Computational Geometry

1. Introduction

1. Introduction

- Computational Geometry is concerned with the design and the analysis of algorithms and data structures for geometric problems from
  - computer graphics,
  - visualization,
  - geographic information systems (GIS),
  - computer vision and pattern recognition,
  - robotics, etc.
- Computational Geometry was established as discipline of its own right between 1975 and 1980.

1.1 Typical problems

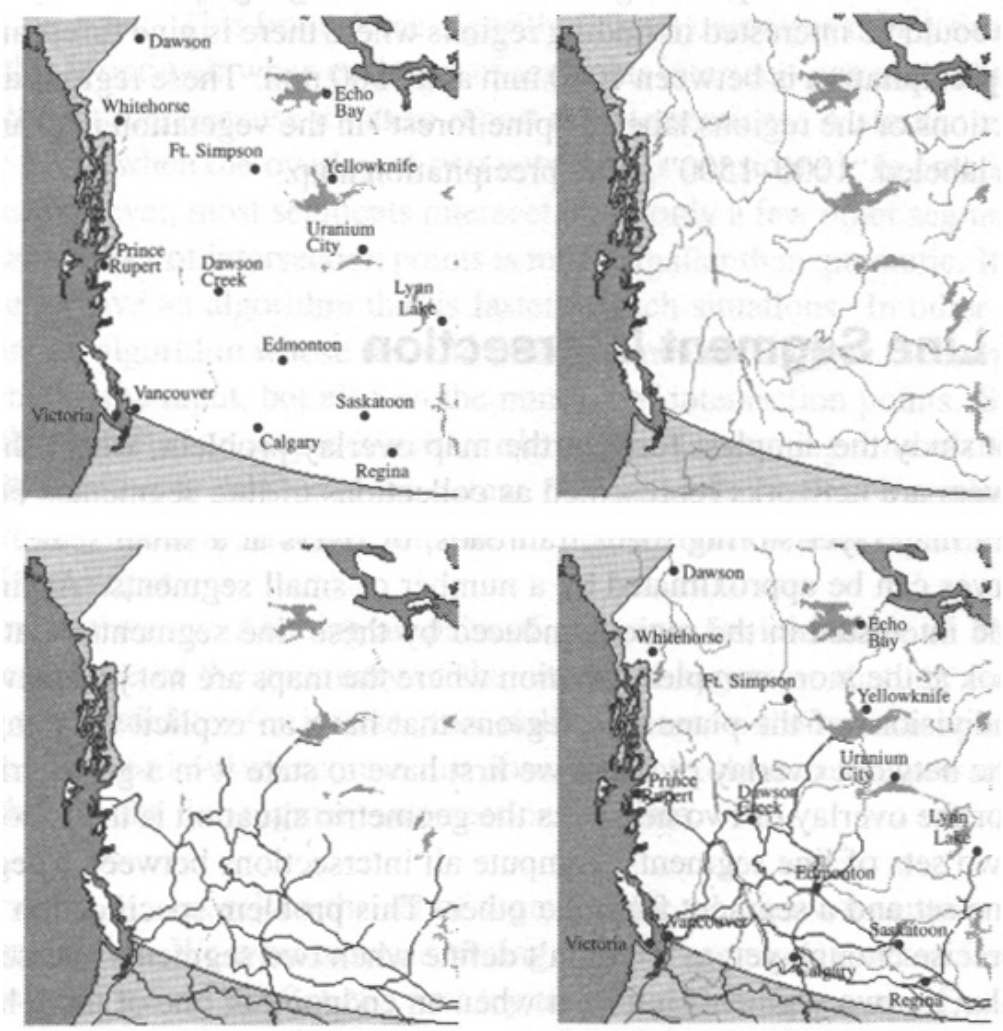
a) In a geographic information systems (GIS) the information is layered in separated maps:

- Maps for streets, rivers, train tracks, villages, land use, etc.
- Every map corresponds to one information layer.
- **Question a):** Find all bridges, i.e. compute all intersections of streets with rivers.
- If both are represented as polygon chains this requires computing all intersections of all polygon chains of both maps.
- Naïve approach: Intersect all lines representing streets with all lines representing rivers...

➡ ... infeasible for real world data.

➡ Line segment intersections: Chapter 2

1.1 Typical problems



Cities, rivers, railroads, and their overlay in western Canada

1.1 Typical problems

b) In a data base the data of the employees of a company are stored: name, date of birth, monthly salary.

Question b): Find all employees born between 1950 and 1955 with a salary between 3.000 $ and 4.000 $.

