

# Geometrie 2 - ICPC Praktikum SS14

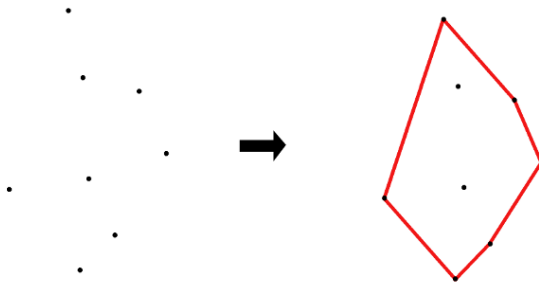
| 25. Juni 2014

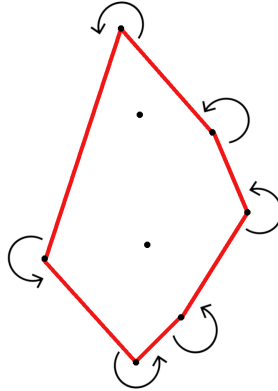
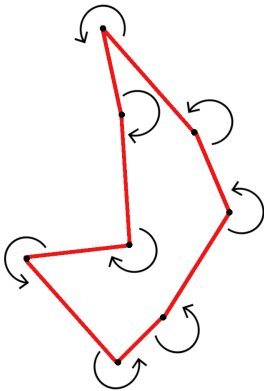
TOBIAS HORNBERGER · PAUL JUNGEBLUT · LENA WINTER



## Problem

Gegeben sei eine Menge  $M$  von Punkten in der Ebene. Die konvexe Hülle von  $M$  ist die kleinste konvexe Menge, in der  $M$  enthalten ist.





## Fazit

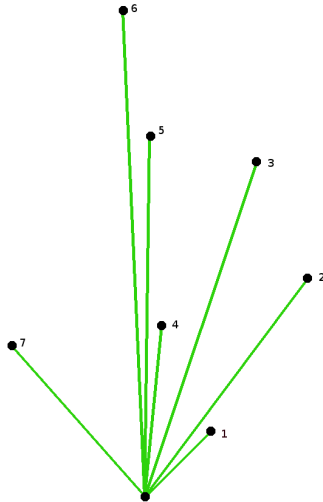
Konvexe Hüllen haben nur links Abbiegungen.

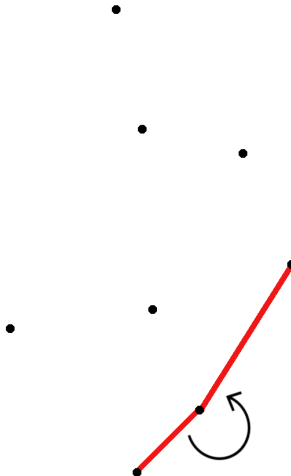
- Bestimme den untersten Punkt  $P_0$
- Sortiere die Punkte nach Winkel relativ zu  $P_0$
- Füge den Punkt mit dem größten Winkel der konvexen Hülle hinzu
- Nimm nun jeweils den Punkt mit dem kleinsten Winkel und überprüfe mit CCW:
  - liegt er links des Vektors  $P_{k-1} P_k$  füge ihn der konvexen Hülle hinzu
  - liegt er rechts so entferne solange Punkte aus der Konvexen Hülle bis er links des letzten Vektors liegt
- Wurden alle Punkte betrachtet so hat man die konvexe Hülle gefunden.

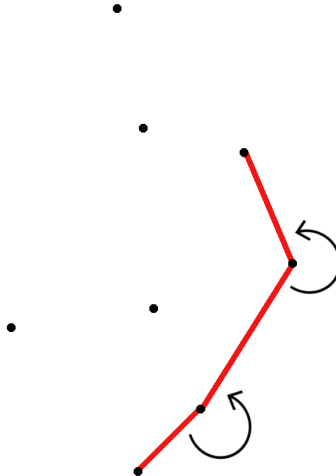
## Sonderfälle

- Liegen drei Punkte auf einer Linie wird das als Linksknick interpretiert
- Haben zwei Punkte den gleichen Winkel so werden Sie lexikographisch sortiert.

