

## 2.7 Geometrische Algorithmen

- 2.7.1 Inside-Test
- 2.7.2 Konvexe Hülle
- 2.7.3 Nachbarschaften
- 2.7.4 Schnittprobleme



## Nachbarschaften

- Gegeben sei eine Punktmenge  $p_1, \dots, p_n$
- Problemstellungen
  - geringster Abstand zwischen zwei  $p_i$
  - $k$  nächste Punkte zu  $p_i$
  - $k$  nächste Punkte zu  $(x, y)$
  - alle Punkte innerhalb eines Kreises  $(x, y, r)$
  - ...



## Maximale Punktdichte

- Gegeben: Punktmenge  $p_1, \dots, p_n$
- $d_{ij} = \|p_j - p_i\|$
- finde  $\min_{ij} \{d_{ij}\}$
- triviale Lösung in  $O(n^2)$
- schnellerer Algorithmus?





## Maximale Punktdichte

- Versuche bestimmte Punktpaare  $p_i, p_j$  ohne weitere Berechnung auszuschliessen
- Ansatz: Divide & Conquer
  - Teile die Menge  $P = \{p_1, \dots, p_n\}$  in  $Q_1 = \{p_1, \dots, p_{n/2}\}$  und  $Q_2 = \{p_{n/2+1}, \dots, p_n\}$
  - minimaler Abstand innerhalb  $Q_1$
  - minimaler Abstand innerhalb  $Q_2$
  - minimaler Abstand zwischen  $Q_1$  und  $Q_2$





### Divide & Conquer

P

5  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

### Divide & Conquer

P

6  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

**Divide & Conquer**

The diagram shows a set of points  $P$  divided into two subsets  $Q_1$  and  $Q_2$  by a vertical solid line. Within  $Q_1$ , two points are connected by a line segment labeled  $d_1$ . Within  $Q_2$ , two points are connected by a line segment labeled  $d_2$ .

7 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  
 RWTH AACHEN UNIVERSITY

**Divide & Conquer**

The diagram shows the same point set  $P$  and subsets  $Q_1$  and  $Q_2$  as in slide 7. A vertical solid line separates the two subsets. Two vertical dashed lines extend from the solid line to the points in  $Q_1$  and  $Q_2$  that are closest to the line. A bracket below these dashed lines is labeled  $d = \min \{d_1, d_2\}$ .

8 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  
 RWTH AACHEN UNIVERSITY

## 1. Algorithmus

- $\text{MinDist}(p[1..n])$ 
  - sort  $p[i]$  by increasing x-coordinate
  - $d \leftarrow \text{MinDistRec}(p[1..n])$
  - return  $d$



## 1. Algorithmus

- $\text{MinDistRec}(p[1..n])$ 
  - $d_1 \leftarrow \text{MinDistRec}(p[1..n/2])$
  - $d_2 \leftarrow \text{MinDistRec}(p[n/2+1..n])$
  - $d \leftarrow \min(d_1, d_2)$
  - $x \leftarrow (p[n/2].x + p[n/2+1].x) / 2$
  - $Q_1 \leftarrow \text{select } p[i \leq n/2] \text{ where } |p[i].x - x| \leq d$
  - $Q_2 \leftarrow \text{select } p[i > n/2] \text{ where } |p[i].x - x| \leq d$
  - $d_3 \leftarrow \text{MinDistCross}(Q_1, Q_2)$
  - return  $\min(d, d_3)$



## 1. Algorithmus

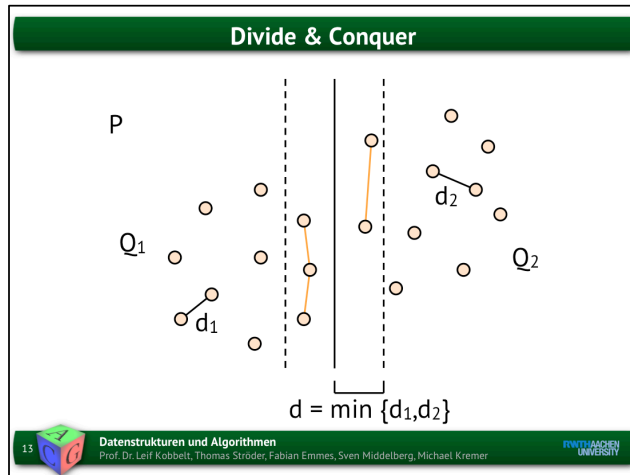
- $T(n) = 2 T(n/2) + O(n^2)$
- $T(n) = O(n^2)$
- ... nicht besser als die triviale Lösung



