

# Übung 13 – Algorithmen II

Michael Axtmann – [michael.axtmann@kit.edu](mailto:michael.axtmann@kit.edu)

[http://algo2.iti.kit.edu/AlgorithmenII\\_WS16.php](http://algo2.iti.kit.edu/AlgorithmenII_WS16.php)

Institut für Theoretische Informatik - Algorithmik II

```
    result = current_weight;
    return true;
}

for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){
    const Edge & edge = graph.getEdge( eid );
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ); )
    if( edge.forward ){
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ); )
        weight new_weight = edge.weight + current_weight;
        GUARANTEE( new_weight >= current_weight, std::runtime_error, "Weight overflow detected." );
        if( !priority_queue.isReached( edge.target ) ){
            COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES ); )
            COUNTING( statistic_data.inc( DijkstraStatisticData::REACHED_NODES ); )
            priority_queue.push( edge.target, new_weight );
        } else {
            if( priority_queue.getCurrentKey( edge.target ) > new_weight ){
                COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_NODES ); )
                priority_queue.decreaseKey( edge.target, new_weight );
            }
        }
    }
}
```

## Klausur und Anmeldung

- Klausur am 22.02.2017 um 14:30 Uhr
- Klausuranmeldung inzwischen möglich  
→ noch heute einen der begehrten Plätze sichern

## Klausur und Anmeldung

- Klausur am 22.02.2017 um 14:30 Uhr
- Klausuranmeldung inzwischen möglich  
→ noch heute einen der begehrten Plätze sichern

## Klausur und Anmeldung

- Klausur am 22.02.2017 um 14:30 Uhr
- Klausuranmeldung inzwischen möglich  
→ **noch heute einen der begehrten Plätze sichern**

## Weiterer Vorlesungsverlauf

- Mittwoch, 01.02.2017 (morgen)
  - Keine Vorlesung / Übung
- Dienstag, 07.02.2017 (nächste Woche)
  - Wiederholung / Fragen zur Klausur  
**Themenwünsche äußern** im Forum, per Mail, ...
  - Vorstellung der Arbeit am Institut

- Geometrie  
(was und warum?)
- Sweepline
  - allgemeines Prinzip
  - Beispiel: Berechnung einer Skyline
- Linienschnitt  
(überlappende Liniensegmente)
- Rechts oder Links?  
(Orientierung eines Punktes zu einer Linie)

- geometrische **Varianten** bekannter Probleme
  - Spezialfälle oft einfacher
  - allgemeines TSP: nicht approximierbar (wenn  $P \neq NP$ )
  - metric TSP: 1.5-Approximation
  - euclidean TSP:  $\epsilon$ -Approximation
    - Laufzeit in  $O(n(\log n)^{O(\frac{1}{\epsilon} \sqrt{d})^{d-1}})$
  
- geometrisch motivierte Probleme
  - Punktlokalisierung
  - Bewegungsplanung (Robotik)
  - Sichtbarkeitsgraphen/Prüfung
  - Streckenschnitt
  - ...

## Datenstrukturen

### ■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **B**inary**S**pace**P**artition Tree (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

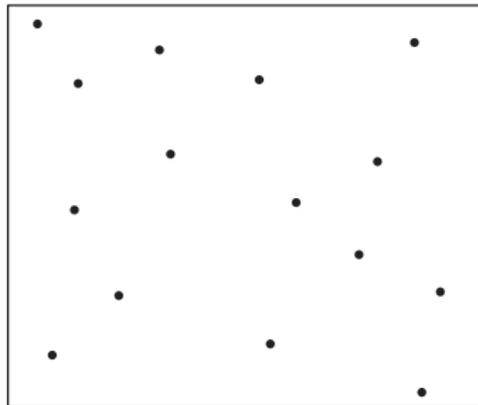
### ■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

## Strukturierter Zugriff

### ■ Sweepline

- sortiert
- topologisch sortiert



## Datenstrukturen

### ■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **B**inary**S**pace**P**artition Tree (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