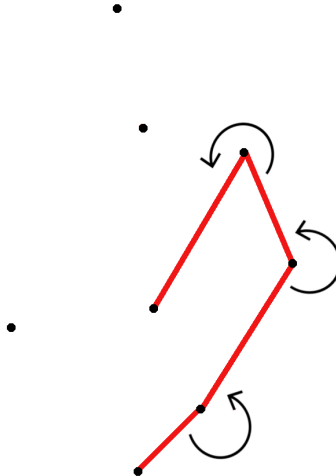
# Beispiel

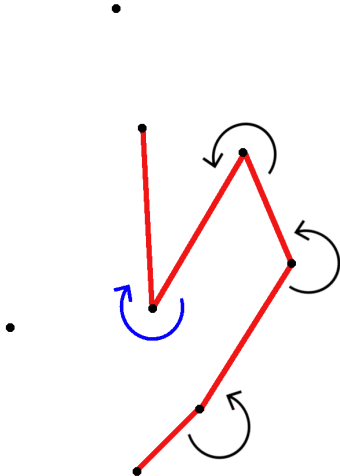


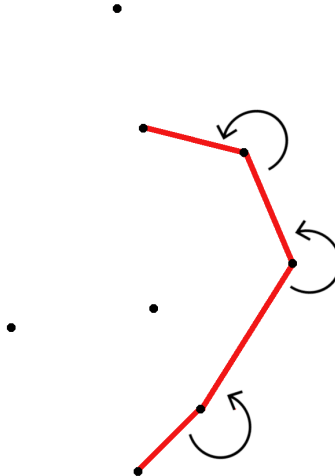


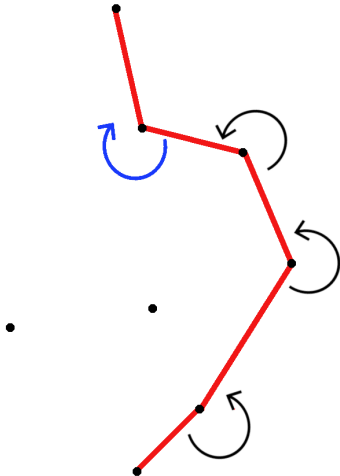


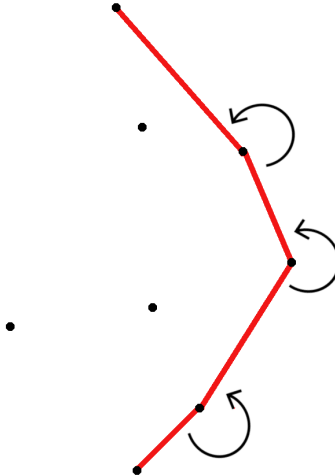




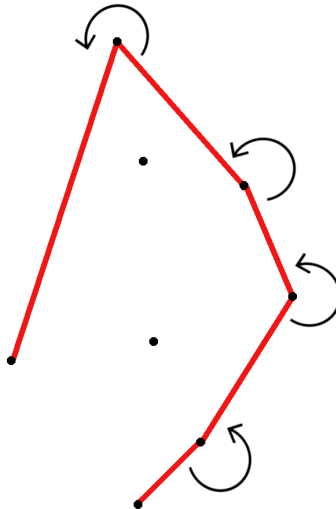




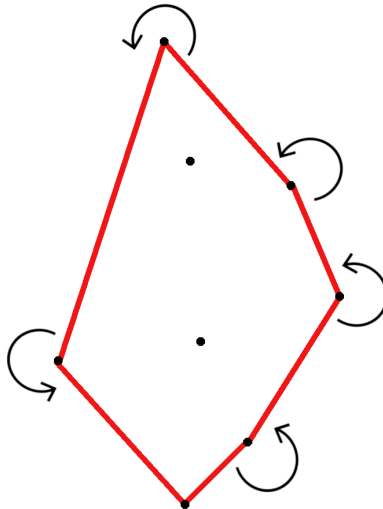




# Beispiel



# Beispiel



**Geben:**  $n$  Punkte auf einer Ebene

**Gesucht:** die beiden am nächsten zusammenliegenden Punkte

## Naiver Ansatz

Mit Vollständiger Suche.

Alle Distanzen zwischen allen möglichen Punktpaaren ausrechnen und davon das Minimum wählen.

**Laufzeit:**  $\mathcal{O}(n^2)$



# Idee: Divide and Conquer

Statt Vollständiger Suche: Divide & Conquer für eine Lösung in  $\mathcal{O}(n \log n)$  Zeit.

