

# Computational Geometry

## Lecture 1: Introduction and convex hulls

- Plane (two-dimensional),  $\mathbb{R}^2$
- Space (three-dimensional),  $\mathbb{R}^3$
- Space (higher-dimensional),  $\mathbb{R}^d$

A **point** in the plane, 3-dimensional space, higher-dimensional space.

$$p = (p_x, p_y), p = (p_x, p_y, p_z), p = (p_1, p_2, \dots, p_d)$$

A **line** in the plane:  $y = m \cdot x + c$ ; representation by  $m$  and  $c$

A **half-plane** in the plane:  $y \leq m \cdot x + c$  or  $y \geq m \cdot x + c$

Represent vertical lines? Not by  $m$  and  $c$  ...

# Geometry: line segments

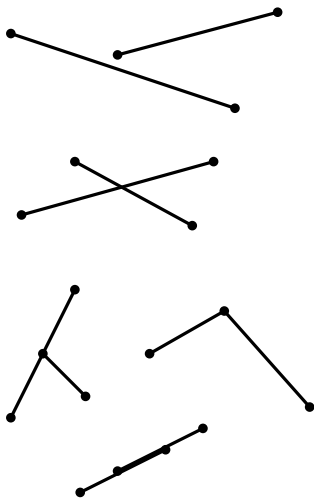
A **line segment**  $\overline{pq}$  is defined by its two endpoints  $p$  and  $q$ :

$$\begin{aligned} &(\lambda \cdot p_x + (1 - \lambda) \cdot q_x, \\ &\lambda \cdot p_y + (1 - \lambda) \cdot q_y) \end{aligned}$$

where  $0 \leq \lambda \leq 1$

Line segments are assumed to be **closed** = with endpoints, not **open**

Two line segments **intersect** if they have some point in common. It is a **proper intersection** if it is exactly one interior point of each line segment



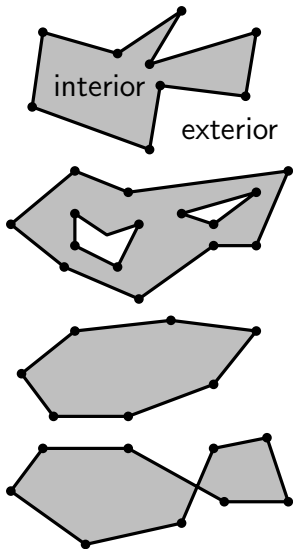
# Polygons: simple or not

A **polygon** is a connected region of the plane bounded by a sequence of line segments

- simple polygon
- polygon with holes
- convex polygon
- non-simple polygon

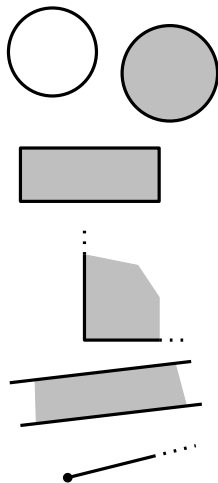
The line segments of a polygon are called its **edges**, the endpoints of those edges are the **vertices**

Some abuse: polygon is only boundary, or interior plus boundary



A **circle** is only the boundary, a **disk** is the boundary plus the interior

Rectangles, squares, quadrants, slabs, half-lines, wedges, ...



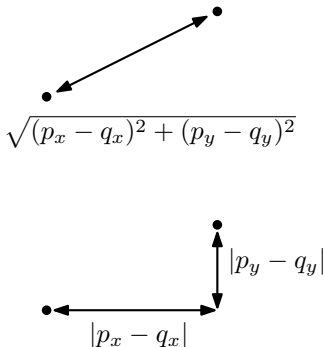
The distance between two points is generally the **Euclidean distance**:

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Another option: the **Manhattan distance**:

$$|p_x - q_x| + |p_y - q_y|$$

**Question:** What is the set of points at equal Manhattan distance to some point?



The distance between two geometric objects other than points usually refers to the minimum distance between two points that are part of these objects

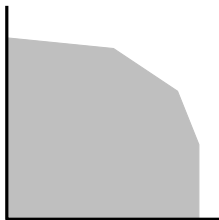
**Question:** How can the distance between two line segments be realized?