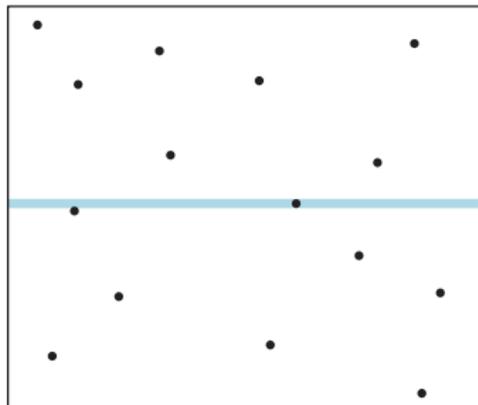
### ■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

## Strukturierter Zugriff

### ■ Sweepline

- sortiert
- topologisch sortiert



## Datenstrukturen

### ■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **BinarySpacePartition** Tree (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

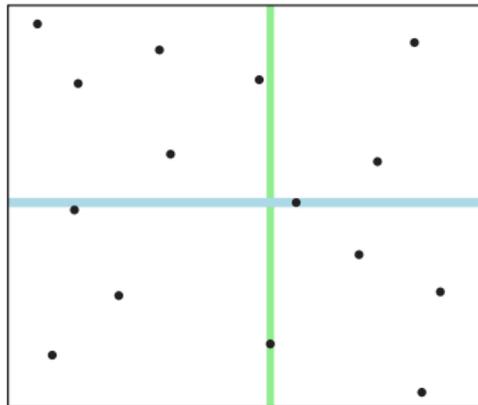
### ■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

## Strukturierter Zugriff

### ■ Sweepline

- sortiert
- topologisch sortiert



## Datenstrukturen

### ■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **B**inary**S**pace**P**artition Tree (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

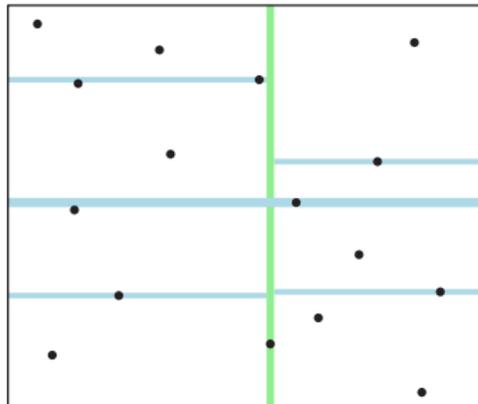
### ■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

## Strukturierter Zugriff

### ■ Sweepline

- sortiert
- topologisch sortiert



## Datenstrukturen

### ■ Baumstrukturen

- Interval Tree (1-dim)
- Quad Tree (2-dim)
- **BinarySpacePartition** Tree (3-dim)
- k-d-Tree (n-dim)
- Wavelet-Tree (2-dim)

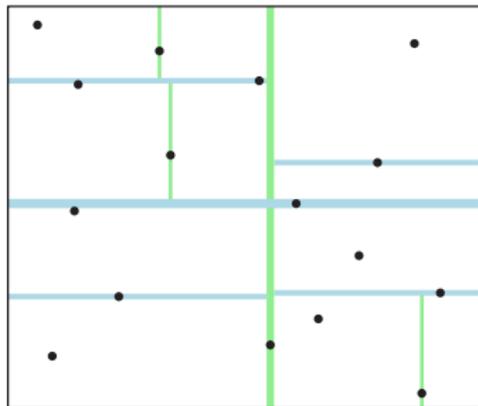
### ■ Facetten

- Delaunay Triangulierung
- Voronoi Diagramm

## Strukturierter Zugriff

### ■ Sweepline

- sortiert
- topologisch sortiert



### Idee

- strukturierte Abarbeitung eines Problems
- nutze Nähe aus
  - geometrisch nahe Objekte beeinflussen sich
  - geometrisch weit entfernte Objekte (nahezu) unabhängig