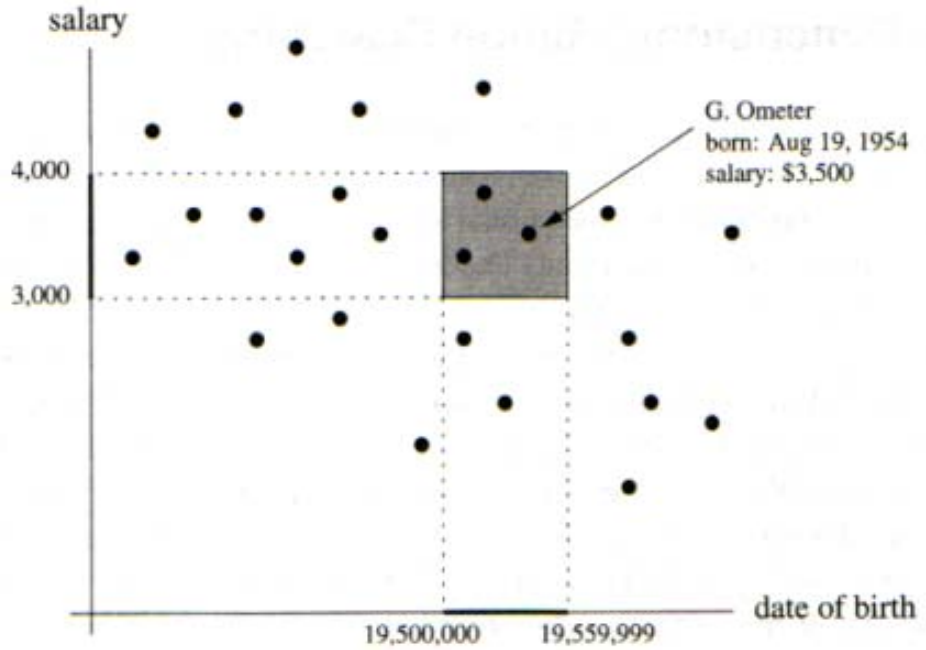
- To sort the date of birth compute the numbers

$$10.000 \text{ x year} + 100 \text{ x month} + \text{day}.$$

- In combination with the salary this yields a 2d range query:
  - The first dimension is the date of birth.
  - The second dimension is the salary.

1.1 Typical problems

Geometric interpretation of a 2d range query

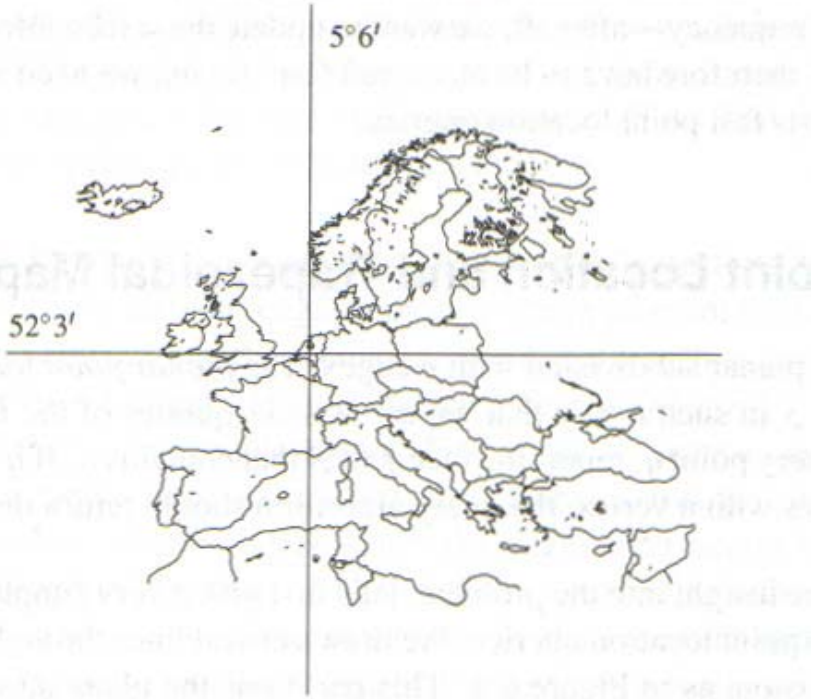


- **Remark:** Range queries run usually for very high dimensional data.

➡ Range query: Chapter 4

1.1 Typical problems

c) **Question c):** Compute from the geographic position the country in a map.



For humans this task is easy on 2d maps, but...

1.1 Typical problems

... for the computer this is difficult.

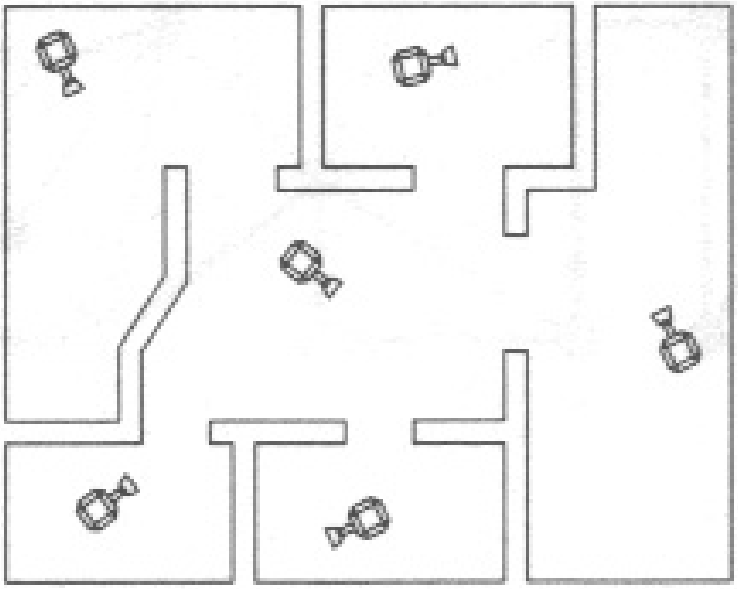
- The performance and usability of visualization systems depends on a fast answer to this **point query** for large data sets.
- Point location queries must run in real time, e.g.:
  - navigation systems
  - mouse clicks in windows or graphical object on the screen
  - …

