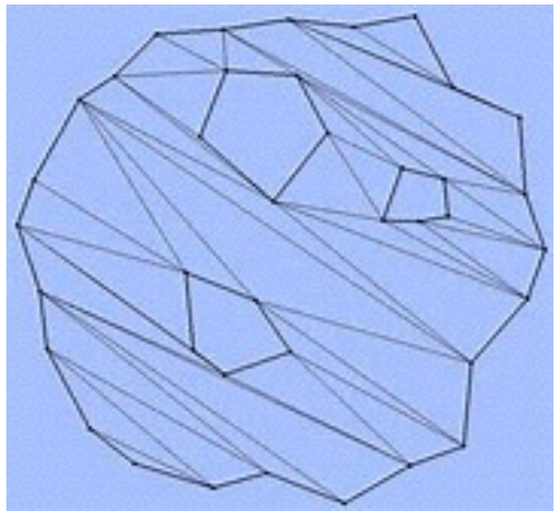
How many guards necessary to cover this polygon?

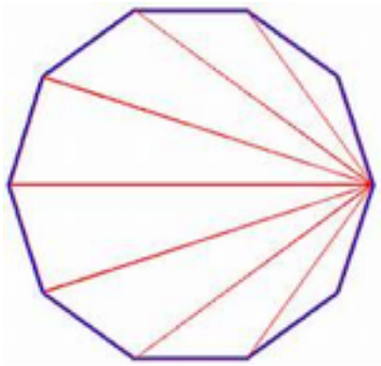
Class overview

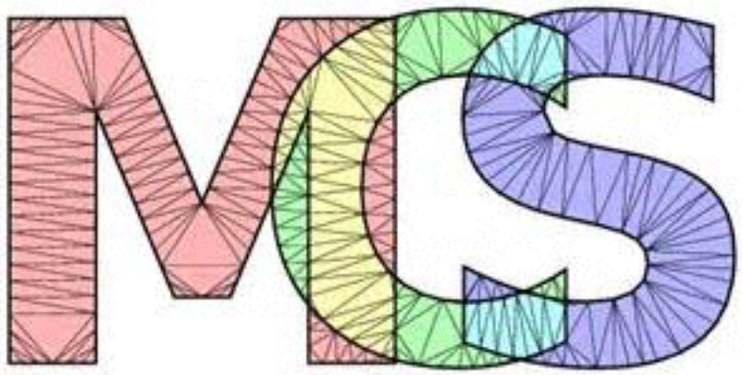
- Triangulation and partitioning
  - subdivide a complex domain into simpler objects
  - simplest object: triangulation

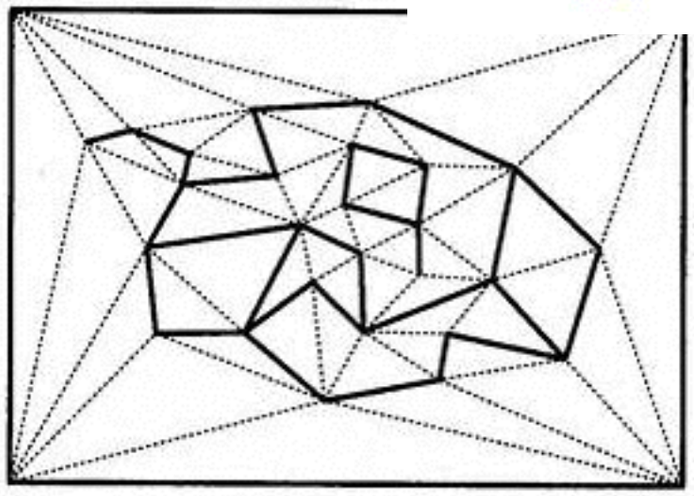
Class overview

- Triangulation and partitioning
  - polygon triangulation: divide a polygon into a set of triangles