The **intersection** of two geometric objects is the set of points (part of the plane, space) they have in common



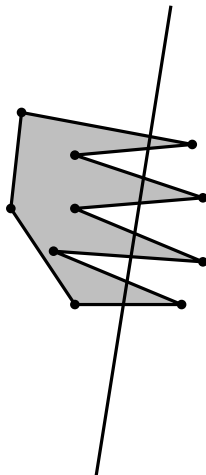
**Question 1:** How many intersection points can a line and a circle have?

**Question 2:** What are the possible outcomes of the intersection of a rectangle and a quadrant?

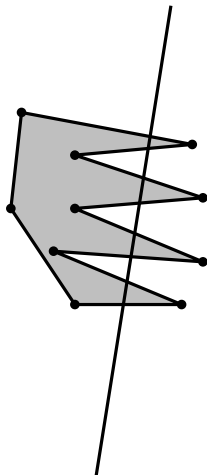




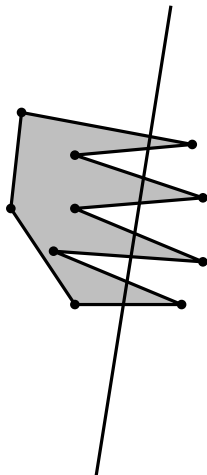
**Question 3:** What is the maximum number of intersection points of a line and a simple polygon with 10 vertices (trick question)?



**Question 4:** What is the maximum number of intersection points of a line and a simple polygon *boundary* with 10 vertices (still a trick question)?



**Question 5:** What is the maximum number of edges of a simple polygon *boundary* with 10 vertices that a line can intersect?



A point in the plane can be represented using two reals


A line in the plane can be represented using two reals and a Boolean (for example)

A line segment can be represented by two points, so four reals

A circle (or disk) requires three reals to store it (center, radius)

A rectangle requires four reals to store it

$$y = m \cdot x + c$$



false,  $m$ ,  $c$


$$x = c$$

true, ...,  $c$

A simple polygon in the plane can be represented using  $2n$  reals if it has  $n$  vertices (and necessarily,  $n$  edges)

A set of  $n$  points requires  $2n$  reals

A set of  $n$  line segments requires  $4n$  reals

A point, line, circle, ... requires  $O(1)$ , or constant, storage.

A simple polygon with  $n$  vertices requires  $O(n)$ , or linear, storage

Any computation (distance, intersection) on two objects of  $O(1)$  description size takes  $O(1)$  time!

**Question:** Suppose that a simple polygon with  $n$  vertices is given; the vertices are given in counterclockwise order along the boundary. Give an efficient algorithm to determine all edges that are intersected by a given line.

How efficient is your algorithm? Why is your algorithm efficient?

Recall from your algorithms and data structures course:

A set of  $n$  real numbers can be sorted in  $O(n \log n)$  time

A set of  $n$  real numbers can be stored in a data structure that uses  $O(n)$  storage and that allows searching, insertion, and deletion in  $O(\log n)$  time per operation

These are fundamental results in 1-dimensional computational geometry!



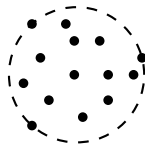
In computational geometry, problems on input with more than constant description size are the ones of interest

**Computational geometry (theory):** Study of geometric problems on geometric data, and how efficient geometric algorithms that solve them can be

**Computational geometry (practice):** Study of geometric problems that arise in various applications and how geometric algorithms can help to solve well-defined versions of such problems



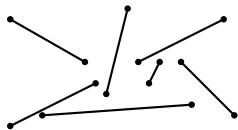
Computational geometry (theory):  
Classify abstract geometric problems  
into classes depending on how  
efficiently they can be solved



smallest enclosing circle



closest pair



any intersection?

find all intersections

Application areas that require geometric algorithms are computer graphics, motion planning and robotics, geographic information systems, CAD/CAM, statistics, physics simulations, databases, games, multimedia retrieval, ...