➡ Point location: Chapter 5

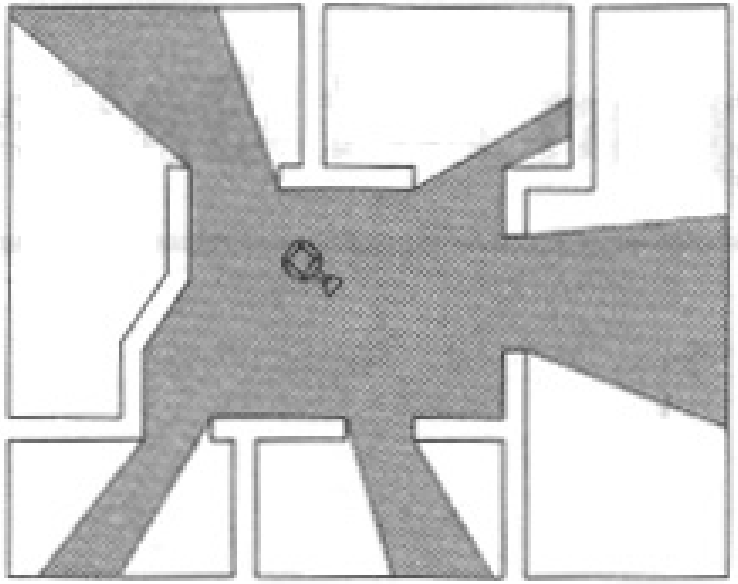
1.1 Typical problems

d) For security reasons an art gallery shall be equipped with pivoting surveillance cameras.

Question d): How many cameras do you need at least and where do you place them?



Art gallery with cameras



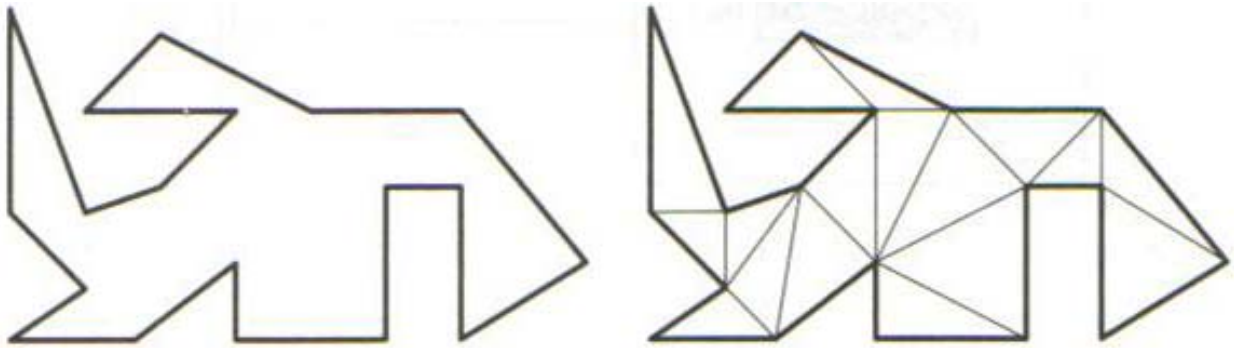
Area controlled by one camera

1.1 Typical problems

This leads to triangulations of arbitrary (simple) polygons.

- A polygonal area represents the art gallery
- Camera positions correspond to points inside the polygon.
- Extreme case: Convex polygons can be controlled by one camera from an arbitrary point inside the polygon.

➡ This implies that you need at least as many cameras as there are triangles on the triangulation.



A simple polygon and a possible triangulation of it.

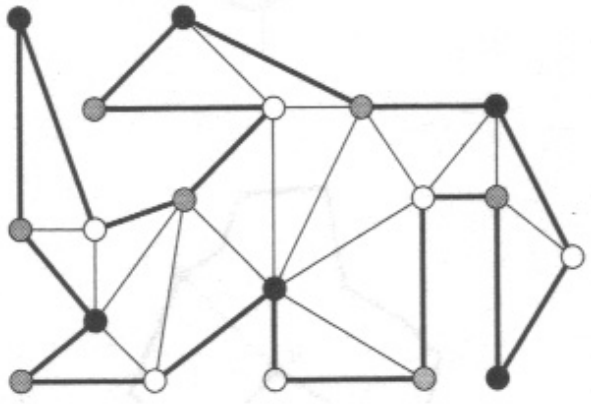
1.1 Typical problems

Instead of placing in each triangle one camera, one can look for better solutions with less cameras:

- place cameras on particular diagonals,

  or

- place cameras in particular corners polygon.



- The connection to computer graphics is obvious because since GPUs process triangles.
  - Every CAD system must be able to triangulate arbitrary polygons automatically.