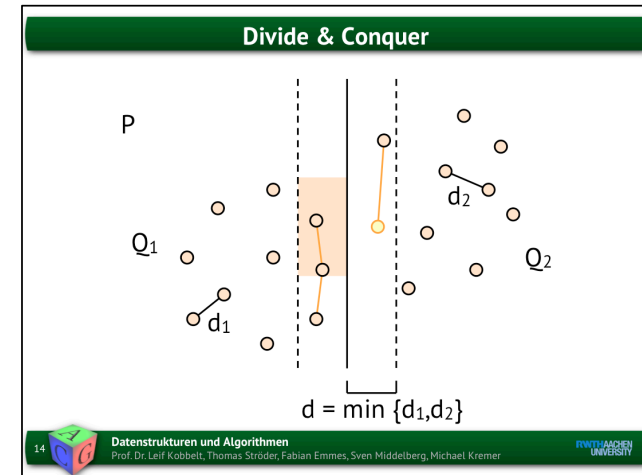
## 2. Algorithmus

- Die Kandidaten für minimale Distanzen in der Menge  $Q$  können weiter eingeschränkt werden ...





Bilder Korridor um Teillinie, in welcher das globale Minimum liegen kann (da wir bereits zwei Punkte mit Abstand  $\min\{d_1, d_2\}$  gefunden haben)

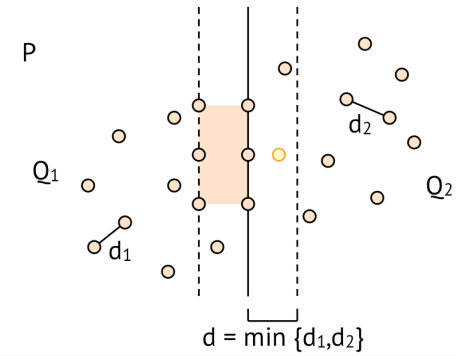


## 2. Algorithmus

- Die Kandidaten für minimale Distanzen in der Menge  $Q$  können weiter eingeschränkt werden ...
- ... jeder Punkt muss mit maximal sechs Kandidaten verglichen werden



## Divide & Conquer



Wie finden wir die Punkte mit welchen man einen Punkt der anderen Seite vergleichen muss? Sortiere Punkte!



## 2. Algorithmus

- MinDistRec( p[1..n] )  
   $d_1 \leftarrow \text{MinDistRec}( p[1..n/2] )$   
   $d_2 \leftarrow \text{MinDistRec}( p[n/2+1..n] )$   
   $d \leftarrow \min( d_1, d_2 )$   
   $x \leftarrow ( p[n/2].x + p[n/2+1].x ) / 2$   
   $Q_1 \leftarrow \text{select } p[i \leq n/2] \text{ where } |p[i].x - x| \leq d$   
   $Q_2 \leftarrow \text{select } p[i > n/2] \text{ where } |p[i].x - x| \leq d$   
  sort  $Q_1, Q_2$  by increasing y-coordinate  
   $d_3 \leftarrow \text{MinDistRestrict}(Q_1, Q_2, d)$   
  return  $\min(d, d_3)$



## 2. Algorithmus

- MinDistRestrict( q[1..k], q'[1..k'], d )  
  dist  $\leftarrow d$   
   $l \leftarrow 1, r \leftarrow 1$   
  for i  $\leftarrow 1$  to k do  
    while  $q'[l].y < q[i].y - d$  and  $l < k'$  do  
       $l \leftarrow l + 1$   
    while  $q'[r].y < q[i].y + d$  and  $r \leq k'$  do  
       $r \leftarrow r + 1$   
    for j  $\leftarrow l$  to  $r - 1$  do  
      dist  $\leftarrow \min(\text{dist}, \text{dist}(q[i], q'[j]))$   
  return dist



## 2. Algorithmus

```
• MinDistRestrict( q[1..k], q'[1..k'], d )
  dist ← d
  l ← 1, r ← 1
  for i ← 1 to k do
    while q'[l].y < q[i].y-d and l < k' do
      l ← l + 1
    while q'[r].y < q[i].y+d and r ≤ k' do
      r ← r + 1
    for j ← l to r - 1 do
      dist ← min(dist, dist(q[i], q'[j]))
  return dist
```