- Computing shadows from virtual light sources
- Spatial interpolation from groundwater pollution measurements
- Computing a collision-free path between obstacles
- Computing similarity of two shapes for shape database retrieval

**Early 70s:** First attention for geometric problems from algorithms researchers

**1976:** First PhD thesis in computational geometry (Michael Shamos)

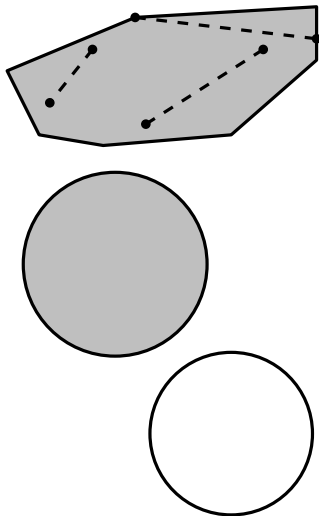
**1985:** First Annual ACM Symposium on Computational Geometry. Also: first textbook

**1996:** CGAL: first serious implementation effort for robust geometric algorithms

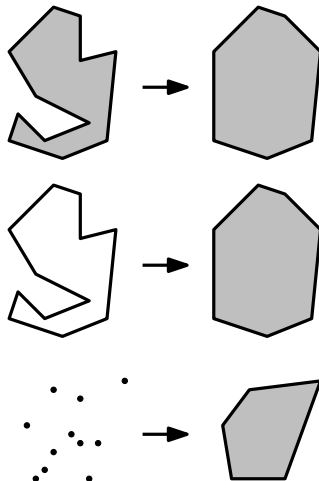
**1997:** First handbook on computational geometry (second one in 2000)

A shape or set is **convex** if for any two points that are part of the shape, the whole connecting line segment is also part of the shape

**Question:** Which of the following shapes are convex? Point, line segment, line, circle, disk, quadrant?



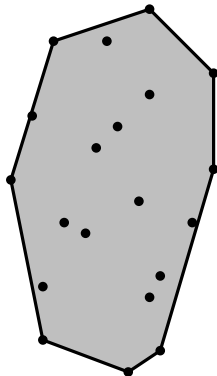
For any subset of the plane (set of points, rectangle, simple polygon), its **convex hull** is the smallest convex set that contains that subset



Give an algorithm that computes the convex hull of any given set of  $n$  points in the plane efficiently

The **input** has  $2n$  coordinates, so  $O(n)$  size

**Question:** Why can't we expect to do any better than  $O(n)$  time?



Assume the  $n$  points are distinct

The **output** has at least 4 and at most  $2n$  coordinates, so it has size between  $O(1)$  and  $O(n)$

The output is a convex polygon so it should be returned as a sorted sequence of the points, clockwise (CW) along the boundary

**Question:** Is there any hope of finding an  $O(n)$  time algorithm?

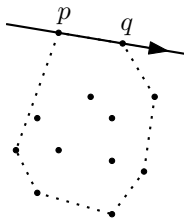
# Developing an algorithm

To develop an algorithm, find useful *properties*, make various *observations*, draw many *sketches* to gain *insight*

Property: The vertices of the convex hull are always points from the input

Consequently, the edges of the convex hull connect two points of the input

Property: The supporting line of any convex hull edge has all input points to one side



all points lie right of the directed line from  $p$  to  $q$ , if the edge from  $p$  to  $q$  is a CW convex hull edge



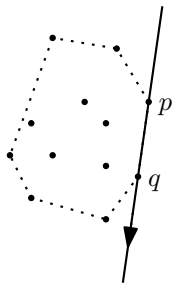
# Developing an algorithm

To develop an algorithm, find useful *properties*, make various *observations*, draw many *sketches* to gain *insight*

Property: The vertices of the convex hull are always points from the input

Consequently, the edges of the convex hull connect two points of the input

Property: The supporting line of any convex hull edge has all input points to one side



all points lie right of the directed line from  $p$  to  $q$ , if the edge from  $p$  to  $q$  is a CW convex hull edge

## **Algorithm** SLOWCONVEXHULL( $P$ )

*Input.* A set  $P$  of points in the plane.