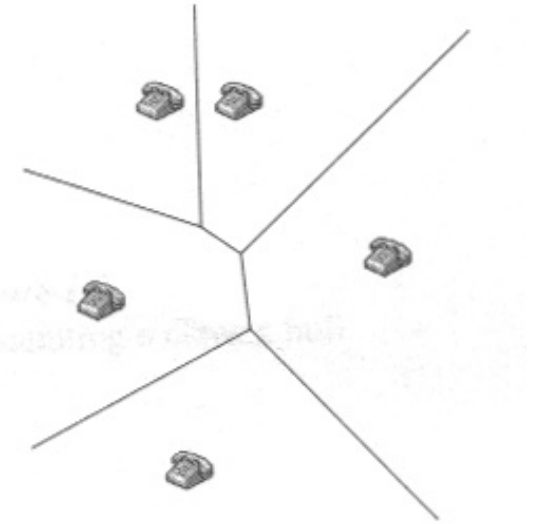
➡ Polygon triangulation: Chapter 6

1.1 Typical problems

e) You need to make an urgent telephone call on campus but you do not have a cell phone.

However, you know some telephone boxes near by.

- **Question e):** Which is the closest telephone box to your position?
- A solution would be a map subdivided into regions, where each region contains the closest telephone box.
- **Further questions:**
  - What is the shape of these regions?
  - How can these regions be computed?

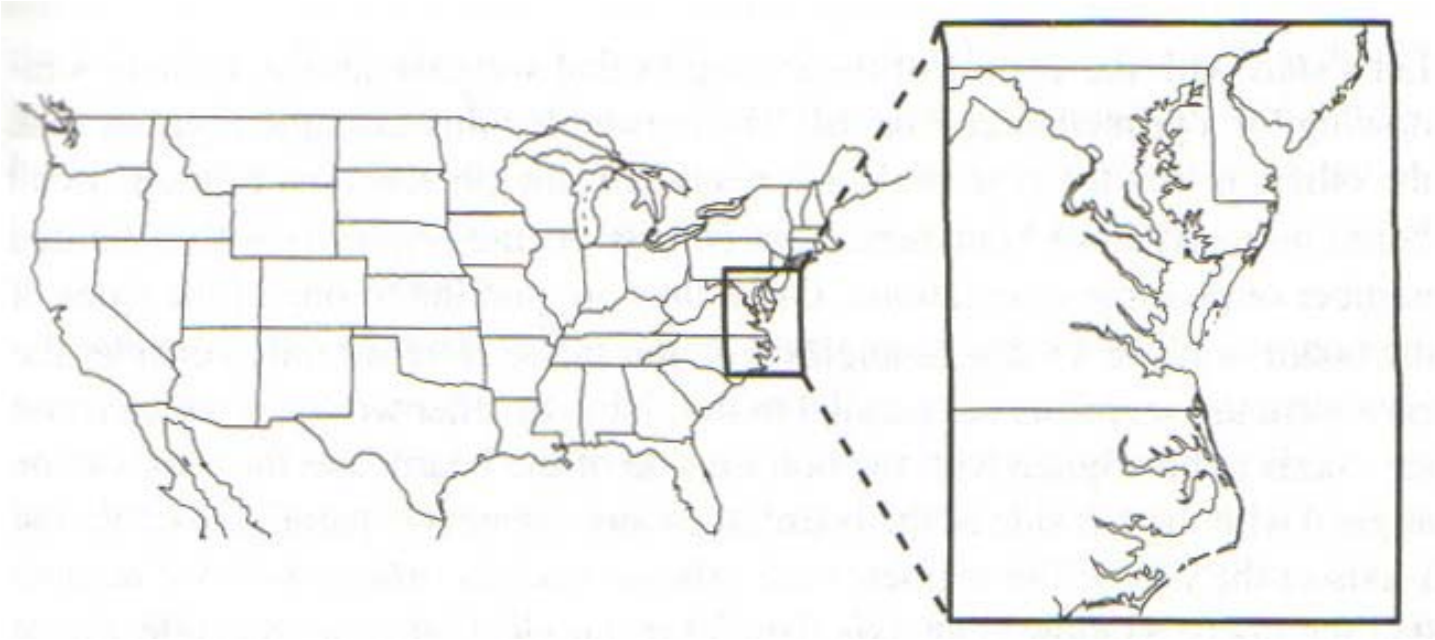


➡ Voronoi diagrams & Delaunay triangulations: Chapter 7

1.1 Typical problems

f) In a navigation system all streets, villages, rivers, etc. of the USA are stored.

However, on the display only a small section around the current position is visible.



1.1 Typical problems

Question f): Compute all entities that are visible within a given viewing window.

- To solve this a window query is computed on the data base of points, line segments and polygons.
- **Remark:** This window query must run a cheap hardware with very little resources.
- Further applications:
  - Flight simulators, displaying only a small section (viewing volume) of a 3d landscape.
  - Control of chip layouts.

➡ Window query: Chapter 9

1.2 Basics

1.2.1 Geometric objects

- ***Points*** in d-dimensional Euclidian space $\mathbb{E}^d$
- Fixed Cartesian coordinate system with origin O.
  - Points are represented by coordinate tuples:

$$p = (p_1, \dots, p_d), q = (q_1, \dots, q_d), \dots \in \mathbb{R}^d.$$

- A coordinate tuple p represents also a vector from the origin O to the point p.