$O(n)$  [bracketed next to the for loop]

$O(1)$  [bracketed next to the while loops]

$O(1)$  [bracketed next to the for loop]



## 2. Algorithmus

```
• MinDistRec( p[1..n] )
  d1 ← MinDistRec( p[1..n/2] )
  d2 ← MinDistRec( p[n/2+1..n] )
  d ← min( d1, d2 )
  x ← ( p[n/2].x + p[n/2+1].x ) / 2
  O(n) [ Q1 ← select p[i ≤ n/2] where |p[i].x - x| ≤ d
  O(n) [ Q2 ← select p[i > n/2] where |p[i].x - x| ≤ d
  O(n log n) [ sort Q1, Q2 by increasing y-coordinate
  O(n) [ d3 ← MinDistRestrict(Q1, Q2, d)
  return min(d, d3)
```



## 2. Algorithmus

- $T(n) = 2 \times T(n/2) + O(n) + O(n \times \log n)$
- $T(n) = 2 \times T(n/2) + O(n \times \log n)$
- $T(n) = O(n \times \log^2 n)$



## 3. Algorithmus

- Beobachtung: der MinDist-Algorithmus hat dieselbe Struktur wie Merge-Sort
  - Splitting in zwei gleichgroße Teile
  - Sortierung beim Zusammenfügen
- Merging von sortierten Folgen kostet nur  $O(n)$
- `MinDistRec()` gibt nach der y-Koordinate sortierte Folgen zurück ...



### 3. Algorithmus

```
• MinDistRec( p[1..n] )
  x ← ( p[n/2].x + p[n/2+1].x ) / 2
  d1 ← MinDistRec( p[1..n/2] )
  d2 ← MinDistRec( p[n/2+1..n] )
  d ← min(d1, d2)
  Q1 ← select p[i ≤ n/2] where |p[i].x - x| ≤ d
  Q2 ← select p[i > n/2] where |p[i].x - x| ≤ d
  merge pre-sorted lists Q1, Q2
  d3 ← MinDistRestrict(Q1, Q2, d)
  merge pre-sorted lists p[i ≤ n/2], p[i > n/2]
  return min(d, d3)
```



### 3. Algorithmus

```
• MinDistRec( p[1..n] )
  x ← ( p[n/2].x + p[n/2+1].x ) / 2
  d1 ← MinDistRec( p[1..n/2] )
  d2 ← MinDistRec( p[n/2+1..n] )
  d ← min(d1, d2)
  O(n) [ Q1 ← select p[i ≤ n/2] where |p[i].x - x| ≤ d
  O(n) [ Q2 ← select p[i > n/2] where |p[i].x - x| ≤ d
  O(n) [ merge pre-sorted lists Q1, Q2
  O(n) [ d3 ← MinDistRestrict(Q1, Q2, d)
  O(n) [ merge pre-sorted lists p[i ≤ n/2], p[i > n/2]
  return min(d, d3)
```



### 3. Algorithmus

- $T(n) = 2 \times T(n/2) + O(n)$
- $T(n) = O(n \times \log n)$



### Weitester Abstand

- Gegeben: Punktmenge  $p_1, \dots, p_n$
- $d_{ij} = \|p_j - p_i\|$
- finde  $\max_{ij} \{d_{ij}\}$
- Durchmesser der Punktmenge
- triviale Lösung in  $O(n^2)$
- schnellerer Algorithmus?