- 1 Divide:**  
Sortieren der Punkte (Primär x-Koordinate, sekundär y-Koordinate).  
Aufteilen der Punktmenge in zwei Hälften

- 2 Conquer:**  
Größe der Punktmengen:
  - $|Punktmenge| = 1$ , return  $\infty$
  - $|PunktMenge| = 2$ , return Euklidische Distanz der beiden Punkte

- 3 Combine:**  
Sei  $d_1$  die kleinste Distanz innerhalb der Punktmenge  $A_1$ .  
Sei  $d_2$  die kleinste Distanz innerhalb der Punktmenge  $A_2$ .  
Sei  $d_3$  die kleinste Distanz von 2 Punkte aus jeweils  $A_1$  und  $A_2$ .  
→ Die kleinste Distanz innerhalb  $A_1 \cup A_2$  ist  $\min(s_1, s_2, s_3)$

Naiver Ansatz für Combine immer noch in Laufzeit  $\mathcal{O}(n^2)$

## Optimierbar!

Sei  $d' = \min(d_1, d_2)$ .

Für jeden Punkt in der unteren Punktmenge kann der nähere Punkt nur in einem Rechteck mit Breite  $d'$  und Höhe  $2 * d'$  liegen

### Beweisbar

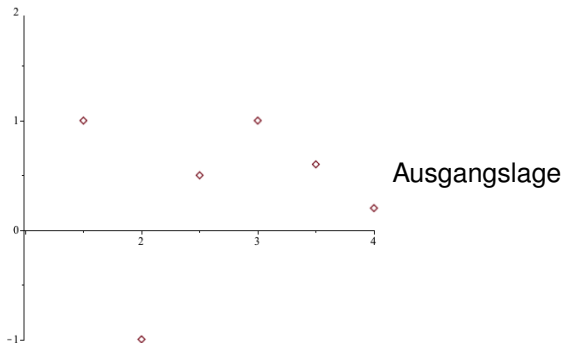
Es gibt maximal 6 solche Punkte im Rechteck.

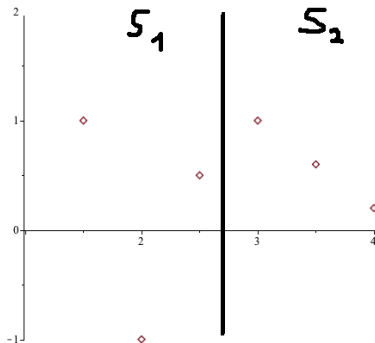
Ohne Beweis

⇒ Maximal  $\mathcal{O}(6n)$  Operationen für Combine

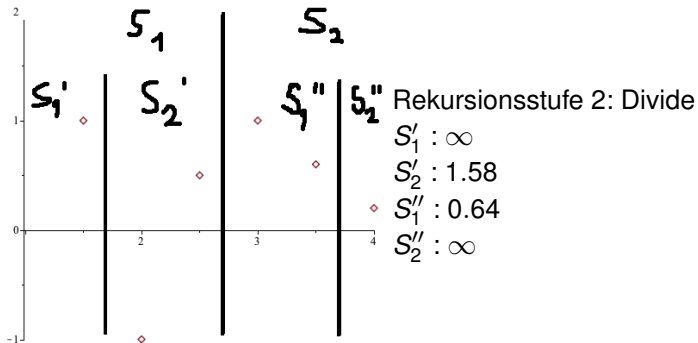
⇒ **Gesamtlaufzeit:**

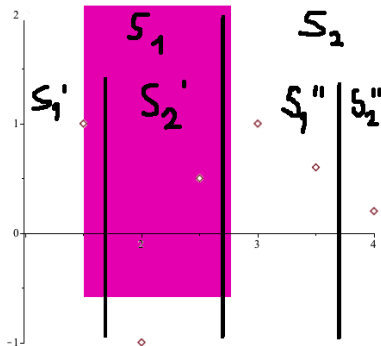
$T(n) = 2 * T(n/2) + \mathcal{O}(n)$ , und es gilt:  $T(n) \in \mathcal{O}(n \log n)$





