Computational Geometry

Tutorial #2 — Working with convex hulls

Institut für Betriebssysteme
und Rechnerverbund
Algorithmik

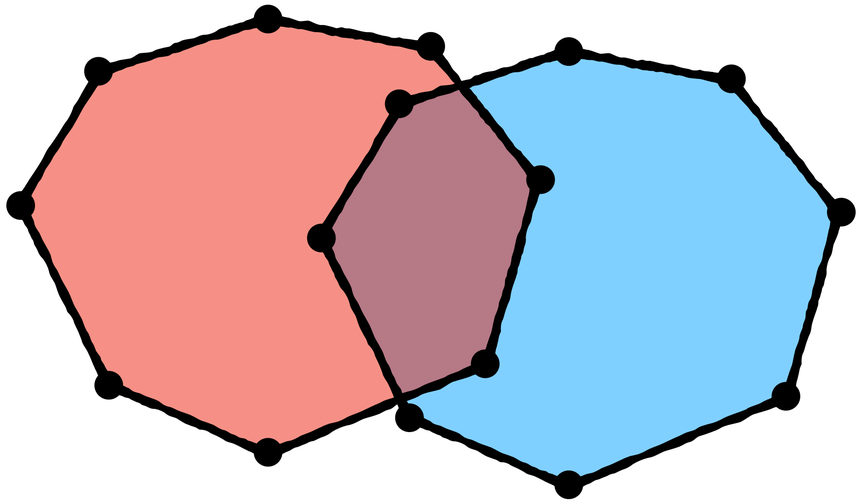
Peter Kramer | *November 5th, 2026*

Boolean Operations

Boolean Operations on Convex Polygons

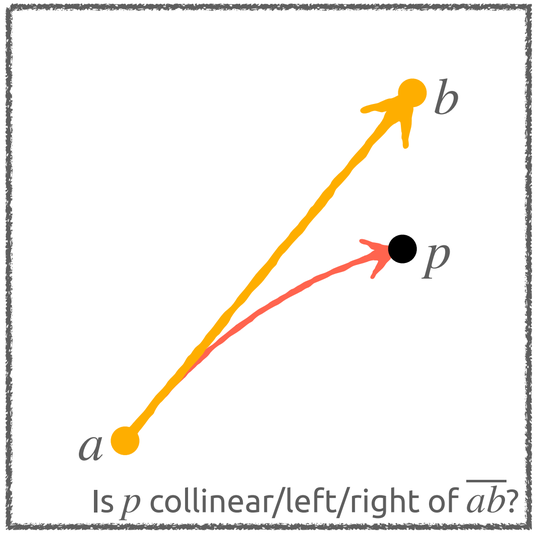
Given two convex Polygons P and Q, we seek to determine:

$$P \cap Q, P \cup Q, P \backslash Q, (Q \backslash P)$$



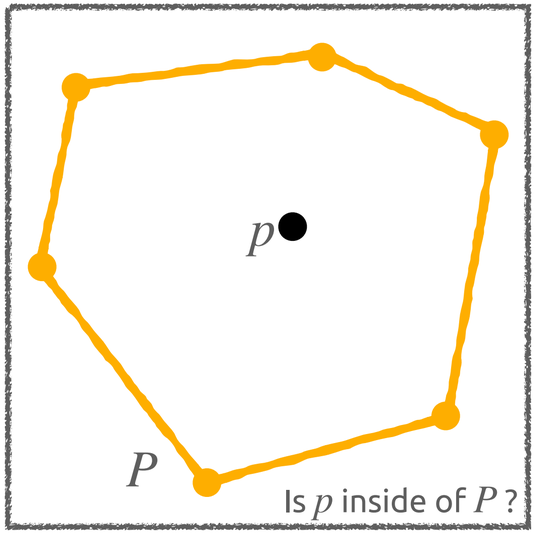
Tools

Point-Line Test



$\mathcal{O}(1)$

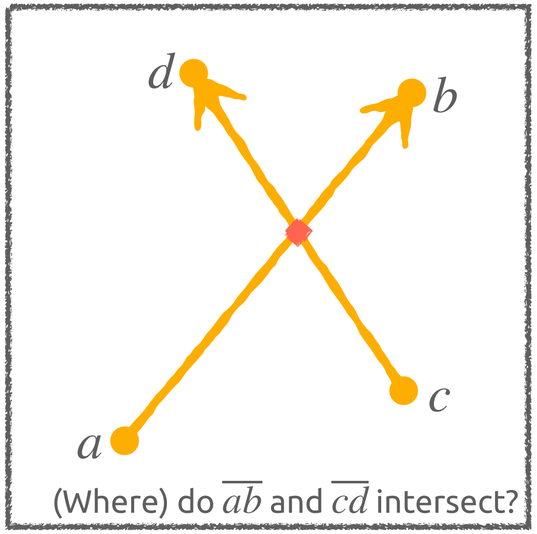
Point Location Problem



$\mathcal{O}(\log n)$

(If P convex)