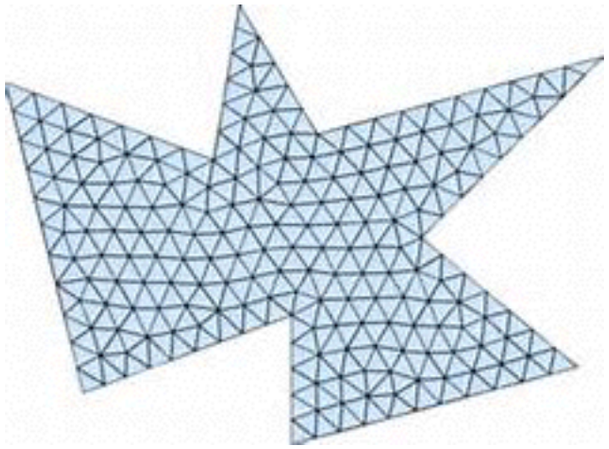


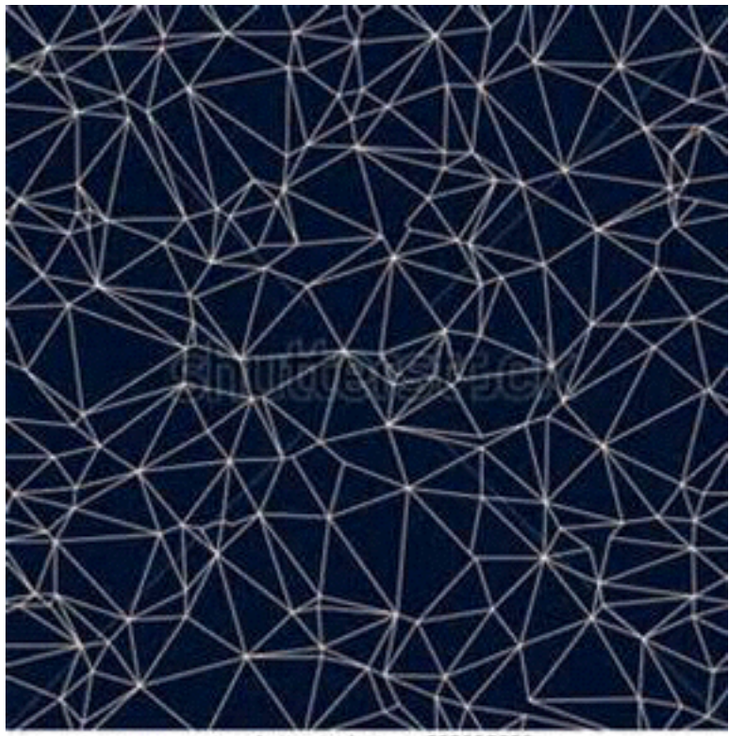




Class overview

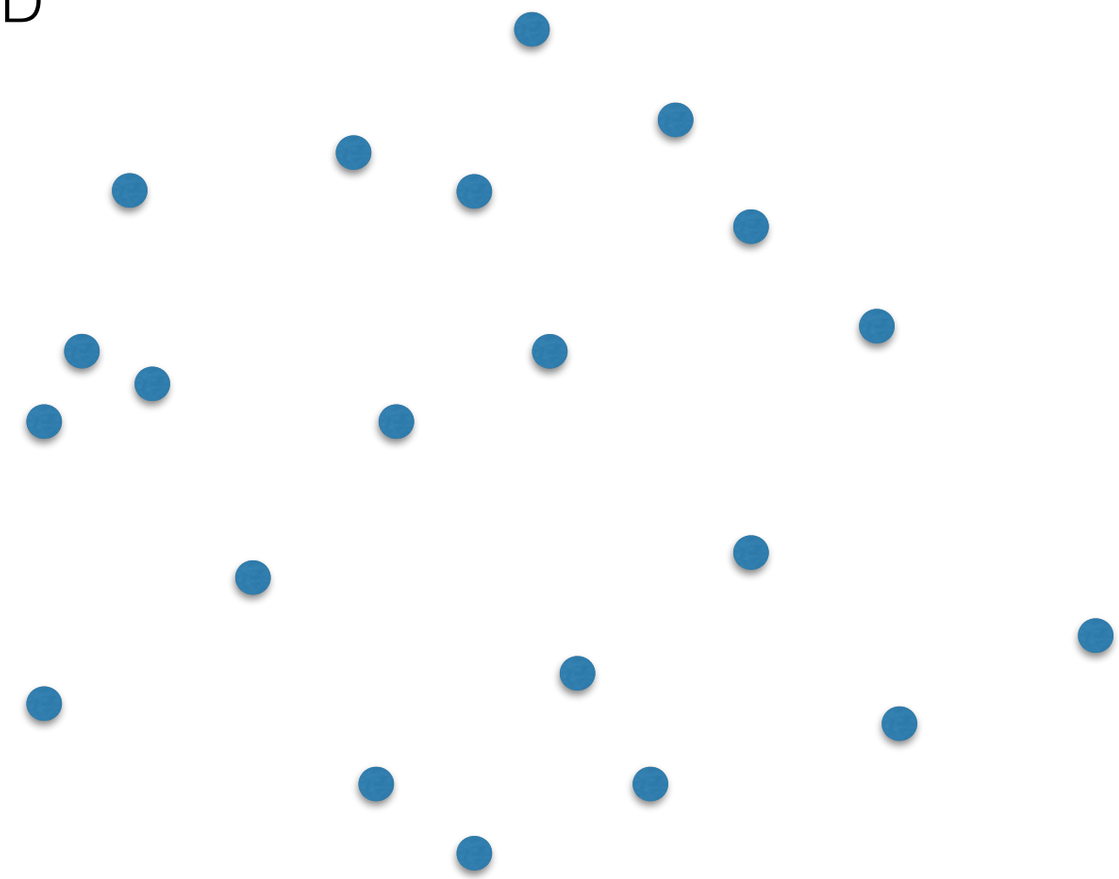
- Triangulation and partitioning
  - triangulation of a set of points