Rekursionsstufe 1: Divide





**Pinkes Rechteck**

Höhe:  $2 * \min(\infty, 1.58)$

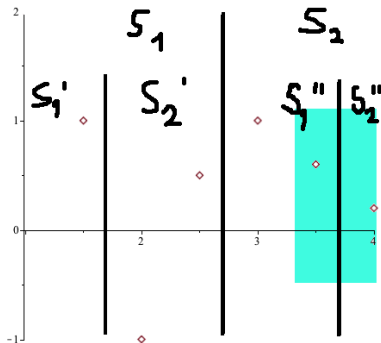
Breite:  $\min(\infty, 1.58)$

$d'_1 : \infty$

$d'_2 : 1.58$

$d'_3 : 1.58$

$S_1 : \min(\infty, 1.58, 1.58) = 1.58$



**Türkises Rechteck**

Höhe:  $2 * \min(0.64, \infty)$

Breite:  $2 * \min(0.64, \infty)$

$d_1' : 0.64$

$d_2' : \infty$

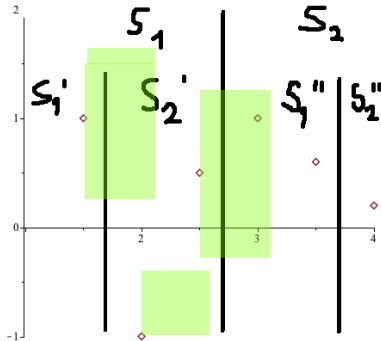
$d_3' : 0.64$

$S_2 : \min(0.64, \infty, 0.64) = 0.64$

## Rechtecke

Höhe:  $2 * \min(1.64, 0.64)$

Breite:  $2 * \min(1.64, 0.64)$



$d_1' : 1.64$

$d_2' : 0.64$

$d_3' : 0.707$

Gesamt :  $\min(1.64, 0.64, 0.707) = 0.64$

Closest Pair ist  $p_5$  und  $p_6$   
mit Abstand 0.64



## Sweepline

- Häufige Methode zum Lösen geometrischer Probleme
- Gesamte Ebene wird mit einer Linie gescannt (Scanline)
- Nur an bestimmten, wichtigen Punkten (Events) muss etwas getan werden

## Beispiele

- Graham-Scan, Sweepline scannt um einen Punkt rotierend
- Closest-Pair, klassisch

## Aufgabe

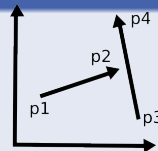
- $n$  Strecken in der Ebene, jeweils gegeben durch die beiden Endpunkte
- **Aufgabe:** Finde alle Schnittpunkte

## Vereinfachungen

- keine zwei End-/Schnittpunkte haben die gleiche x-Koordinate
- kein Endpunkt liegt auf einer anderen Strecke
- max. 2 Strecken schneiden sich in einem Punkt

## Erinnerung

$$\begin{aligned} \text{Schnitt}(p_1, p_2, p_3, p_4) = \\ ccw(p_1, p_2, p_3) \cdot ccw(p_1, p_2, p_4) \leq 0 \wedge \\ ccw(p_3, p_4, p_1) \cdot ccw(p_3, p_4, p_2) \leq 0 \end{aligned}$$



## Algorithmus

- Teste für je zwei Strecken, ob sie sich schneiden
- Berechne Schnittpunkt (LGS)
- Laufzeit:  $O(n^2)$

## Idee

- Lasse Sweepline  $L$  von links nach rechts über die Ebene laufen.
- Zu jedem Zeitpunkt schneidet  $S$  eine Teilmenge der Strecken. Die vertikale Anordnung verändert sich dabei nur bei einem Schnittpunkt.
- Events sind
  - noch nicht gescannte Endpunkte
  - Schnittpunkte von Strecken, die in der vertikalen Anordnung nebeneinander liegen

## Algorithmus - Initialisierung

- 1 Erstelle Priority Queue  $pq$  für zukünftige Events, priorisiert nach x-Koordinate.  $pq$  enthält zu Beginn alle Endpunkte.
- 2 Erstelle Set  $T$  für vertikale Anordnung der Schnittpunkte zwischen den Strecken und der Sweepline. Sortierung nach y-Koordinate. Zu Beginn leer.

## Algorithmus - Fortsetzung

- ③ Solange  $pq$  nicht leer ist, entferne erstes Element aus  $pq$ .
  - **Linker Endpunkt einer Strecke  $s$ :** Füge  $s$  in  $T$  ein. Suche Strecken  $r$  und  $t$  direkt über und unter  $s$  (falls sie existieren). Falls ihr Schnittpunkt als Event in  $pq$  liegt, entferne ihn. Falls  $s$  die Strecken  $r$  oder  $t$  schneidet, füge die Schnittpunkte in  $pq$  ein.
  - **Rechter Endpunkt einer Strecke  $s$ :** Suche Strecken  $r$  und  $t$  direkt über und unter  $s$ . Falls sie sich noch schneiden, füge Schnittpunkt zu  $pq$  hinzu. Entferne  $s$  aus  $T$ .
  - **Schnittpunkt zweier Strecken  $s, t$ :** Tausche Positionen von  $s$  und  $t$  in  $T$ . Finde Strecken  $o$  und  $u$  darüber und darunter. Entferne Schnittpunkte mit diesen, füge neue ein.