*Output.* A list  $\mathcal{L}$  containing the vertices of  $CH(P)$  in clockwise order.

1.  $E \leftarrow \emptyset$ .
2. **for** all ordered pairs  $(p, q) \in P \times P$  with  $p$  not equal to  $q$
3.     **do**  $valid \leftarrow \mathbf{true}$
4.         **for** all points  $r \in P$  not equal to  $p$  or  $q$
5.             **do if**  $r$  lies left of the directed line from  $p$  to  $q$
6.                 **then**  $valid \leftarrow \mathbf{false}$
7.             **if**  $valid$  **then** Add the directed edge  $\vec{pq}$  to  $E$
8. From the set  $E$  of edges construct a list  $L$  of vertices of  $CH(P)$ , sorted in clockwise order.

**Question:** How must line 5 be interpreted to make the algorithm correct?

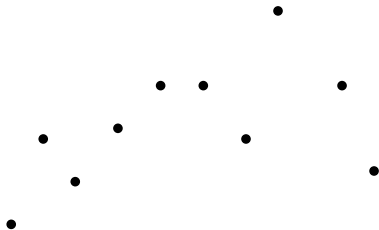
**Question:** How efficient is the algorithm?

Another approach: incremental, from left to right

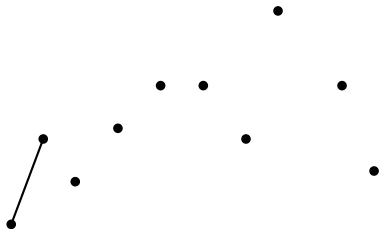
Let's first compute the *upper boundary* of the convex hull this way (property: on the upper hull, points appear in  $x$ -order)

Main idea: Sort the points from left to right (= by  $x$ -coordinate). Then insert the points in this order, and maintain the upper hull so far

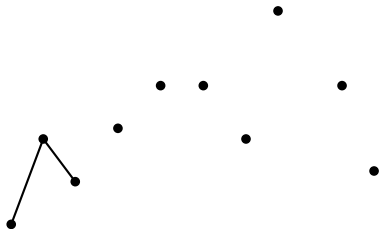
Observation: from left to right, there are only right turns on the upper hull



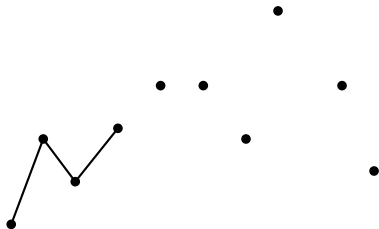
Initialize by inserting the  
leftmost two points



If we add the third point there will be a right turn at the previous point, so we add it

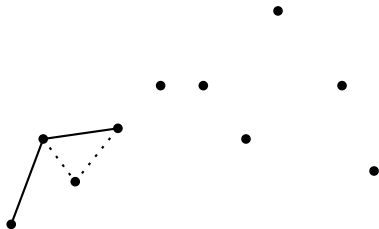


If we add the fourth point we  
get a left turn at the third  
point

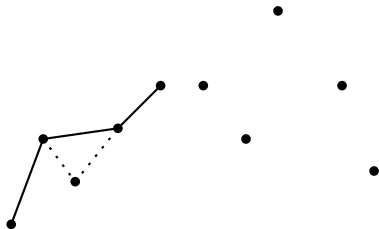




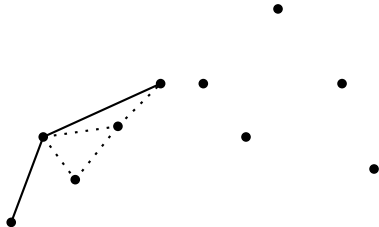
... so we remove the third point from the upper hull when we add the fourth