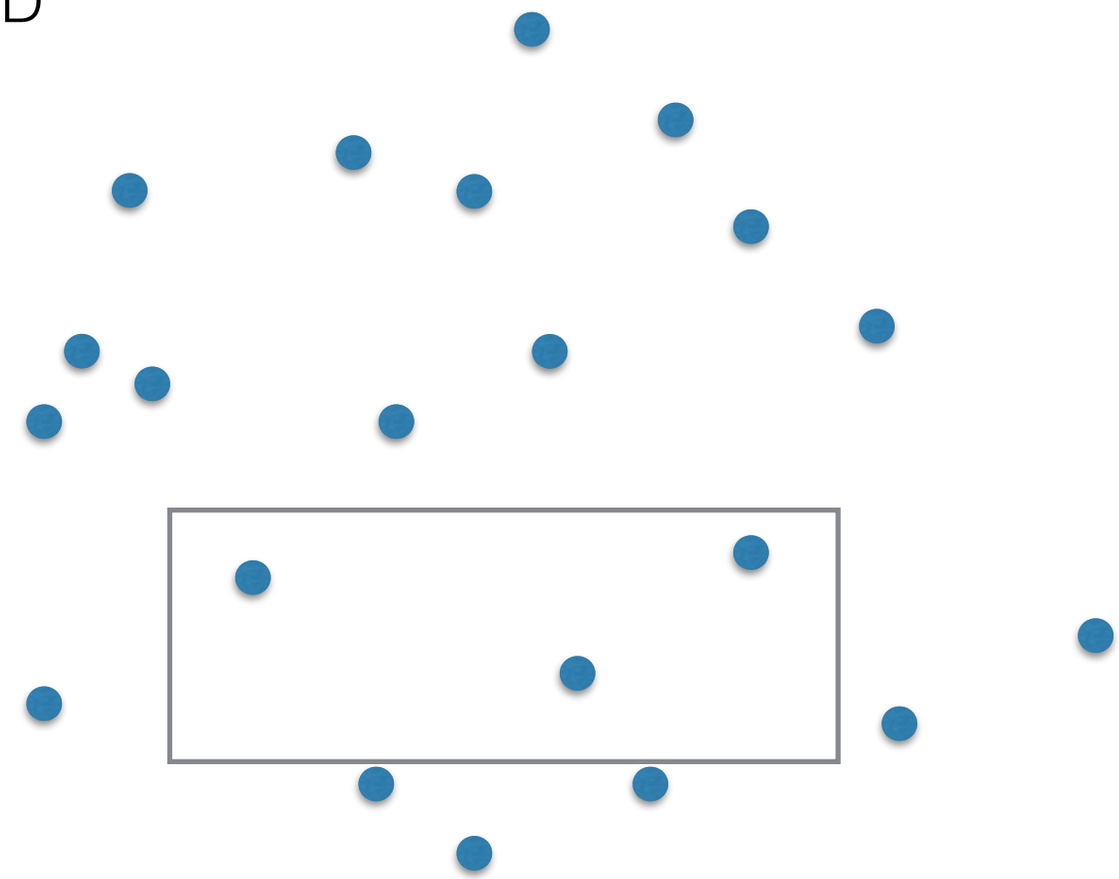
Class overview

- Range searching
  - fundamental problem
  - searching in 2D