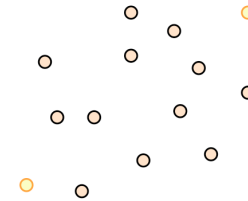


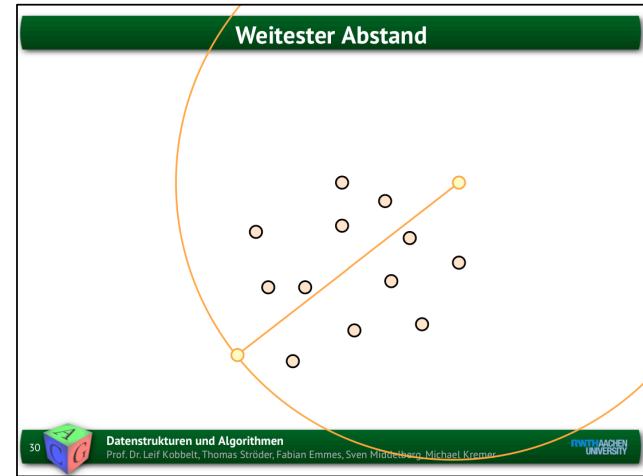
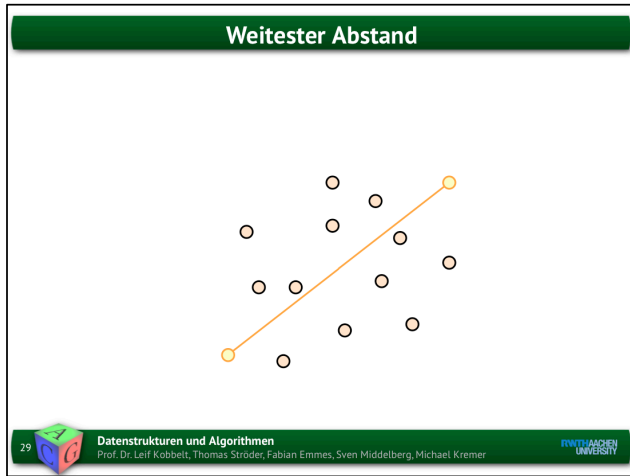
## Weitester Abstand

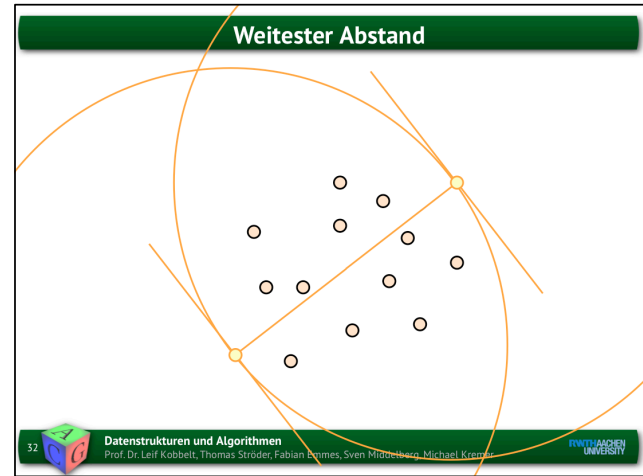
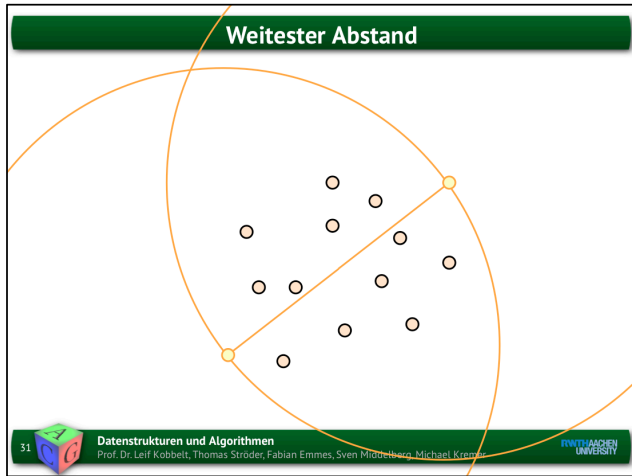
- Versuche wieder bestimmte Punktpaare auszuschließen.
- Der maximale Abstand muss zwischen Extrempunkten auftreten.
- Beschränke die Suche auf die konvexe Hülle.