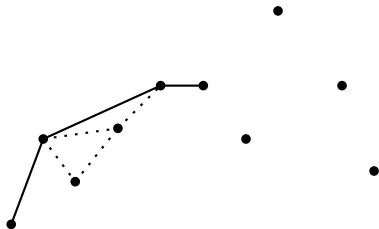
If we add the fifth point we get  
a left turn at the fourth point



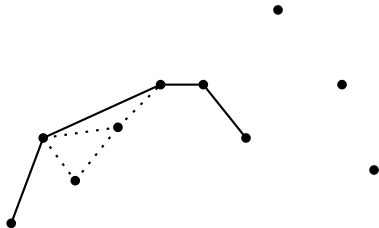
... so we remove the fourth point when we add the fifth



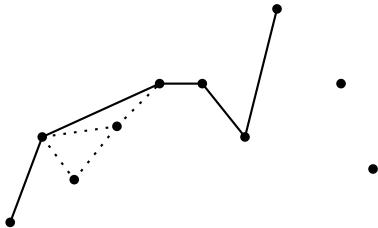
If we add the sixth point we get a right turn at the fifth point, so we just add it



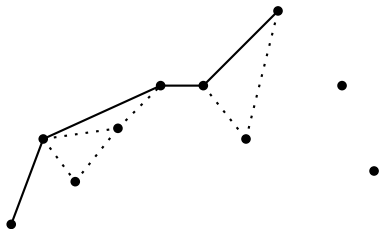
We also just add the seventh point



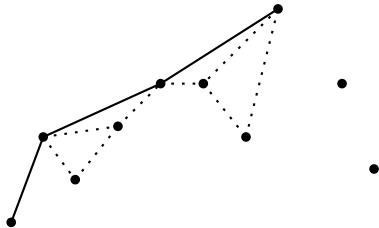
When adding the eight point  
... we must remove the  
seventh point



... we must remove the seventh point



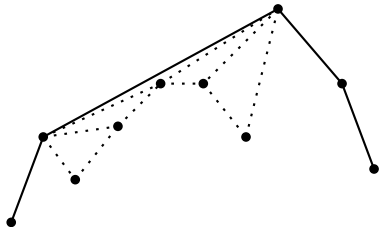
... and also the sixth point







After two more steps we get:



## Algorithm CONVEXHULL( $P$ )

*Input.* A set  $P$  of points in the plane.

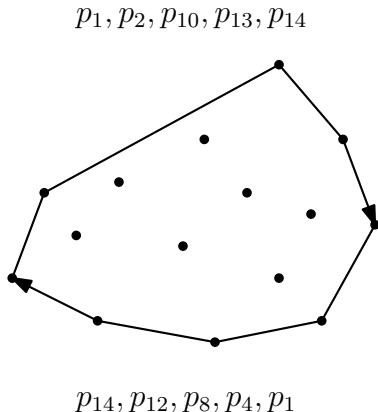
*Output.* A list containing the vertices of  $CH(P)$  in clockwise order.

1. Sort the points by  $x$ -coordinate, resulting in a sequence  $p_1, \dots, p_n$ .
2. Put the points  $p_1$  and  $p_2$  in a list  $L_{\text{upper}}$ , with  $p_1$  as the first point.
3. **for**  $i \leftarrow 3$  **to**  $n$
4.     **do** Append  $p_i$  to  $L_{\text{upper}}$ .
5.     **while**  $L_{\text{upper}}$  contains more than two points **and** the last three points in  $L_{\text{upper}}$  do not make a right turn
6.     **do** Delete the middle of the last three points from  $L_{\text{upper}}$ .