Class overview

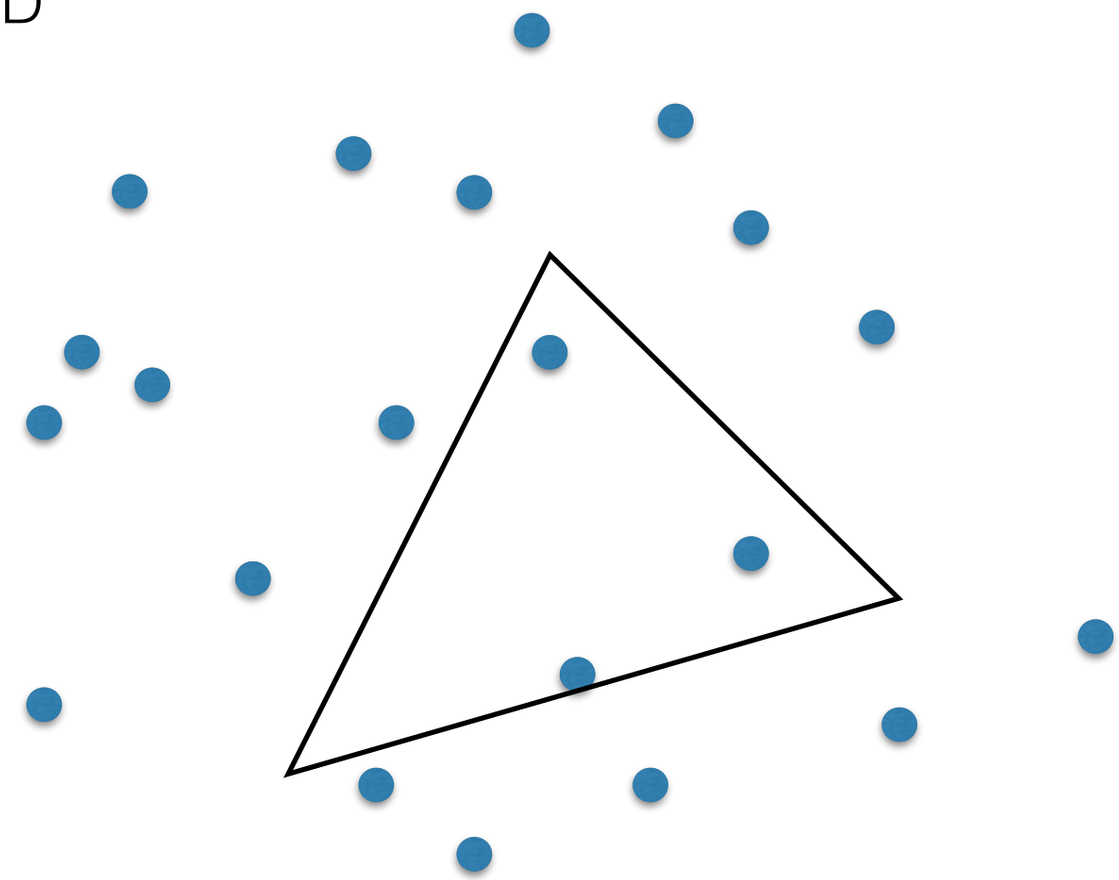
- Range searching
  - fundamental problem
  - searching in 2D