Intersection Test



$\mathcal{O}(1)$

Naive Algorithm using these tools

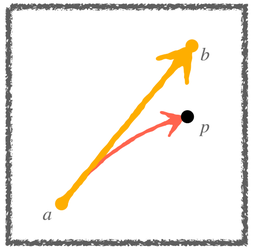
Eliminate/Add points, then recompute convex hull

Given: Convex polygons $P := p_1, \ldots, p_n$ and $Q := q_1, \ldots, q_m$ in counterclockwise order.

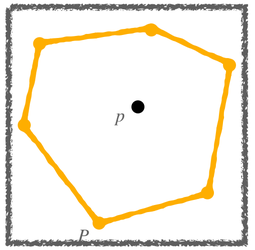
Wanted: The convex polygon $P \cap Q$.

Idea: Determine extreme points of $P \cap Q$ in $\mathcal{O}(n^2)$, then compute the convex hull in $\mathcal{O}(n \log h)$.

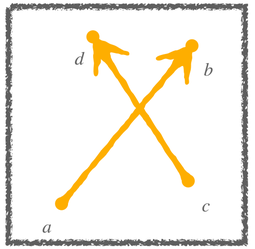
This gives us an $\mathcal{O}(n^2)$-algorithm.



$\mathcal{O}(1)$



$\mathcal{O}(\log n)$

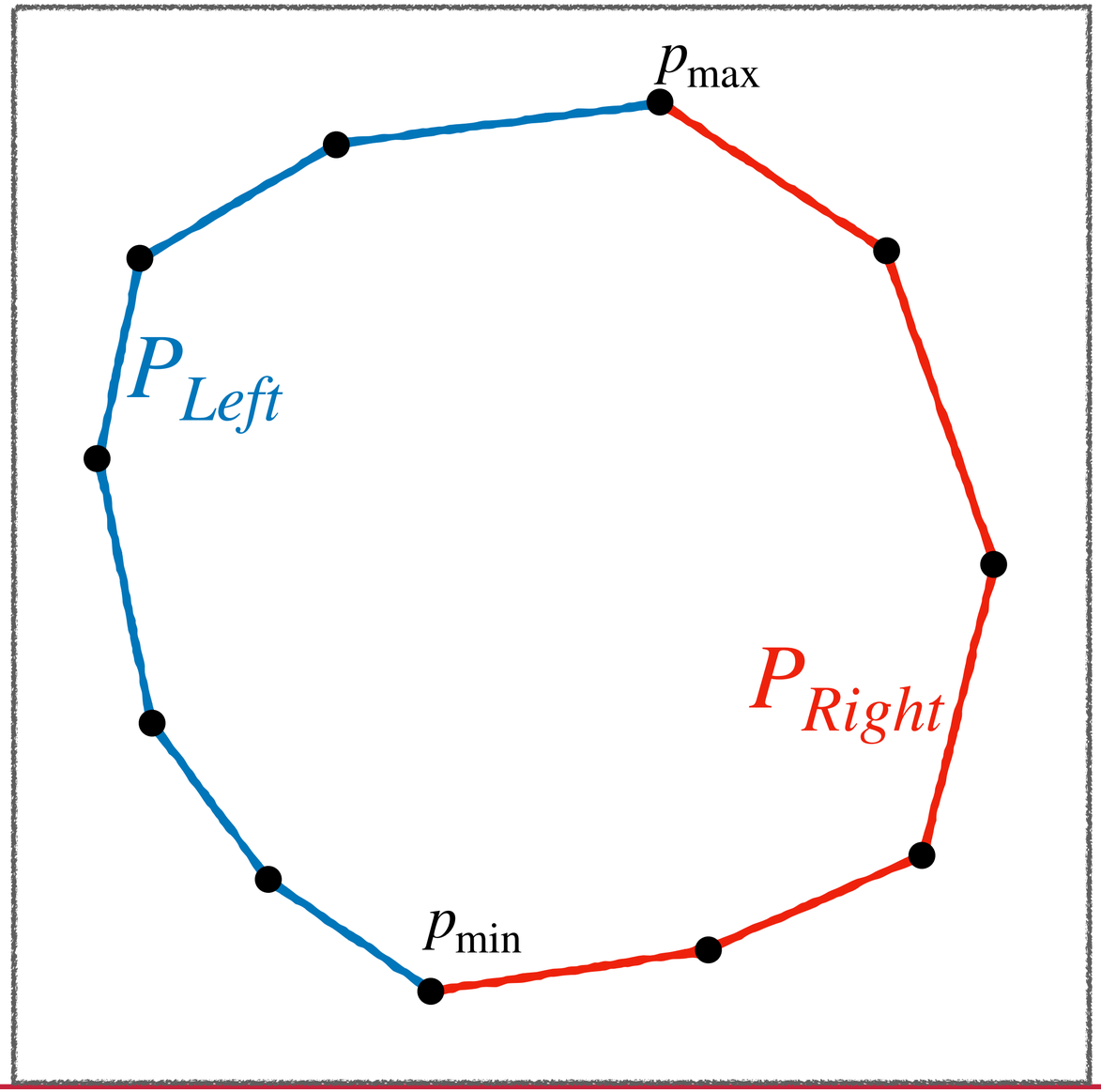


