Computational Geometry

- Introduction
- Algorithms Classification
- Computational complexity
- Degeneracies and Robustness

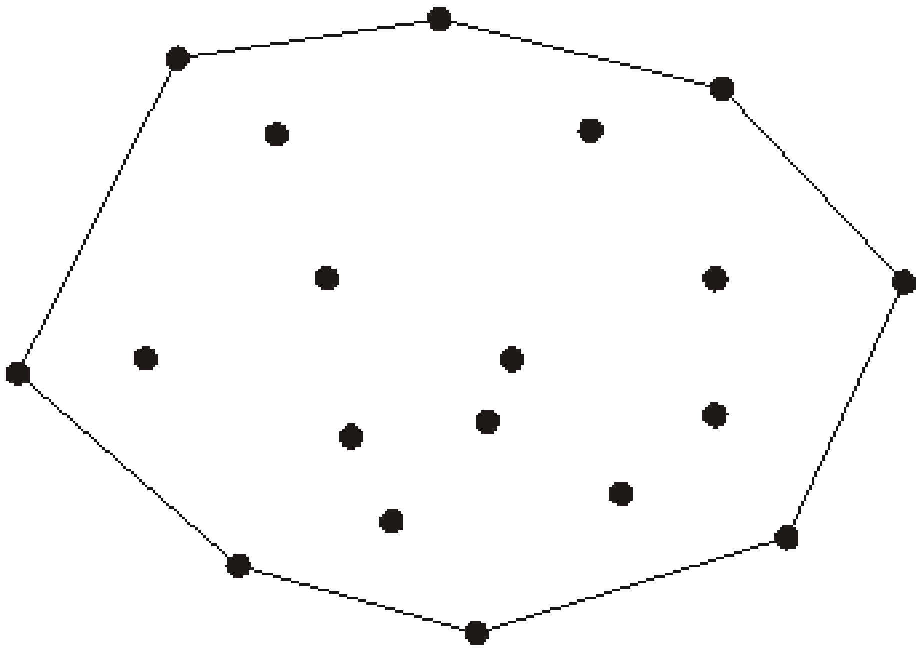
 11.10.2021

What is computational geometry ?

Computational geometry is a branch of the theory of computations that studies geometric problems of great size focusing on robust and asymptotically fast algorithms.

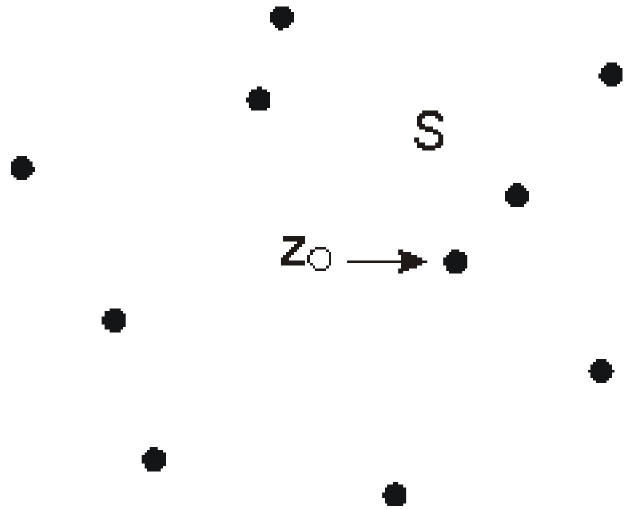
Example

Given a set S of $\boldsymbol{n}$ points on the plane. Find the convex hull of S.



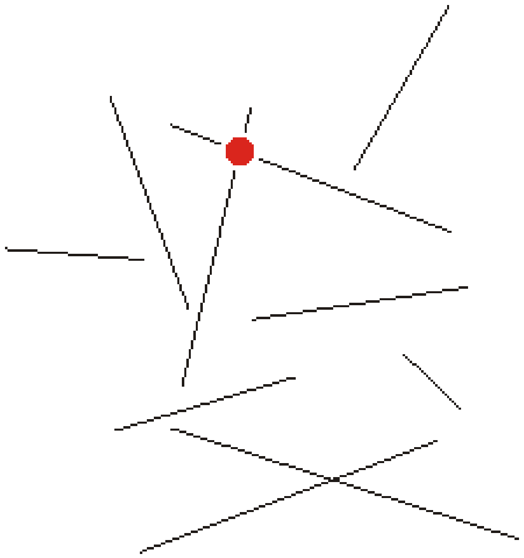
Example

Given a set S of $\boldsymbol{n}$ points on the plane and a point $z \notin S$. Find a point in S closest to z.



Example

Given a set of $\boldsymbol{n}$ line segments on the plane. Do any two segments intersect?



Computational complexity

- Computational complexity of an **algorithm**: amount of time spending by the algorithm as a function of size of the problem. Usual notation is $\boldsymbol{T(n)}$, where n denotes the size of the problem.

- Computational complexity of a **problem**: complexity of the best algorithm that solves this problem.