### im Allgemeinen

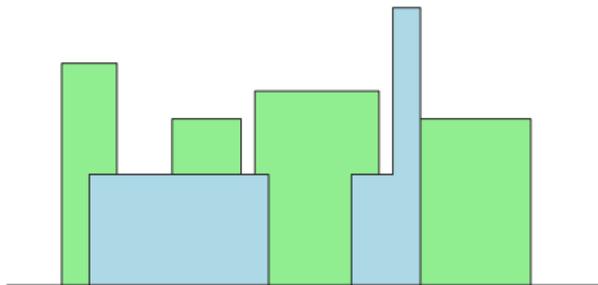
- reduziere  $n$ -dim  $\rightarrow (n - 1)$ -dim

# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

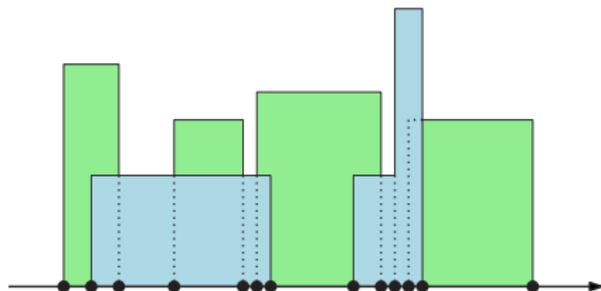


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

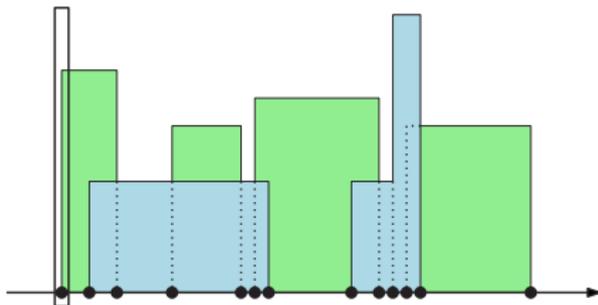


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

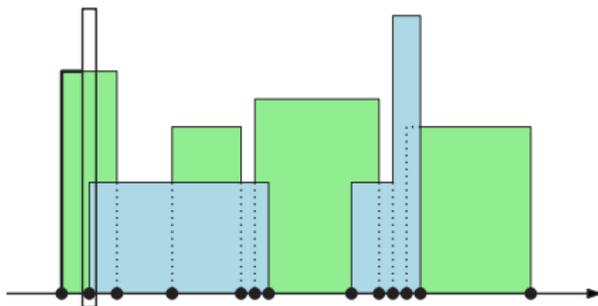


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

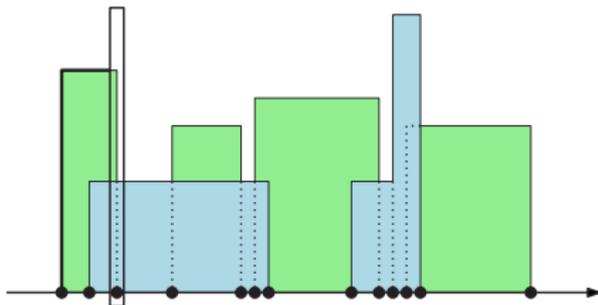


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

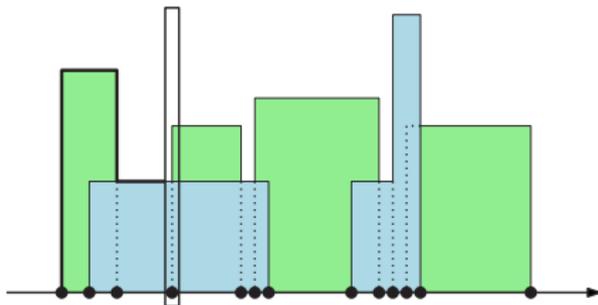


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

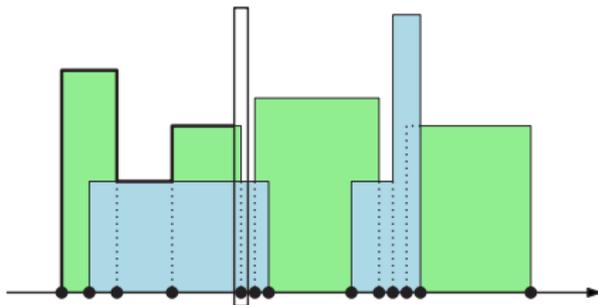


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

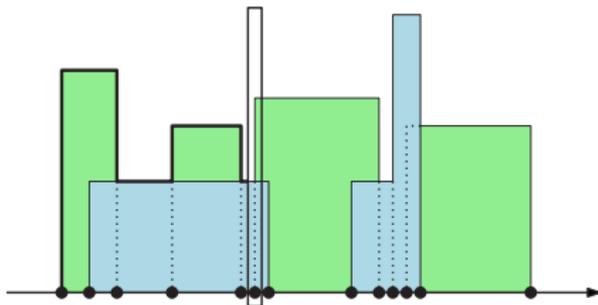