## Weitester Abstand







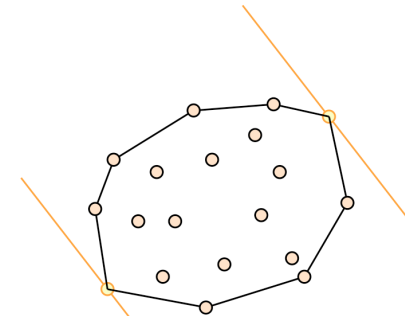


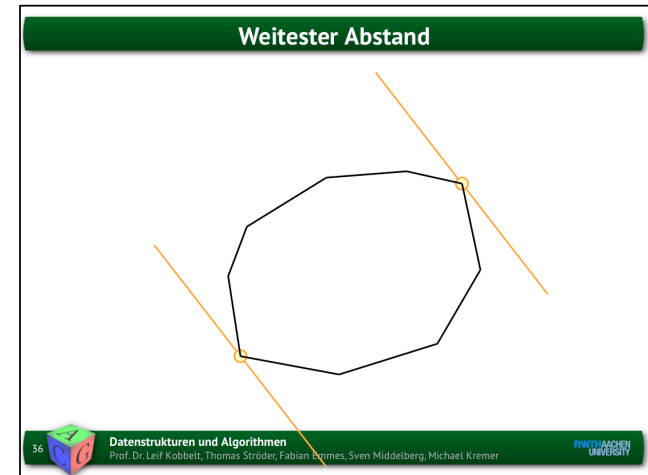
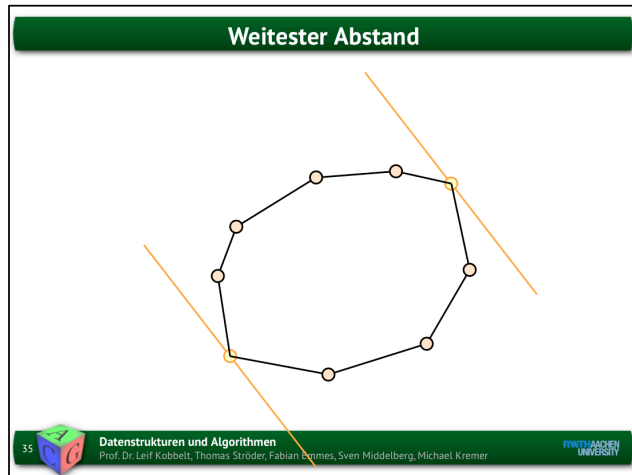
## Weitester Abstand

- Geg. Punktmenge  $p_1, \dots, p_n$ 
  1. Berechne die konvexe Hülle
  2. Suche Extrempunkte mit parallelen Supporting Lines
  3. Bestimme den maximalen Abstand zwischen diesen Kandidaten

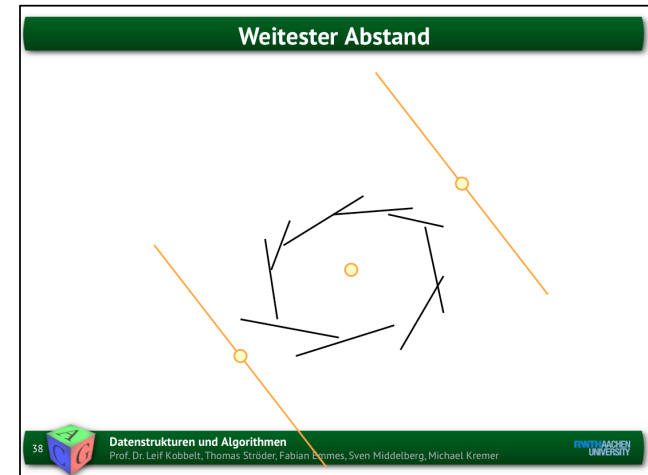
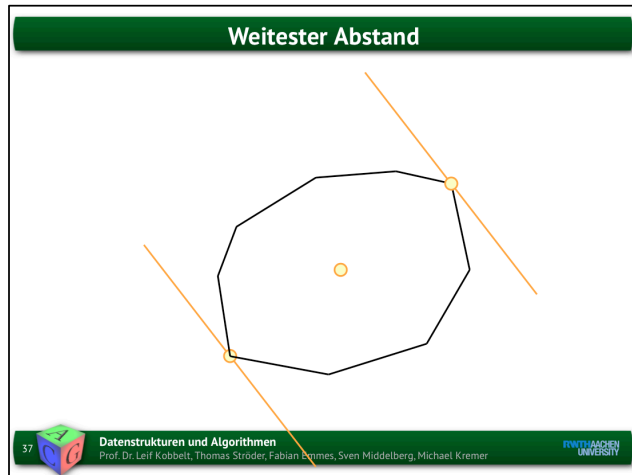