Computational complexity

DETECT-SEGMENT-INTERSECTION (S)

for $i \to 0$ **to** $n-1$ **do**

  for $j \to i + 1$ **to** n **do**

    if (INTERSECTS($S[i]$, $S[j]$))

      return true;

return false;

$$T(n) = c_1 \frac{n(n-1)}{2} + c_2 n + c_3$$

Asymptotic complexity

Asymptotic complexity is behavior of computational complexity $T(n)$ if n approaches the infinity.

Example:

If for some constant $C > 0$ and $N \geq 0$ such that for all $n \geq N$

$$T(n) = c_1 \frac{n(n-1)}{2} + c_2 n + c_3 \leq C\boldsymbol{n^2},$$

then an *asymptotic notation* is used

$$T(n) = O(\boldsymbol{n^2}).$$

The constants C and N can be estimated as

$$C = a + |b| + c, N = \left\lceil \frac{2|b|}{a} \right\rceil,$$

where $a = c_1/2, b = c_2 - a, c = c_3$.

O-notation

- A notation
$$T(n) = O\big(f(n)\big)$$
means that there exist constants $C > 0$ and $N > 0$ such that for all $n \geq N$
$$T(n) \leq Cf(n),$$

- By definition, it means that
$$\lim_{n\to\infty} \frac{T(n)}{f(n)} = 1.$$

- Is used to denote the **worst-case** of algorithm running time.

- We say that, in the worst-case, the algorithm will spend **at most** $O(f(n))$ time to solve the problem.

Ω-notation

- A notation
$$T(n) = \Omega\big(f(n)\big)$$
means that there exist constants $C > 0$ and $N > 0$ such that for all $n \geq N$
$$T(n) \geq Cf(n),$$

- Is used to denote the **lower-bound** of the complexity of a problem.

- We say that the best algorithm will need **at least** $\Omega(f(n))$ time to solve this problem.

θ-notation

- A notation

$$T(n) = \theta\big(f(n)\big)$$

  means that

$$T(n) = O\big(f(n)\big) = \Omega\big(f(n)\big).$$

- We say that $f(n)$ is an **asymptotically tight** bound of $T(n)$.

- Is used to denote computational complexity of **optimal** algorithms for the specified problem.

Growth of functions

$T(n)$	$T_1 = T(1000)$	$T_2 = T(1000000)$	T_2/T_1
$\log n$	10	20	2
n	10^3	10^6	10^3
$n \log n$	10^4	2×10^7	2×10^3
n^2	10^6	10^{12}	10^6
2^n	10^{300}	10^{300000}	10^{299700}

Growth of functions

$T(n)$	$T_1 = T(1000)$	$T_2 = T(1000000)$	T_2/T_1
$\log n$	10	20	2
n	10^3	10^6	10^3
$n \log n$	10^4	2×10^7	2×10^3
n^2	10^6	10^{12}	10^6
2^n	10^{300}	10^{300000}	10^{299700}

Linear growth of complexity regarding growth of the input

Growth of functions

$T(n)$	$T_1 = T(1000)$	$T_2 = T(1000000)$	T_2/T_1
$\log n$	10	20	2
n	10^3	10^6	10^3
$n \log n$	10^4	2×10^7	2×10^3
n^2	10^6	10^{12}	10^6
2^n	10^{300}	10^{300000}	10^{299700}

1000 times growth of complexity regarding growth of the input!