find all points in this range

Class overview

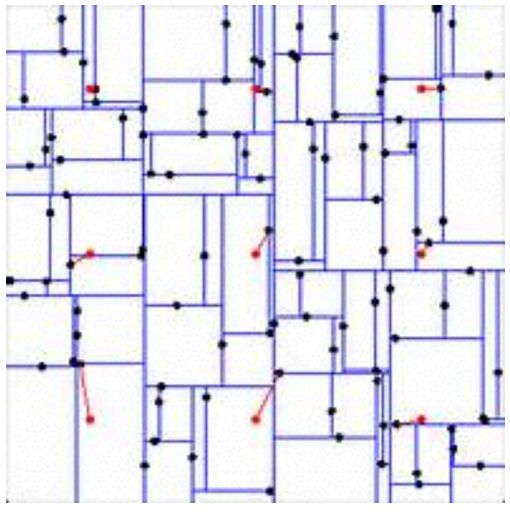
- Range searching
  - fundamental problem
  - searching in 2D



find all points in this range

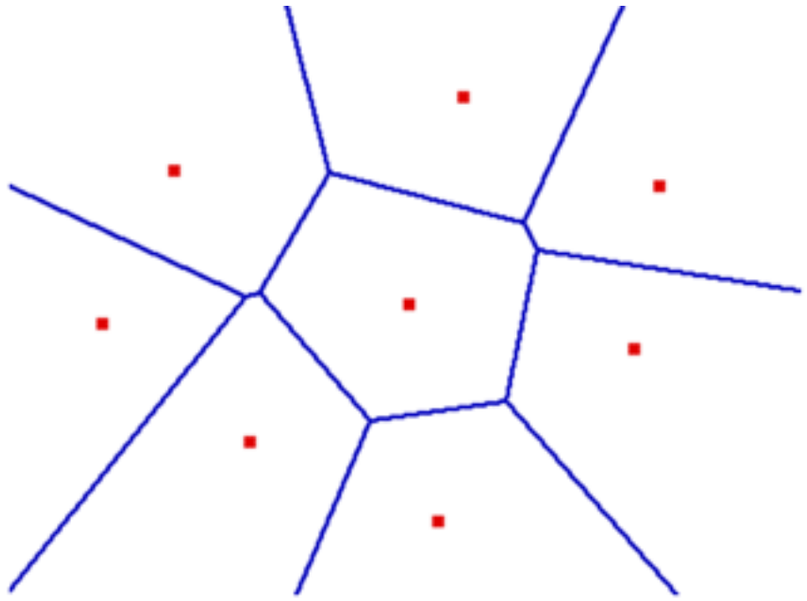
Class overview

- Range searching
  - range tree
  - kd-tree