## Weitester Abstand

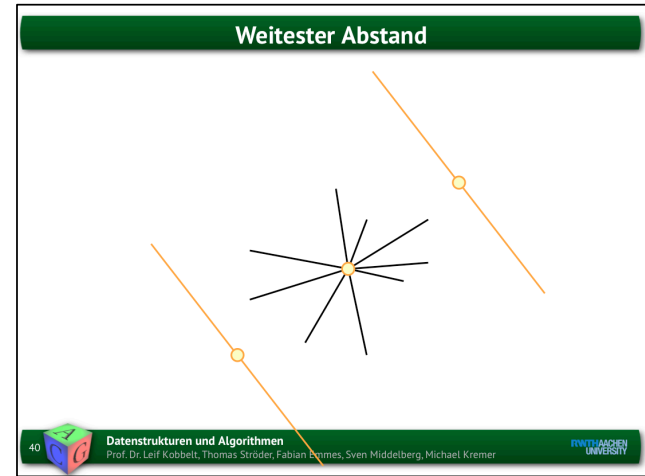
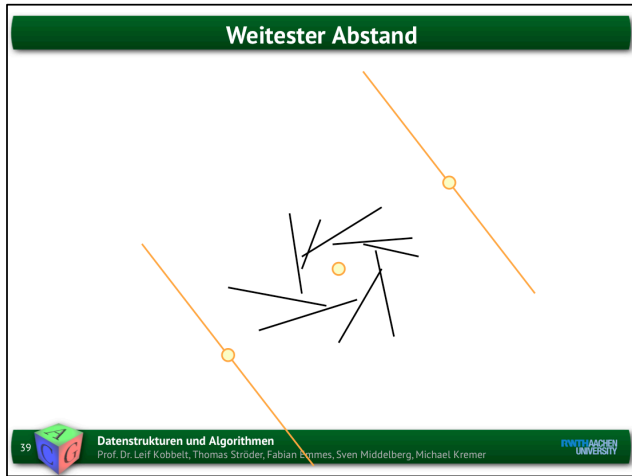


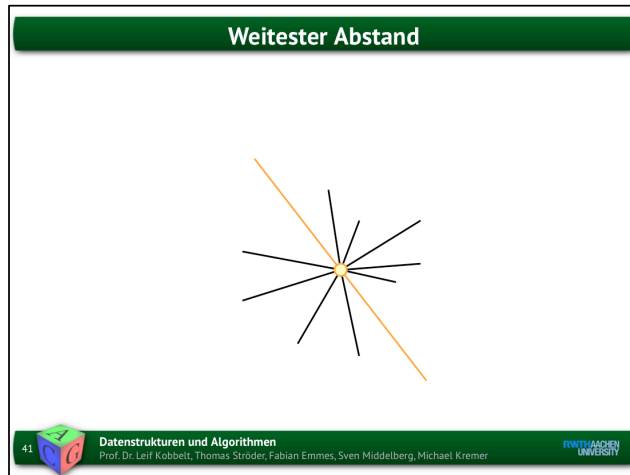


Konvexe Hülle liefert auch Sortierung der Kanten nach Winkel

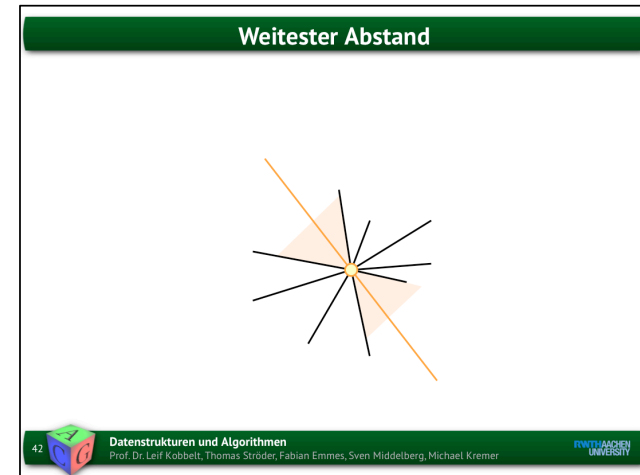


Verschiebe Kanten in Ursprung





Betrachte nur noch gegenüberliegende Sektoren



## Weitester Abstand

1. Berechne die konvexe Hülle ]  $O(n \log n)$
2. Suche Extrempunkte mit parallelen Supporting Lines
  - a. die Kanten der konvexen Hülle bilden Sektoren
  - b. gegenüberliegende Sektoren enthalten gemeinsame Gerade ]  $O(n)$
3. Finde den maximalen Abstand zwischen diesen Kandidaten ]  $O(n)$



## Nachbarschaften

- Geg. Punktmenge  $p_1, \dots, p_n$
- Typische Problemstellungen :
  - min. / max. Abstand zwischen zwei  $p_i$
  - $k$  nächste Punkte zu  $p_i$
  - $k$  nächste Punkte zu  $(x,y)$
  - alle Punkte innerhalb eines Kreises  $(x,y,r)$
  - ...