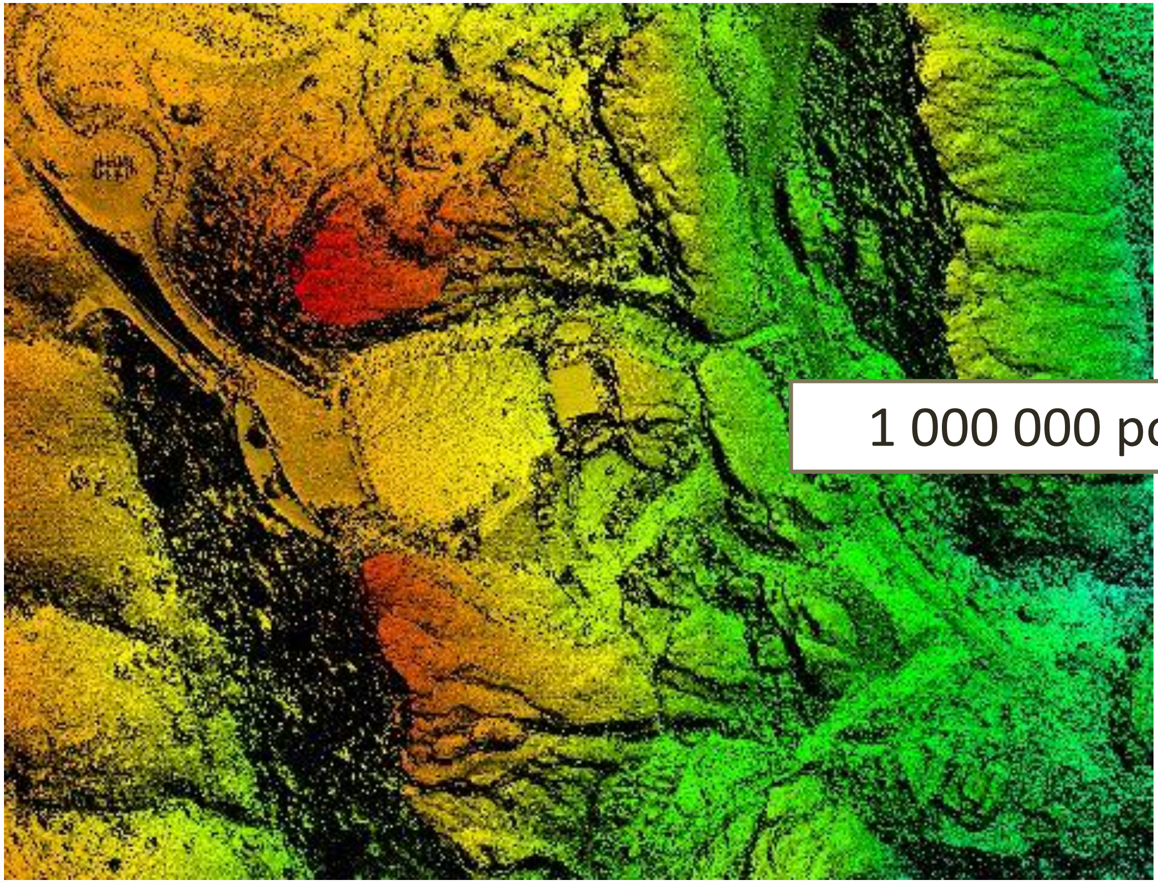
Growth of functions

$T(n)$	$T_1 = T(1000)$	$T_2 = T(1000000)$	T_2/T_1
$\log n$	10	20	2
n	10^3	10^6	10^3
$n \log n$	10^4	2×10^7	2×10^3
n^2	10^6	10^{12}	10^6
2^n	10^{300}	10^{300000}	10^{299700}

1000 times growth of complexity regarding growth of the input!

Natural limitation for solving problems of great size

Digital Terrain Model



1 000 000 polygons

Stanford's Digital Michelangelo

2 000 000 000 polygons

When and what kind of optimization is appropriate?

- $O(\log n)$: **real time** request (milliseconds)
  - Search and display of the current information under mouse pointer, `onMouseMove()` event handler.
  - Analysis and display of permanently changing object status.
- $O(n)$: mouse click request (0.5 – 1 sec).
- $O(n \log n)$: short time computation (1 sec – few minutes).
- $O(n^k)$: long computation (few hours – few days).
- $O(2^n)$:  (years, centuries)

Algorithm development: three steps

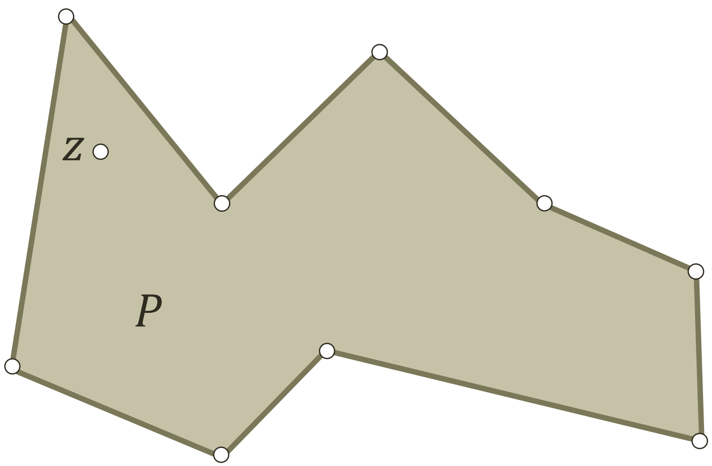
Step 1. Design an algorithm with the **best asymptotical complexity**.

Step 2. Handle **degeneracies**.

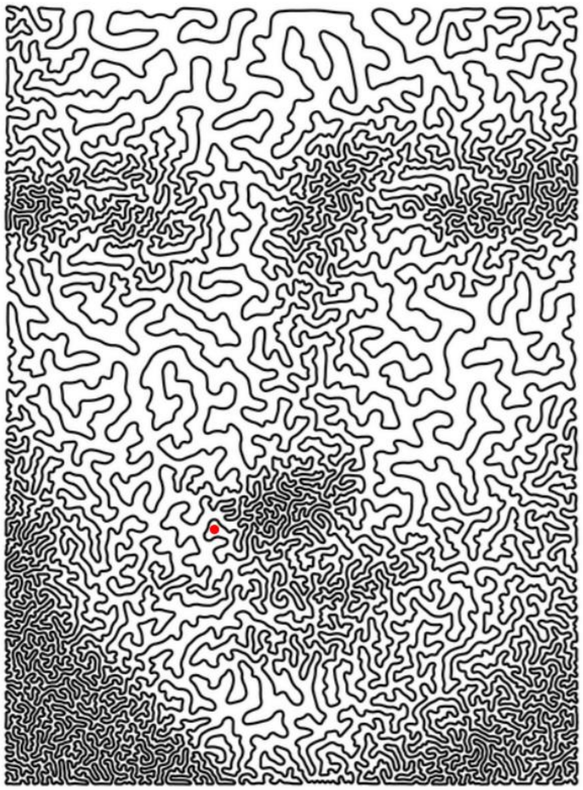
Step 3. Provide **computational robustness**.