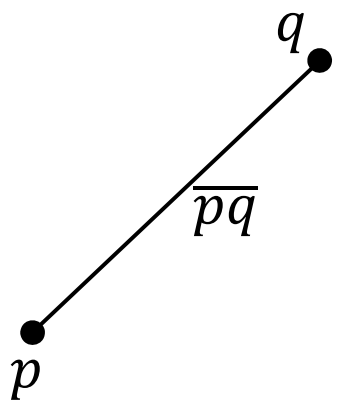
1.2.1 Geometric objects

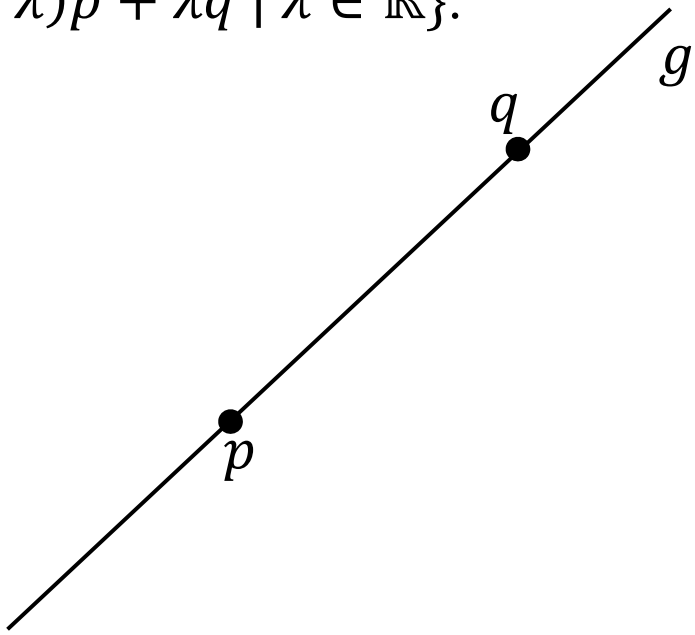
- For coordinate tupels $p, q \in \mathbb{R}^d, \lambda \in \mathbb{R}$, we have the operations:
  - Addition: $p + q := (p_1 + q_1, \dots, p_d + q_d).$
  - Multiplication: $\lambda \cdot p := (\lambda p_1, \dots, \lambda p_d).$
  - Eucidean Norm: $|p| := \sqrt{p_1^2 + \cdots + p_d^2}.$
  - Scalar product: $\langle p, q \rangle := (p_1 q_1 + \cdots + p_d q_d) = |p| \cdot |q| \cos(\sphericalangle pq).$
  - Cross product: $|p \times q| = |p| \cdot |q| \sin(\sphericalangle pq).$
- ***Orthogonality***: Two vectors are orthogonal, if

$$\langle p, q \rangle = 0.$$

1.2.1 Geometric objects

- ***Line segments*** (Strecke)
  - Two points p and q with $\overline{pq} := \{(1-\lambda)p + \lambda q \mid \lambda \in [0,1]\}$.
- ***Lines*** (Gerade)
  - Two points p and q with $g := \{(1-\lambda)p + \lambda q \mid \lambda \in \mathbb{R}\}$.

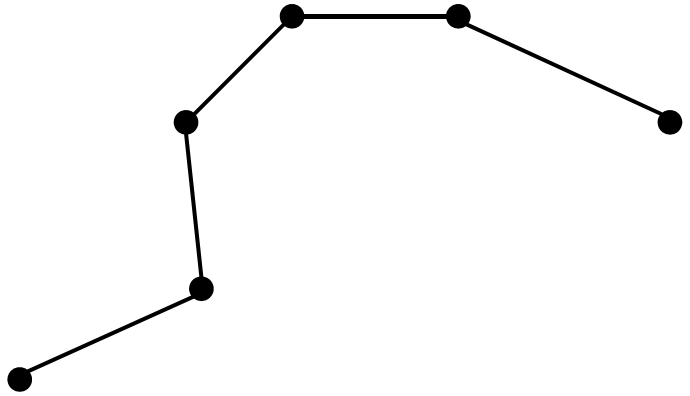




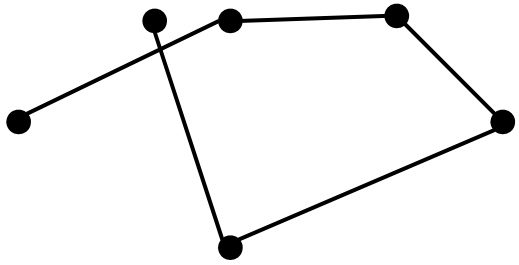
1.2.1 Geometric objects

Polygon chains

- A ***polygon chain*** of length n is a sequence of line segments with $\overline{p_0p_1}, \overline{p_1p_2}, \dots, \overline{p_{n-1}p_n}$.



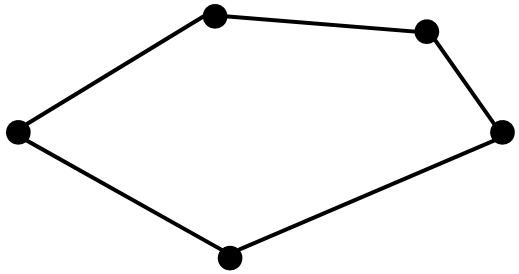
- A polygon chain is called ***simple***, if it has no self intersections.



Self intersecting polygon chain in $\mathbb{R}^2$

1.2.1 Geometric objects

- ***Polygons*** (closed polygon chain)
  - A polygon chain with $p_0 = p_n$ is called ***polygon.***



1.2.2 Primitive operations

- ***Primitive*** **geometric operations** involve only objects of constant size, e.g.:
  - Intersection of two lines.
  - Point lies in half-space of a hyper-plane.
  - *Not primitive:* Point lies in polygon, intersection of two polygons.