Then we do the same for the lower convex hull, from right to left

We remove the first and last points of the lower convex hull

... and concatenate the two lists into one



Algorithm analysis generally has two components:

- proof of correctness
- efficiency analysis, proof of running time

Are the **general observations** on which the algorithm is based correct?

Does the algorithm handle **degenerate cases** correctly?

Here:

- Does the sorted order matter if two or more points have the same  $x$ -coordinate?
- What happens if there are three or more collinear points, in particular on the convex hull?

Identify of **each line** of pseudo-code how much time it takes, if it is executed once (note: operations on a constant number of constant-size objects take constant time)

Consider the **loop-structure** and examine how often each line of pseudo-code is executed

Sometimes there are **global arguments** why an algorithm is more efficient than it seems, at first

## Algorithm CONVEXHULL( $P$ )

*Input.* A set  $P$  of points in the plane.

*Output.* A list containing the vertices of  $CH(P)$  in clockwise order.

1. Sort the points by  $x$ -coordinate, resulting in a sequence  $p_1, \dots, p_n$ .
2. Put the points  $p_1$  and  $p_2$  in a list  $L_{\text{upper}}$ , with  $p_1$  as the first point.
3. **for**  $i \leftarrow 3$  **to**  $n$
4.     **do** Append  $p_i$  to  $L_{\text{upper}}$ .
5.     **while**  $L_{\text{upper}}$  contains more than two points **and** the last three points in  $L_{\text{upper}}$  do not make a right turn
6.     **do** Delete the middle of the last three points from  $L_{\text{upper}}$ .



The sorting step takes  $O(n \log n)$  time

Adding a point takes  $O(1)$  time for the adding-part. Removing points takes constant time for each removed point. If due to an addition,  $k$  points are removed, the step takes  $O(1+k)$  time

Total time:

$$O(n \log n) + \sum_{i=3}^n O(1+k_i)$$

if  $k_i$  points are removed when adding  $p_i$

Since  $k_i = O(n)$ , we get

$$O(n \log n) + \sum_{i=3}^n O(n) = O(n^2)$$

Global argument: each point can be removed only once from the upper hull

This gives us the fact:

$$\sum_{i=3}^n k_i \leq n$$

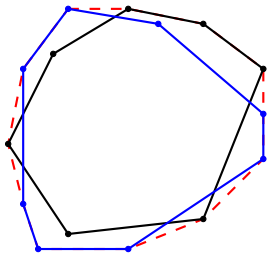
Hence,

$$O(n \log n) + \sum_{i=3}^n O(1 + k_i) = O(n \log n) + O(n) = O(n \log n)$$

The convex hull of a set of  $n$  points in the plane can be computed in  $O(n \log n)$  time, and this is optimal

Divide-and-conquer: split the point set in two halves, compute the convex hulls recursively, and merge

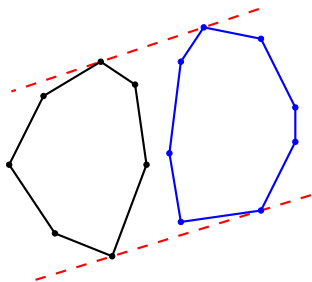
A merge involves finding “extreme vertices” in every direction



## Other approaches: divide-and-conquer

Alternatively: split the point set in two halves on  $x$ -coordinate, compute the convex hulls recursively, and merge

A merge now comes down to finding two common tangent lines



For a 3-dimensional point set, the convex hull is a convex polyhedron

It has vertices (0-dim.), edges (1-dim.), and facets (2-dim.) in its boundary, and a 3-dimensional interior

The boundary is a planar graph, so it has  $O(n)$  vertices, edges and facets



For a 4-dimensional point set, the convex hull is a convex polyhedron

It has vertices (0-dim.), edges (1-dim.), 2-facets (2-dim.), and 3-facets (3-dim.) in its boundary, and a 4-dimensional interior

Its boundary can have  $\Theta(n^2)$  facets in the worst case!