Algorithm development: example

Given a simple polygon P and a point z, identify whether z belongs to P.

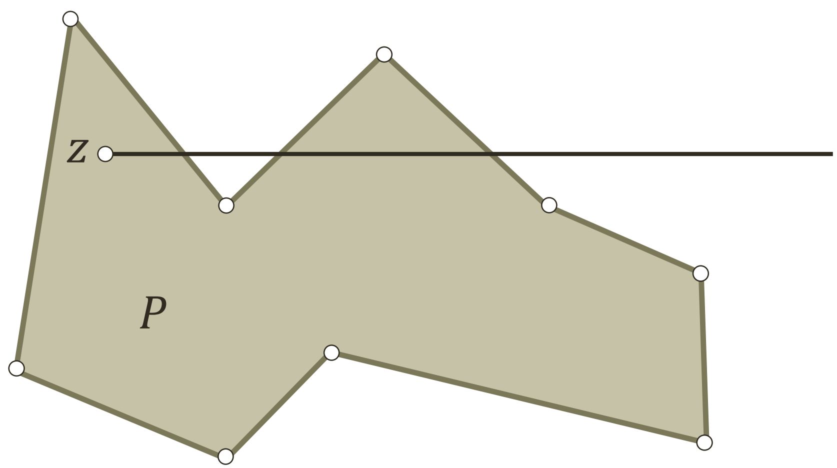


A simple polygon, as well…



Algorithm development: example

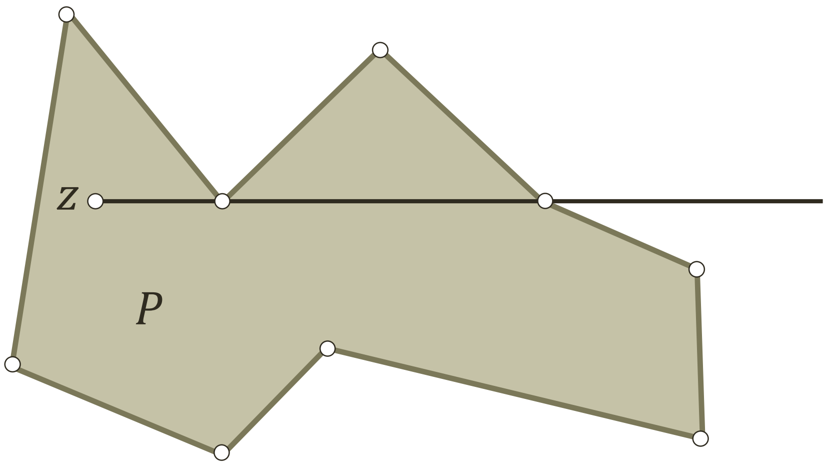
Step 1: design an algorithm with the best asymptotical complexity.



 11.10.2021

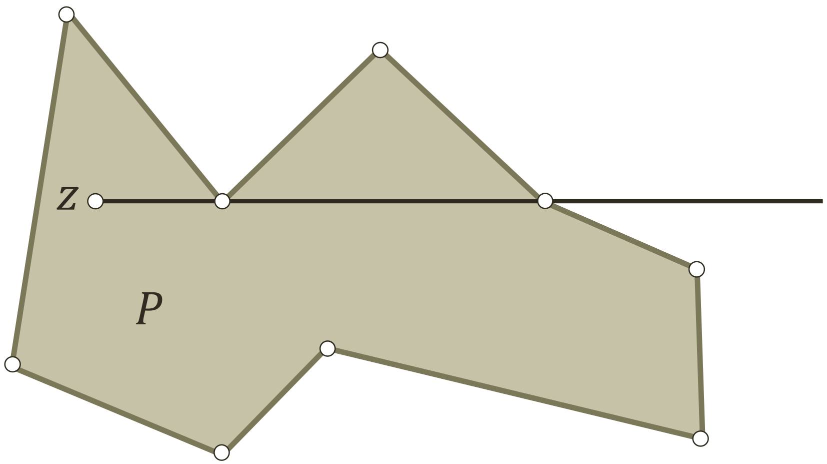
Algorithm development: example

Step 1: design an algorithm with the best asymptotical complexity.