$\mathcal{O}(1)$

Convex polygons
… can be decomposed!

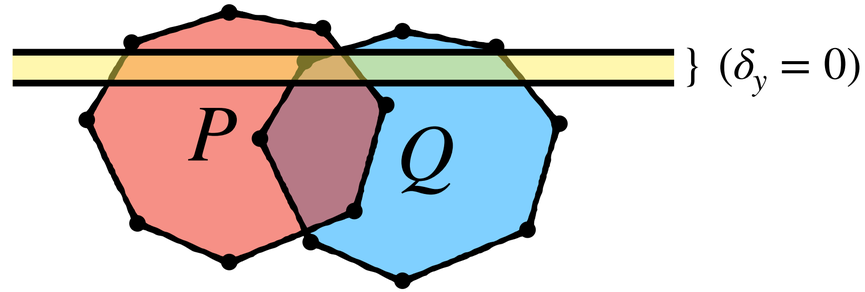
- Given $P = p_1, \ldots, p_n$
- Compute $p_{\min}$ and $p_{\max}$ in $\mathcal{O}(n)$ along the y-axis
- We obtain P_L and P_R:

$$P_L = p_{\max}, \ldots, p_{\min}$$
$$P_R = p_{\min}, \ldots, p_{\max}$$

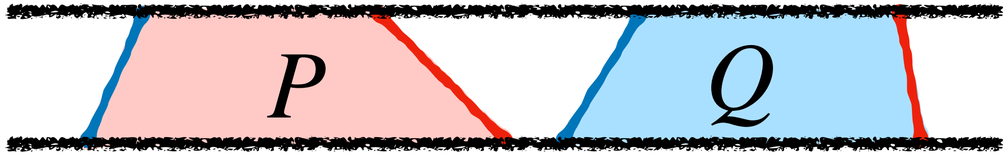


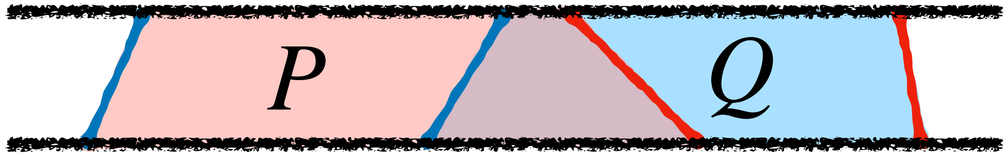
Towards an $\mathcal{O}(n)$-Algorithm

Using left- and right decomposition



- If we slice through P and Q horizontally, at some y (see right, exaggerated):
  - (a) P and Q do not intersect,
  - (b) P and Q overlap partially, or
  - (c) one contains the other.