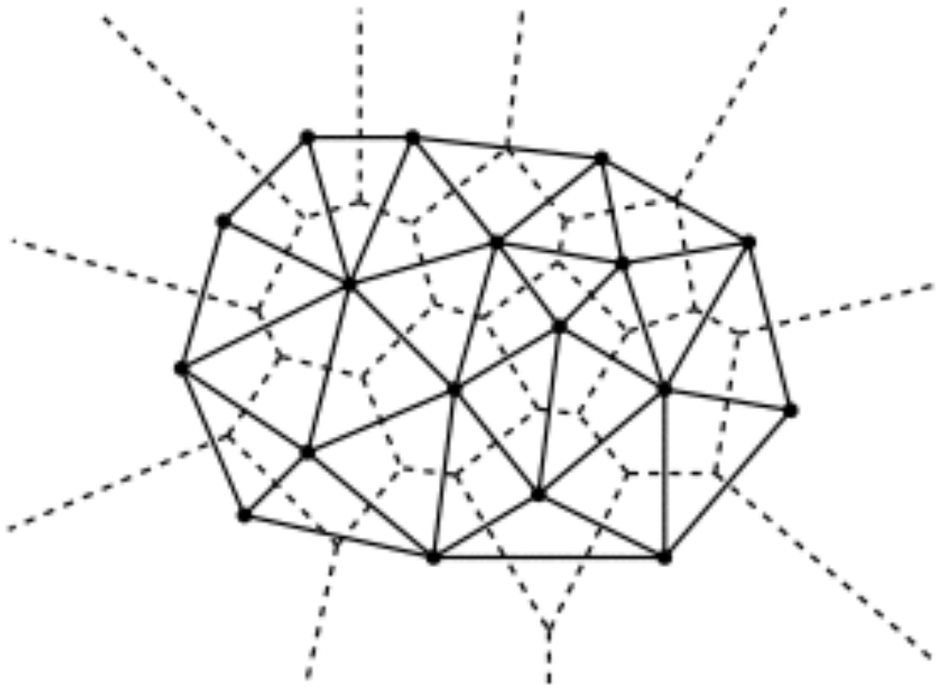
Class overview

- Proximity problems
  - Voronoi diagram
    - applications: nearest neighbor



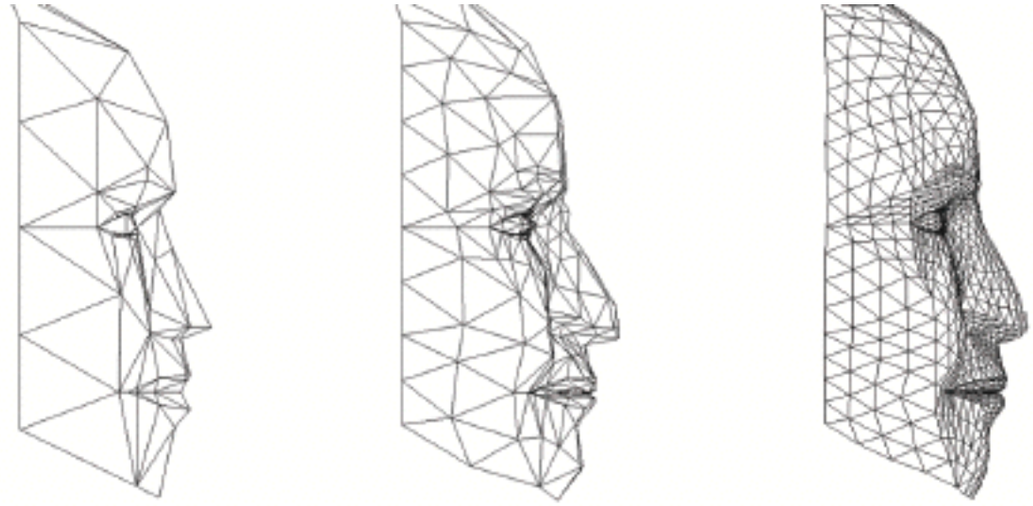
Class overview

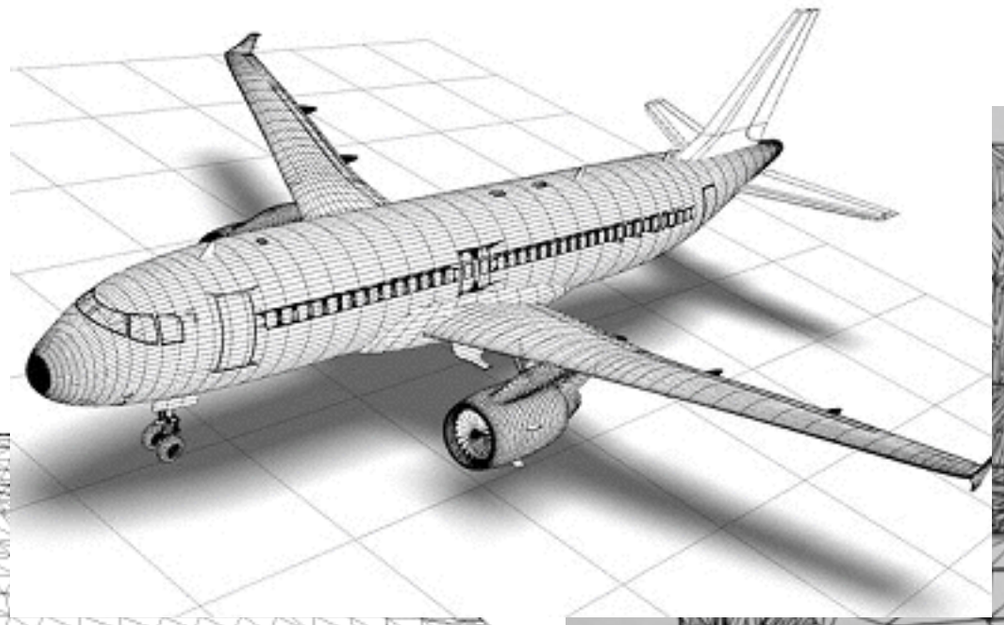
- Proximity problems
  - Voronoi diagram
  - Delaunay triangulations