- Primitive operations are treated in the ***computation model*** (see §1.2.3) as operations with constant run time and memory consumption.

1.2.3 Computation model

- **Real RAM**
  - Countable infinitely many memory cells, that are addressable by natural numbers.
  - Memory cells can take an arbitrary number of real numbers and integers.
  - The following operations are primitive and take only constant time:
    - arithmetic operations (+, - , × , / , *mod* , *div*),
    - comparisons of real numbers,
    - indirect addressing,
    - roots, trigonometric functions, logarithms.

1.2.4 Runtime of algorithms

- ***Runtime of an algorithm*** = Number of used primitive operations!
  - One primitive operation corresponds to one time step.
- Runtime depends on size n of input!
- Thus, the runtime is a ***function of the input size!***
  - E.g. $T(n) = a\,n^2 + b\,n + c$ for constants a, b and c.
  - $T(n)$ is also called *runtime function in* n.

1.2.4 Runtime of algorithms

- Reminder:
  - *Best-Case*-Analysis
  - *Worst-Case*-Analysis
  - *Average-Case*-Analysis
- The complete runtime functions is often very complicated.
- For simplicity only the ***order of runtime*** is considered!!
  - The order of $T(n) = a\,n^2 + b\,n + c$ is e.g. n^2.
  - The order of $T(n) = a\,n + b$ is n.

1.2.5 O-calculus

The O-Calculus (Landau-Symbols)

- Runtime functions can be very complicated.
- To determine the complexity of a problem runtime functions must be compared.

➡ The asymptotic growth of a function is most relevant.

➡ **Idea:** Compare the growth of two functions $g(n)$ and $f(n)$ only for large n.

- „Which function wins the race?“

- Advantages:
  - Constant factors are irrelevant.
  - For polynomials only the larges exponent counts.

Edmund Landau, 1877-1938.
Quelle: Wikipedia

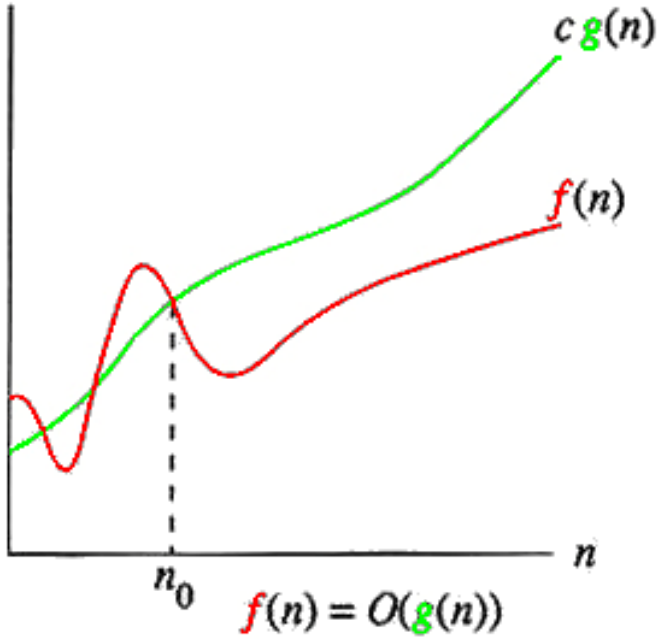
1.2.5 O-calculus

Definition: (pronounced: „Big-Oh“)

$O(g(n)) = \{f(n)$: There are positive constants c and n_0, such that

$$0 \leq |f(n)| \leq c\,|g(n)| \text{ for all } n \geq n_0\}.$$

- Notation: $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.
- Examples:
  - $3n = O(n^2)$
  - $n^2/2 - 3n = O(n^2)$
  - $6n^3 \neq O(n^2)$
- $f(n) = O(g(n))$ corresponds to „$a \leq b$“.



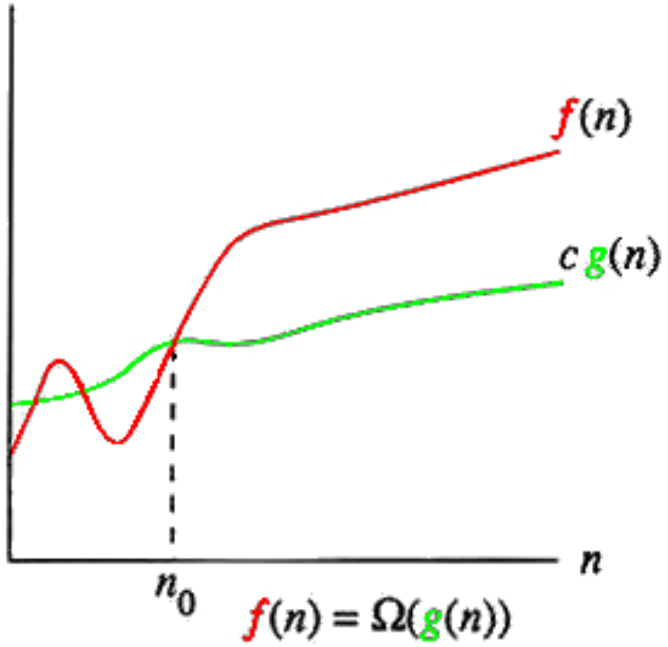
1.2.5 O-calculus

Definition: (pronounced: „Big-Omega“)

$\Omega(g(n)) = \{f(n)$: There are positive constants c and n_0, such that

$0 \leq c\,|g(n)| \leq |f(n)|$ for all $n \geq n_0\}$.