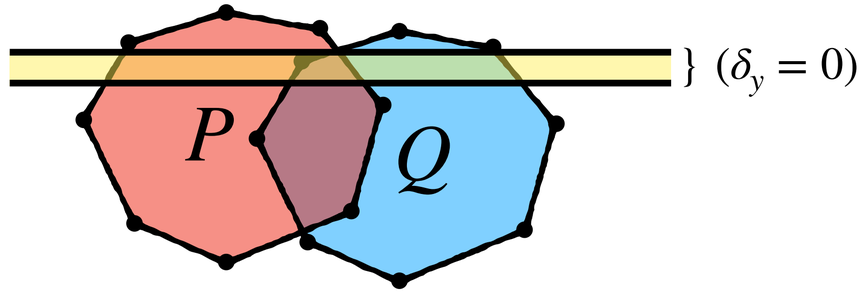




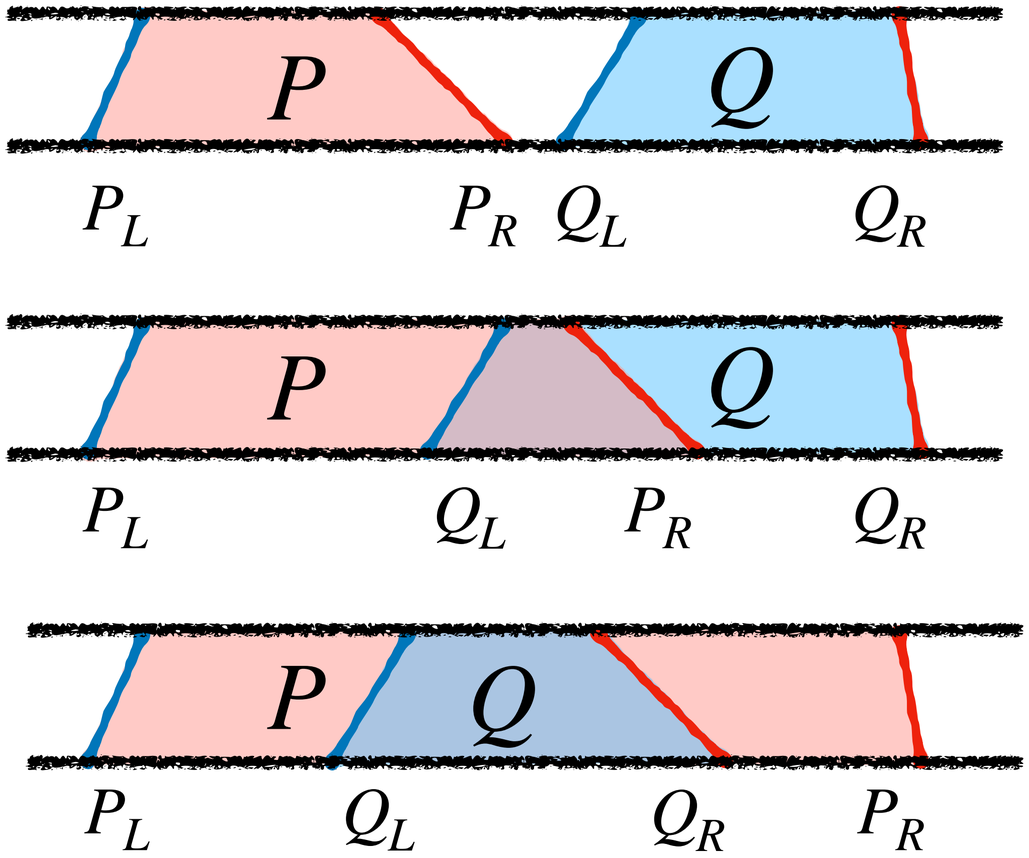


Towards an $\mathcal{O}(n)$-Algorithm

Using left- and right decomposition



- If we slice through P and Q horizontally, at some y (see right, exaggerated):
  - (a) P and Q do not intersect,
  - (b) P and Q overlap partially, or
  - (c) one contains the other.
- Each case corresponds to an x-order of the chains at that y-coordinate.



```
Intersection(P, Q):
    Split P into P_L, P_R and Q into Q_L, Q_R
    L, R ← empty lists
    Sweep horizontally from bottom to top
    over the common y-range of P and Q.

    While not at bottom:
        y ← topmost among the four paths
        l ← max(P_L(y), Q_L(y))
        r ← min(P_R(y), Q_R(y))
        if l ≤ r:
            append (l, y) to L
            append (r, y) to R
    return L + reverse(R)
```