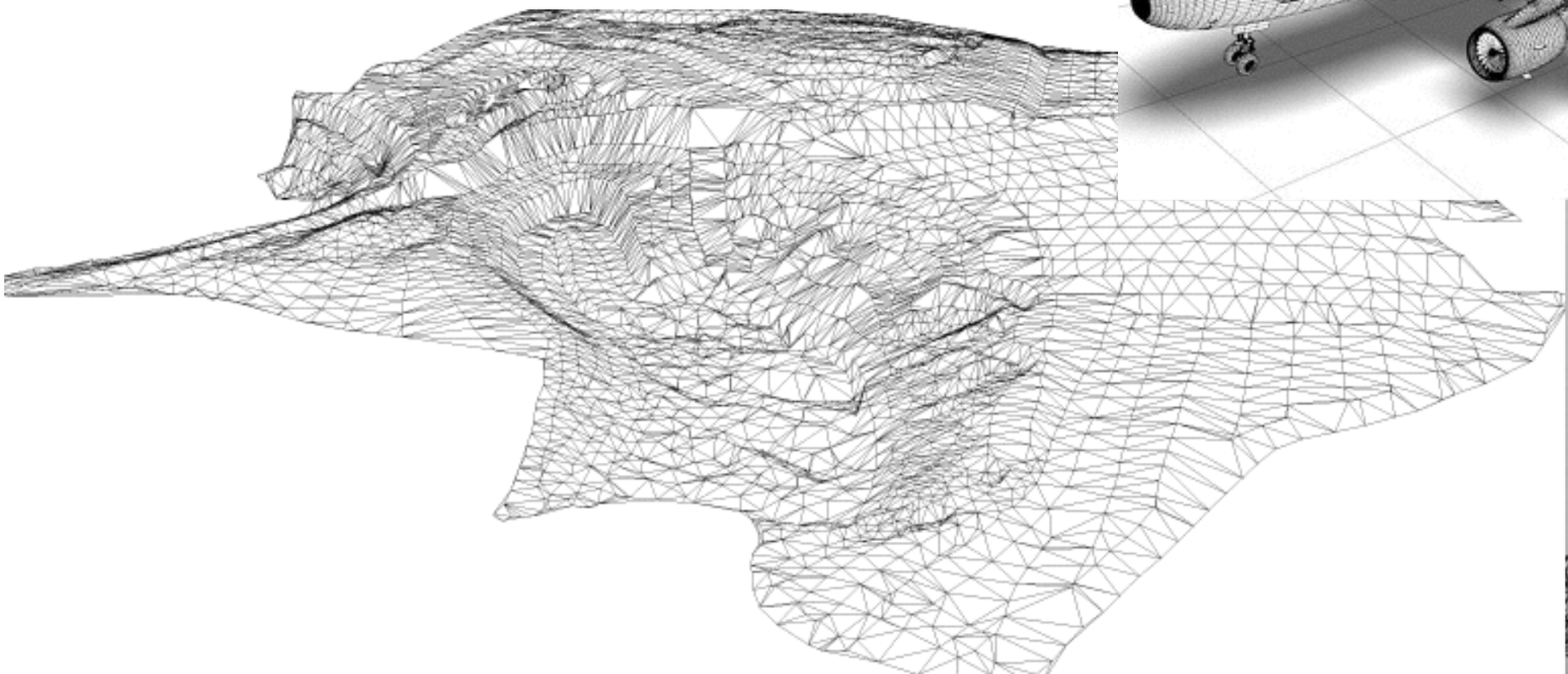


The Delaunay triangulation of a set of points (solid lines) and the Voronoi diagram (broken lines).