Algorithm development: example

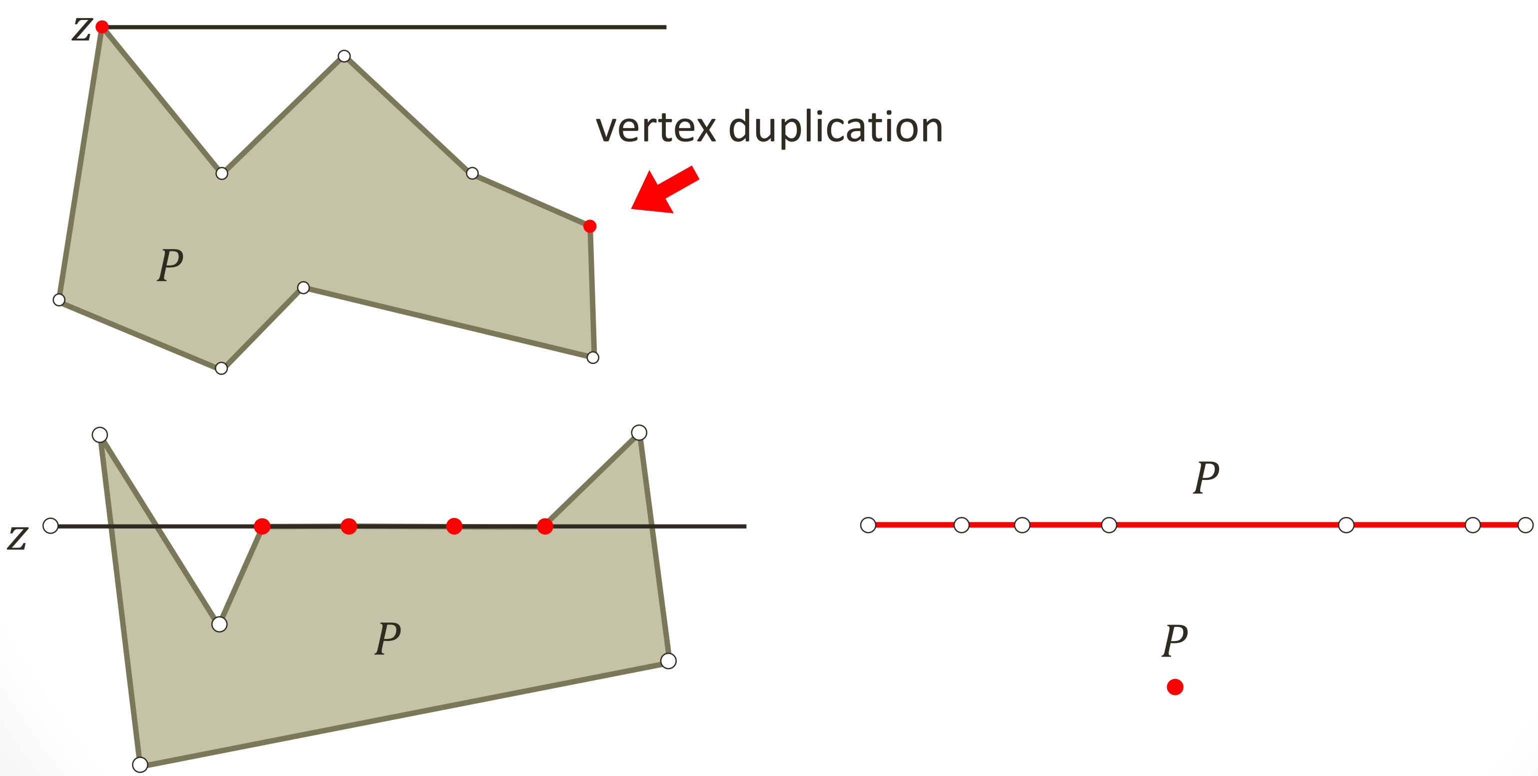
Step 1: design an algorithm with the best asymptotical complexity.



Asymptotical complexity: $\boldsymbol{T(n) = \theta(n)}$

Algorithm development: example

Step 2: handle degeneracies.



 11.10.2021

Algorithm development: example

Step 3: provide computational robustness.

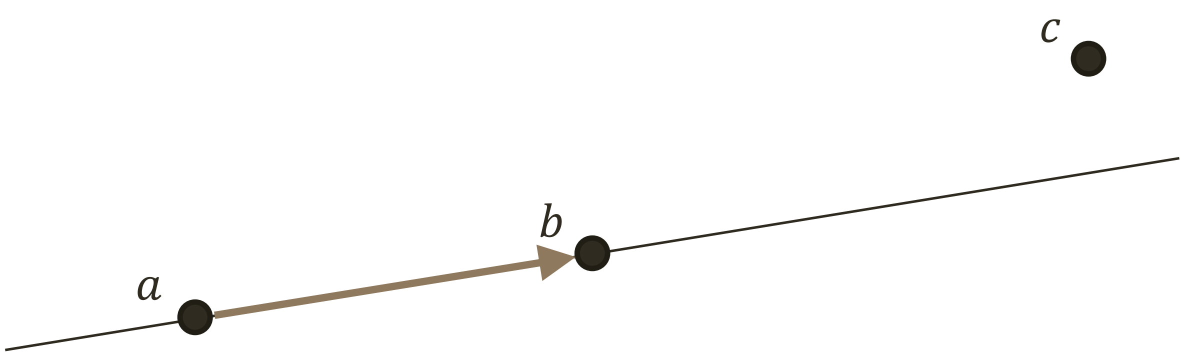
Rounding errors may **crucially impact on algorithm execution**.