- Examples:
  - $3n \neq \Omega(n^2)$
  - $n^2/2 - 3n = \Omega(n^2)$
  - $6n^3 = \Omega(n^2)$
- $f(n) = \Omega(g(n))$ corresponds to „$a \geq b$“.



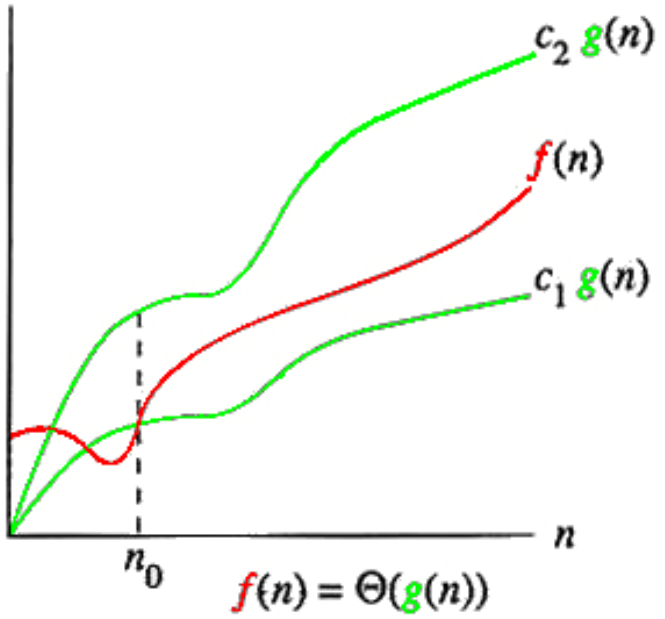
1.2.5 O-calculus

Definition: (pronounced: „Big-Theta“)

$\Theta(g(n)) = \{f(n)$: There are positive constants c_1, c_2 and n_0, such that

$0 \leq c_1|g(n)| \leq |f(n)| \leq c_2\,|g(n)|$ for all $n \geq n_0\}$.

- Examples:
  - $3n \neq \Theta(n^2)$
  - $n^2/2 - 3n = \Theta(n^2)$
  - $6n^3 \neq \Theta(n^2)$
- $f(n) = \Theta(g(n))$ corresponds to „$a = b$“.



1.2.5 O-calculus

Definition: (pronounced: „Small-Oh“)

$o(g(n)) = \{f(n)$: For all constants $c > 0$ there is a $n_0 > 0$, such that

$$0 \leq |f(n)| < c\,|g(n)| \text{ for all } n \geq n_0\}.$$

- Examples:
  - $3n = o(n^2)$
  - $n^2/2 - 3n \neq o(n^2)$
  - $6n^3 \neq o(n^2)$
- $f(n) = o(g(n))$ corresponds to „$a < b$“.
- An equivalent condition for $f(n) = o(g(n))$ is

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

1.2.5 O-calculus

Definition: (pronounced: „Small-Omega“)

$\omega(g(n)) = \{f(n)$: For all constants $c > 0$ there is a $n_0 > 0$, such that $0 \leq c\,|g(n)| < |f(n)|$ for all $n \geq n_0\}$.

- Examples:
  - $3n \neq \omega(n^2)$
  - $\frac{n^2}{2} - 3n \neq \omega(n^2)$
  - $6n^3 = \omega(n^2)$
- $f(n) = \omega(g(n))$ corresponds to „$a > b$“.
- An equivalent condition for $f(n) = \omega(g(n))$ is

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

1.2.6 Master-Theorem

- For a divide-and-conquer-algorithm („teile und herrsche“) with runtime function $T(n)$ the problem of input size n
  1. is divided into a sub-problems of size n/b with runtime $f(n)$,
  2. every sub-problem is solved with runtime $T(n/b)$ and
  3. all sub-problems are generated and all sub-solutions are merged („conquer“) with runtime total $f(n)$.
- What is the total runtime of such an algorithm?
- In many cases an answer is given by the

„Master-Theorem“.

1.2.6 Master-Theorem

Theorem (Master Theorem for Recursions)

Let $T(n)$ be a runtime function of the form $T(n) = a \cdot T(n/b) + f(n)$ with

- constants $a \geq 1$ and $b > 1$,
- a function $f(n)$,
- and n/b is interpret as $\lceil n/b \rceil$ or $\lfloor n/b \rfloor$.

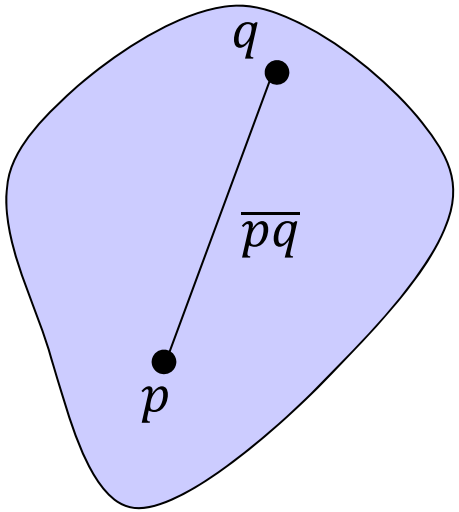
Then $T(n)$ satisfies one of the following estimates:

1. If $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
3. If $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ for $\varepsilon > 0$
   and $a \cdot f(n/b) \leq c \cdot f(n)$ for $c < 1$ and sufficiently large n,
   then $T(n) = \Theta(f(n))$.