DT Applications

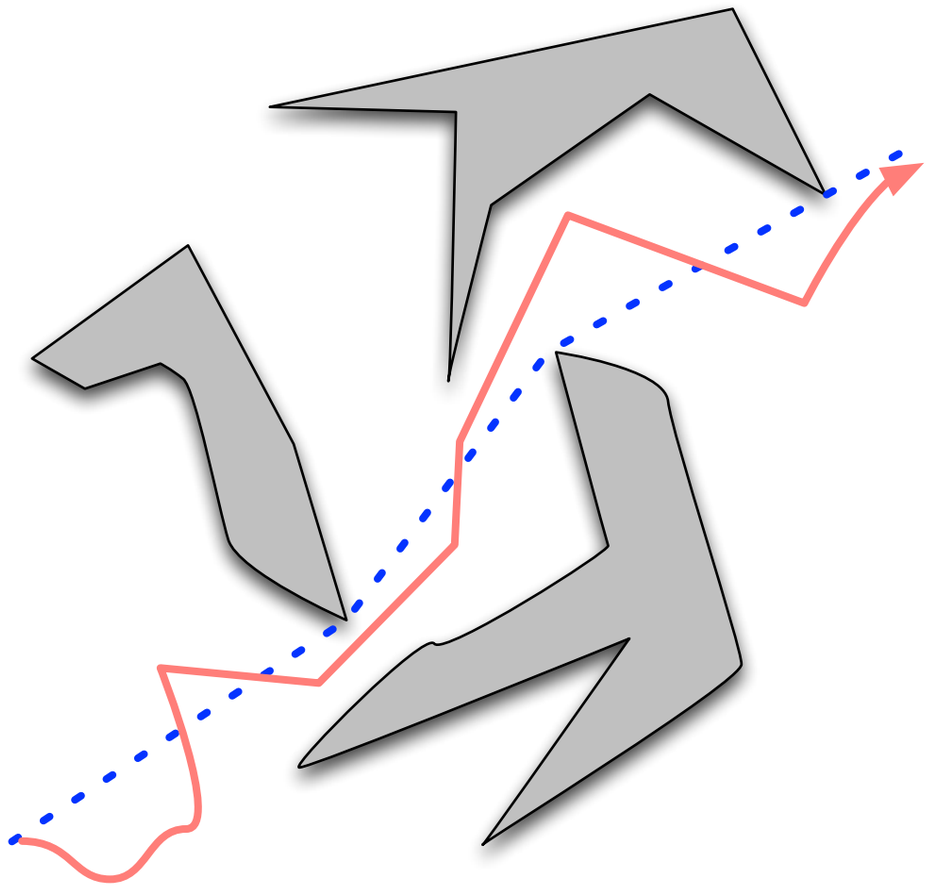






Class overview

- Motion planning
  - robots moving among obstacles, find (shortest) paths



Logistics

- Work
  - in-class group work
  - programming assignments
  - no exams

Today: warmup

Problem 1: finding collinear points

> 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:
  - n chose 2 = $O(n^2)$ 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

Can you find a solution that runs in $O(n^2 \lg n)$ time with only linear space?

Finding collinear points

Can you improve your solution, for example by making some assumption about the input?

e.g.: integer coordinates

Finding collinear points

If points have integer coordinates, we can immediately think of using hashing instead of BBST;

- families of universal hash functions are known for integers ==> guarantee no collision with high probability => O(1) insert/search

Finding collinear points

If points have integer coordinates, we can immediately think of using hashing instead of BBST;

- families of universal hash functions are known for integers ==> guarantee no collision with high probability => O(1) insert/search

Improved idea 4:

- 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