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

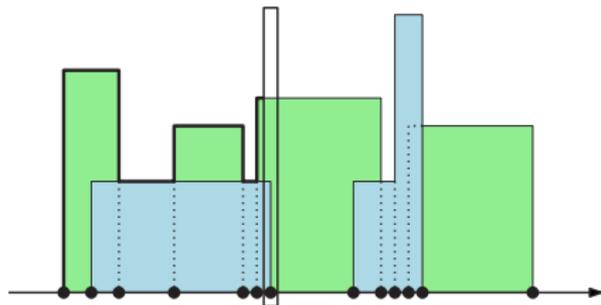


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

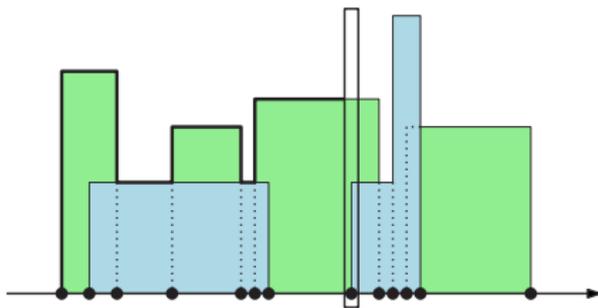


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

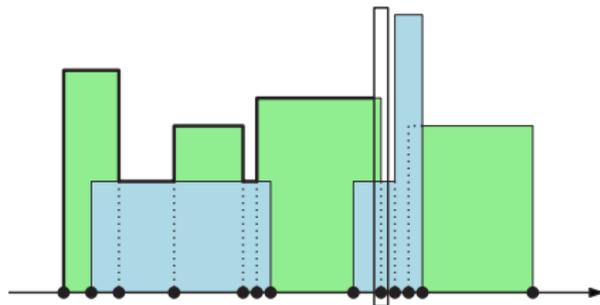


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

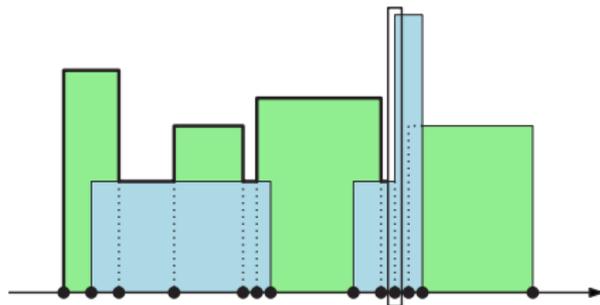


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

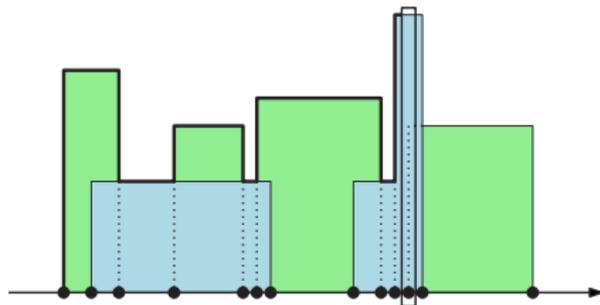


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

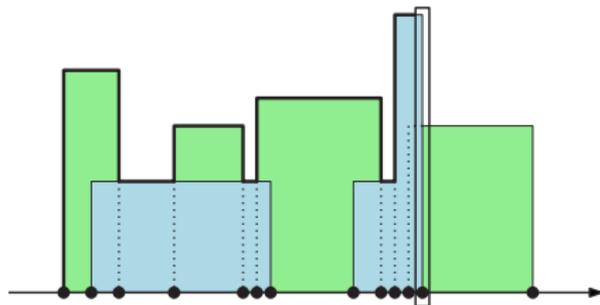


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

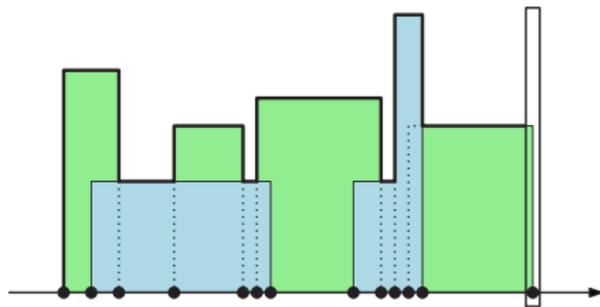


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

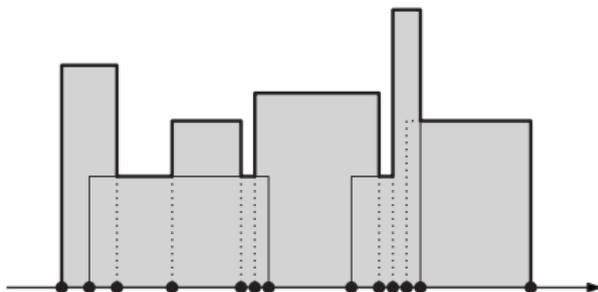