Geometric predicates

Identify location of a point c on the plane relative to an oriented line, that passes through points a and b.

In other words, are the points a, b and c arranged in clockwise (CW) order, counterclockwise (CCW) order or they are collinear?

- The points are defined by their coordinates of type **`double`**.
- The solution must be correct for ***any*** input.



Geometric predicates

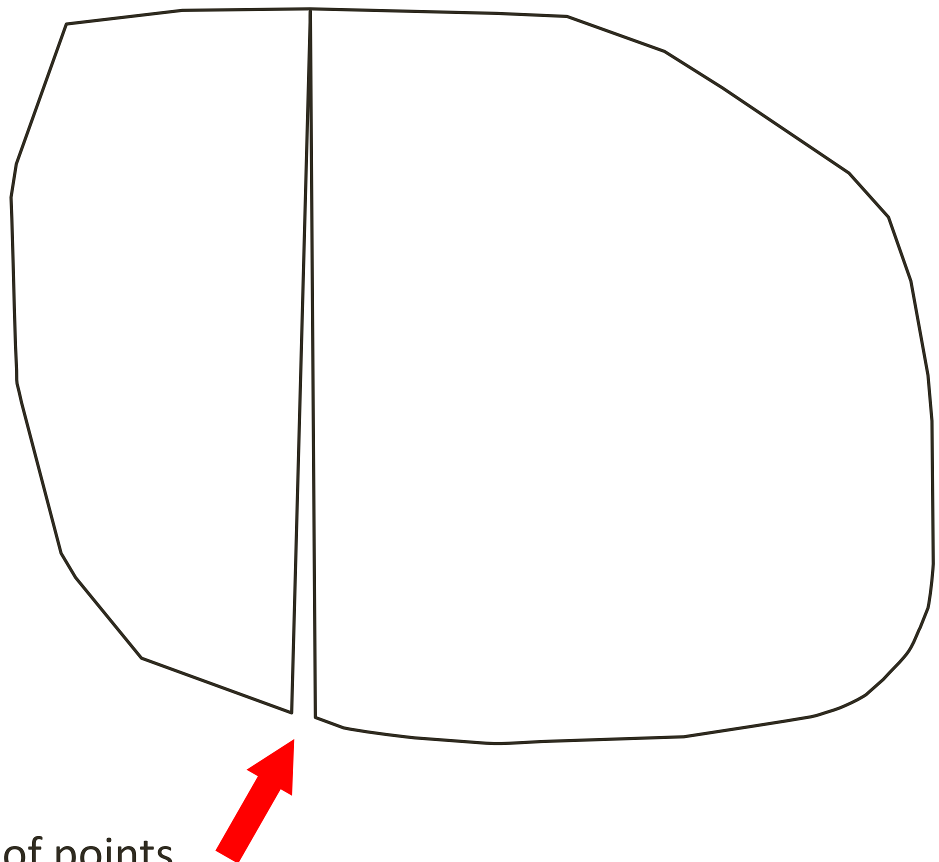
$$D = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix}$$

- $D > 0 \Rightarrow c$ is on the **left** of the line (a, b)
- $D < 0 \Rightarrow c$ is on the **right** of the line (a, b)
- $D = 0 \Rightarrow c$ is **on** the line (a, b)

```
int orient2d(double ax, double ay, double bx, double by, double cx, double cy)
{
    return (ax - cx) * (by - cy) - (bx - cx) * (ay - cy);
}
```

Geometric predicates

Convex hull

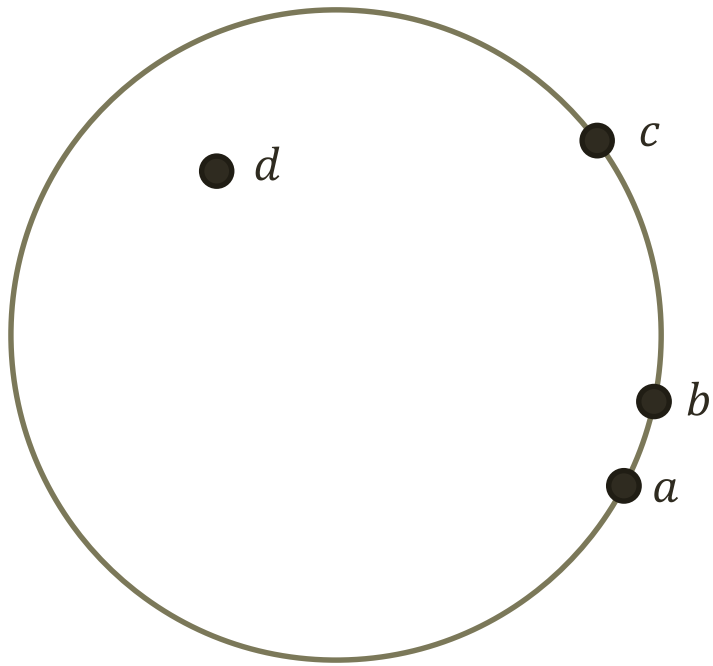


Incorrect identification of points arrangement caused by rounding error

Geometric predicates

Identify location of a point d on the plane relative to a circle that passes through points a, b and c.