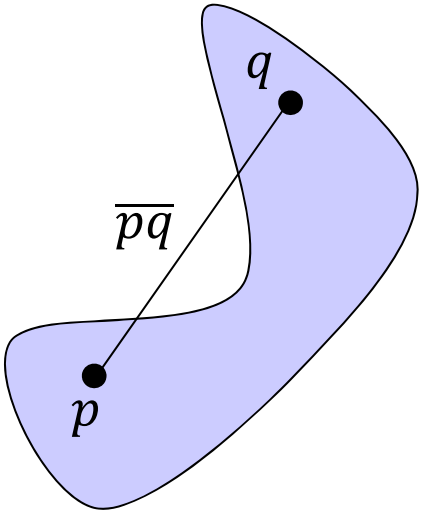
1.2.7 Example: 2d-convex hull

Definition (Convex Set)

A set $S \subseteq \mathbb{R}^d$ is called **convex**, iff for all points $p, q \in S$ the connecting line segment $\overline{pq} = \{x : x = \lambda p + (1-\lambda)q, \lambda \in [0,1]\}$ is contained in S, i.e. $\overline{pq} \subseteq S$.



convex



non-convex

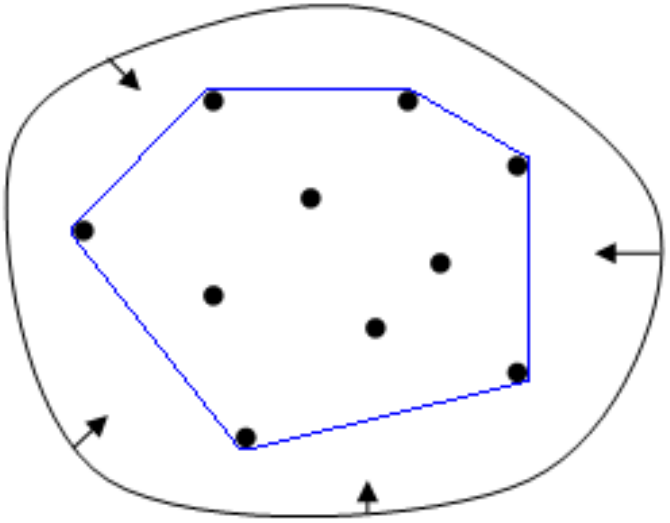
1.2.7 Example: 2d-convex hull

Definition (Convex Hull)

The **convex hull** $\mathcal{CH}(S)$ of a set $S \subseteq \mathbb{R}^d$ is the smallest convex set containing S,

➡ i.e. it is the intersection of all convex sets containing S.

Rubber band metaphor



1.2.7 Example: 2d-convex hull

- For $d = 2$ there is an alternative (informal) definition:

  The convex hull $\mathcal{CH}(P)$ of a finite set P of n points is the unique, convex polygon with corners from P containing all points of P.

➡ This definition is used to develop an algorithm computing the convex hull of a set of points.

1.2.7 Example: 2d-convex hull

Remark: Convex hull algorithms are to Computational Geometry what sorting is to discrete algorithms.

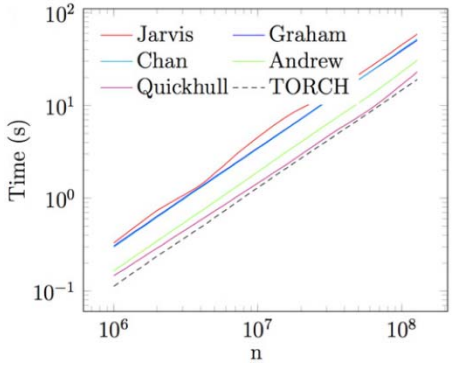
Remark: The complexity of the convex hull problem in 2d is $\Theta(n \log n)$.

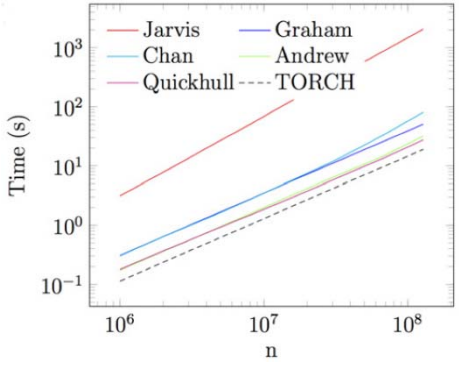
1.2.7 Example: 2d-convex hull

Remark:

There are many approaches to compute the convex hull of a 2d point set P:

1. **Incremental:** Naïve, Kallay: $O(n^3)$, $O(n \log n)$.
2. **Recursive:** Divide-and-conquer, QuickHull: $O(n \log n)$.
3. **Sweeping:** Graham, Andrew, TORCH: $O(n \log n)$.
4. **Output-size sensitive:** Jarvis, Chan: $O(n\,k)$, $O(n \log k)$
   if $\mathcal{CH}(P)$ is a k-gon.





Quelle: Gomes