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)

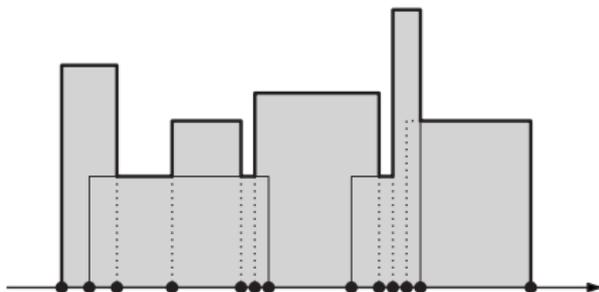


# Sweep-Line

## Beispiel: Skyline

### Berechnung einer Skyline

- Höhenänderungen sind einzig relevante Punkte
- jede Änderung definiert eindimensionales Problem (Maximumsbildung)



### Problem: effiziente Lösung des 1-dimensionalen Problems

- ineffizient:  $O(n^2)$  (vergleiche Linienschnitt)
- Ziel hier: Algorithmus mit  $O(n \log n)$  Zeit

## Sortierte Liste

- Array  $\rightarrow \mathcal{O}(n)$  für Einfügen/Löschen
- Linked List  $\rightarrow \mathcal{O}(n)$  für Positionsbestimmung

## Sortierte Liste

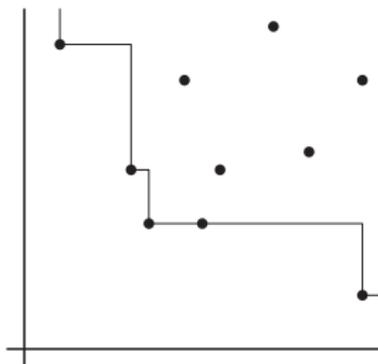
- Array  $\rightarrow \mathcal{O}(n)$  für Einfügen/Löschen
- Linked List  $\rightarrow \mathcal{O}(n)$  für Positionsbestimmung