In other words, is d **inside**, **outside** or **exactly on** the circle?



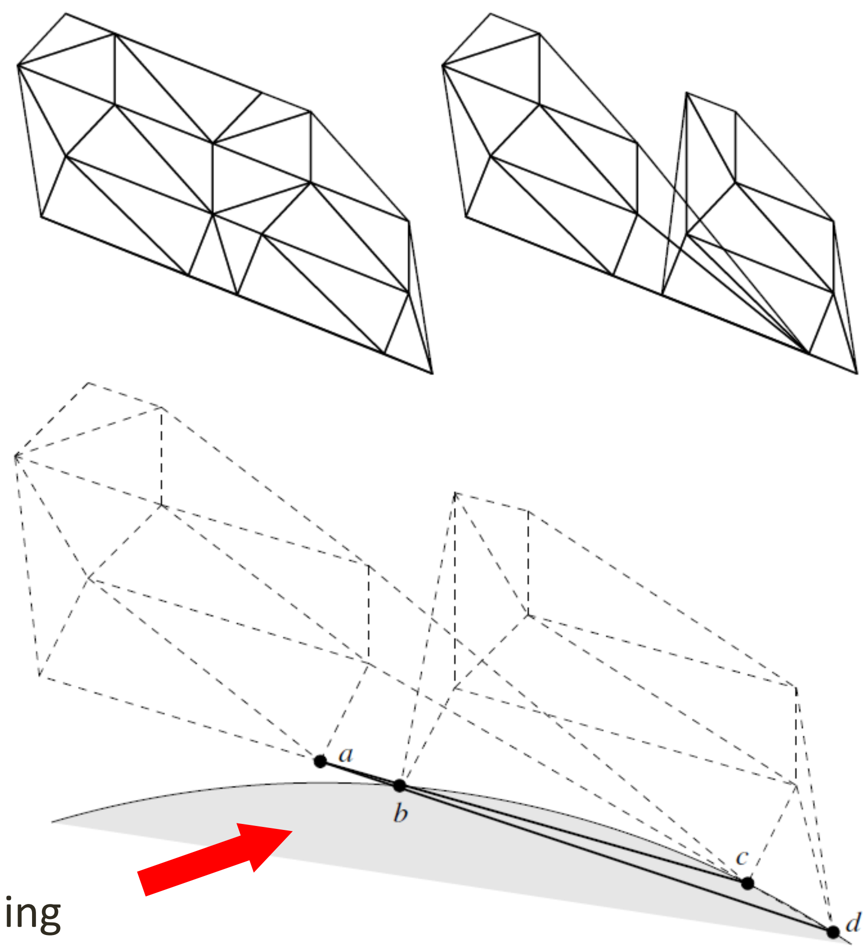
Geometric predicates

$$D = \begin{vmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & a_x^2 + a_y^2 & 1 \\ c_x & c_y & a_x^2 + a_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{vmatrix}$$

- $D > 0 \Rightarrow d$ is inside the circle,
- $D < 0 \Rightarrow d$ is outside the circle,
- $D = 0 \Rightarrow d$ is on the circle

Geometric predicates

Triangulation
[Shewchuk 1999]



Incorrect identification of point location relative to a circle (the Delaunay test) caused by rounding errors

Computational robustness

- Big numbers
- Exact arithmetic
- Adaptive arithmetic

A particular case: integer arithmetic

- point positions are specified by `__int32` (4 bytes),
- intermediate and final results are stored in `__int64` (8 bytes) and `__int128` (16 bytes).

Computational robustness

- Big numbers
- Exact arithmetic
- Adaptive arithmetic

Arbitrary precision floating-point arithmetic:

- a number x is expressed as an *expansion*
$$x = x_n + \cdots + x_2 + x_1,$$
- $|x_n| > \cdots > |x_1|$,
- components x_i are *nonoverlapping* by digit positions

for example, $12.3456 = 12. + 0.34 + 0.0056$

The sign of x is equal to the sign of the largest component x_n!