## Lösung: Priority Queue

- alle Operationen in maximal  $\mathcal{O}(\log n)$  möglich

## Das eigentliche Skyline Problem:

- Berechnung nicht dominierter Punkt Mengen
  - multikriterielle Bewertung von Tupeln  $(t_1, \dots, t_n)$
  - Dominierungsbedingung:  
 $a$  dominiert  $b$  genau dann wenn
    - $\forall i : a_i \leq b_i$
    - $\exists i : a_i < b_i$



# Linienschnitt

## überlappende Liniensegmente

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

1. Steigung des Segments
2.  $y$ -Achsenabschnitt
3.  $x$ -Koordinaten der Punkte

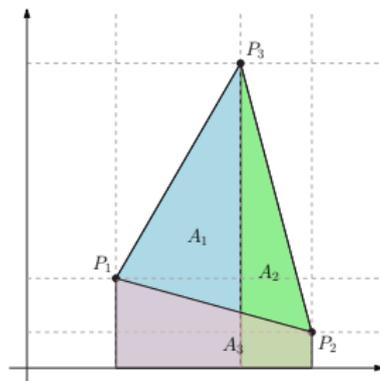
→ gruppiert genau Elemente die auf einer Geraden liegen  
(nicht zwingend überlappend)

### Linearer Scan

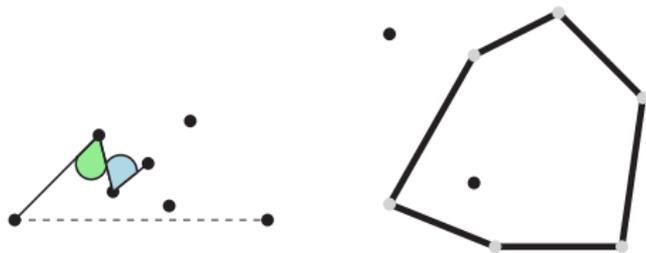
- speichere aktive Elemente in Priority Queue  $\mathcal{O}(n \log n)$ 
  - sortiert nach  $x$ -Koordinate
  - beenden aktuelle Segmente
- alle gleichzeitig aktiven Elemente überlappen

## Vorzeichenbehaftete Fläche

- Dreiecksfläche als Summe dreier Trapezflächen
- Verschiebung von  $P_1$  nach  $(0, 0)$  ergibt:
$$A = \frac{1}{2} \begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix}$$
- Vorzeichen positiv gdw.  $P_1, P_2, P_3$  CCW

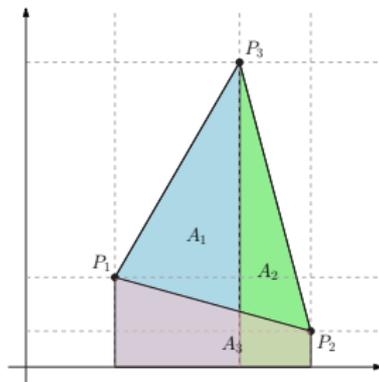


- Unterproblem von Algorithmen
  - Graham Scan
  - Test auf Enthaltensein  
(Punkt in konvexem Polygon)

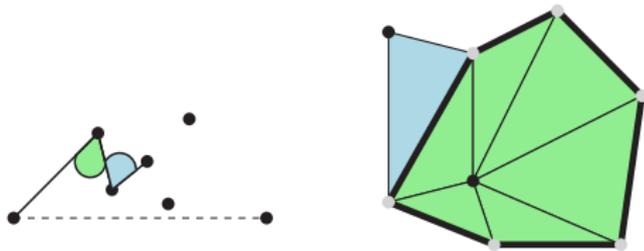


## Vorzeichenbehaftete Fläche

- Dreiecksfläche als Summe dreier Trapezflächen
- Verschiebung von  $P_1$  nach  $(0, 0)$  ergibt:
$$A = \frac{1}{2} \begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix}$$
- Vorzeichen positiv gdw.  $P_1, P_2, P_3$  CCW



- Unterproblem von Algorithmen
  - Graham Scan
  - Test auf Enthaltensein  
(Punkt in konvexem Polygon)



# Ende!



# Feierabend!