Exact arithmetic

Addition rule [Dekker]:

Fast-Two-Sum(a, b)

```
x ← a ⊕ b
b_virt ← x ⊖ a
y ← b ⊖ b_virt
return (x, y)
```

$(a + b) = (7231. + 56.78)$

a	7231.
b	56.78
$x = a \oplus b$	7287.
a	7231.
$b_{virt} = x \ominus a$	56.
$y = b \ominus b_{virt}$	0.78

$(x + y) = (7287. + 0.78)$

Exact arithmetic

Kahan Summation Formula:

Summation $(a_1, \dots, a_n)$

$s \leftarrow a_1$

$y \leftarrow 0$

for $i \leftarrow 1$ **to** n

  $b \leftarrow a_i \ominus y$

  $x \leftarrow s \oplus b$

  $b_{virt} = x - s$

  $y = b_{virt} \ominus b$

  $S = x$

return S

Computed sum is equal to

$$\sum x_i(1 + \delta_i) + O(n\epsilon^2)\sum |x_i|,$$

where

$$|\delta_i| \le 2\epsilon$$

Naïve summation gives

$$\sum x_i(1 + \delta_i),$$

where

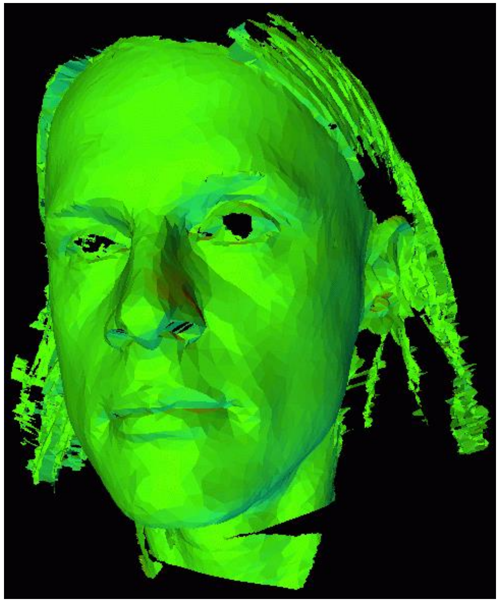
$$|\delta_i| < (n - i)\epsilon.$$

Computational robustness

- Big numbers
- Exact arithmetic
- Adaptive arithmetic

The idea: use exact arithmetic **only when necessary**!

Adaptive arithmetic



Jonathan Shewchuk

Professor in Computer Science University of California at Berkeley

```
int orient2d(p, p1, p2);
int orient3d(p, p1, p2, p3);
int incircle(p, p1, p2, p3);
int insphere(p, p1, p2, p3, p4);
```

More than 4000 lines of source code on C

Example

Does a point belong to a given triangle?

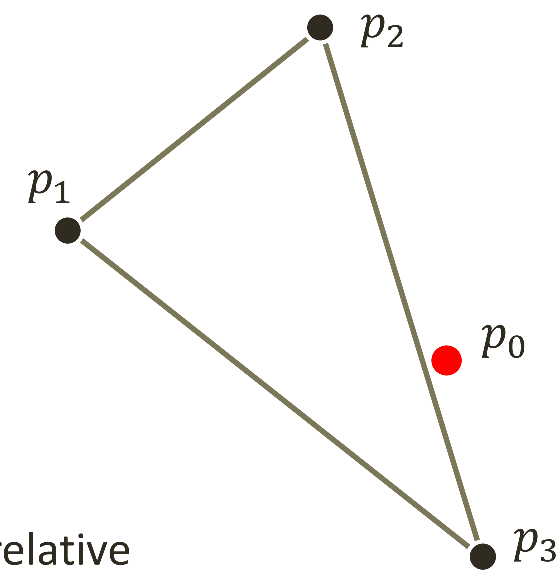
1. Primitive (non-robust) solution: estimate the barycentric coordinates of p_0

$$
\begin{aligned}
x_0 &= x_1 b_1 + x_2 b_2 + x_3 b_3 \\
y_0 &= y_1 b_1 + y_2 b_2 + y_3 b_3 \\
1 &= \quad b_1 + \quad b_2 + \quad b_3
\end{aligned}
$$

```
bool inside = (b1 >= 0 && b1 <= 1) &&
              (b2 >= 0 && b2 <= 1) &&
              (b3 >= 0 && b3 <= 1);
```

2. Robust solution: estimate exact position of p_0 relative to the sides of the triangle

```
bool inside = (orient2d(p0, p1, p2) <= 0 &&
               orient2d(p0, p2, p3) <= 0 &&
               orient2d(p0, p3, p1) <= 0);
```



References

1. **Franco P. Preparata, Michael Ian Shamos.** Computational Geometry: An Introduction. Springer-Verlag, 1985
2. **Mark de Berg, et al.** Computational Geometry. *Algorithms and Applications*. Springer, 2008.
3. **Д. М. Васильков.** Геометрическое моделирование и компьютерная графика: вычислительные и алгоритмические основы. Мн., БГУ, 2011.
4. **David Goldberg**. What every computer scientist should know about floating-point arithmetic. *Journal ACM Computing Surveys (CSUR)*, vol. 23 Issue 1, March 1991 (5-48).
5. **J. R. Shewchuk**. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18:305–363, 1997.
6. https://www.cs.cmu.edu/~quake/robust.html