## Nächste Punkte

- Beispiele:
  - Datenbanksuche mit Unsicherheit
  - Finde nächste Mobilfunkstation
  - Globale Beleuchtungssimulation
  - ...



## Nächste Punkte

- Geg. Punktmenge  $p_1, \dots, p_n$
- Anfrage:  $(x, y)$  [ hier:  $\mathbb{R}^2$ , allg.:  $\mathbb{R}^k$  ]
- Optimale Datenstruktur zum Suchen  
⇒ Bäume
- Mehrdimensionale Schlüssel  
⇒ z.B. kD-Bäume



### Quadtree

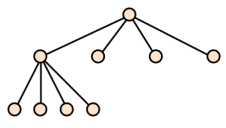
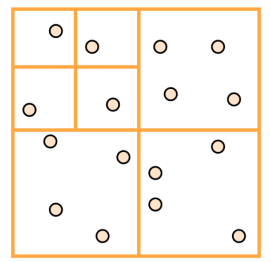
47 **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  
RWTH AACHEN UNIVERSITY

### Quadtree

48 **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  
RWTH AACHEN UNIVERSITY




### Quadtree

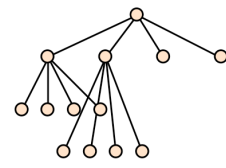
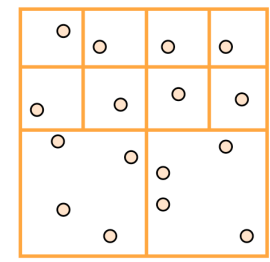



49

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




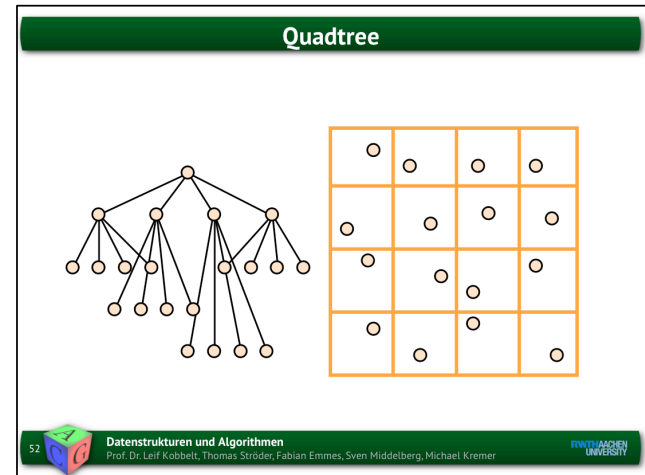
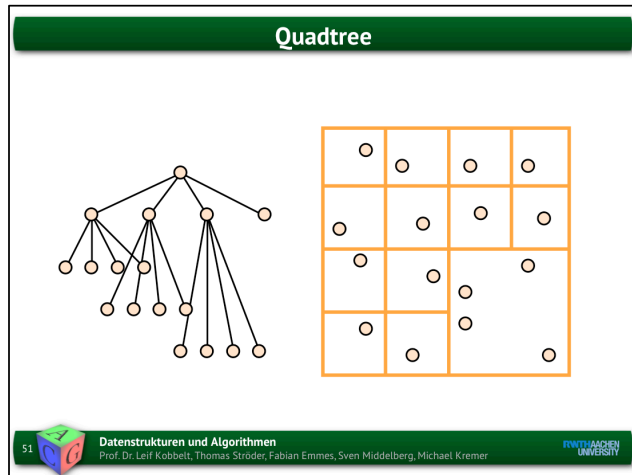
### Quadtree


50

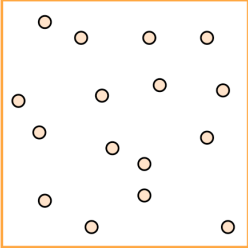
**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer






### kD-Bäume



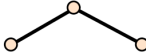


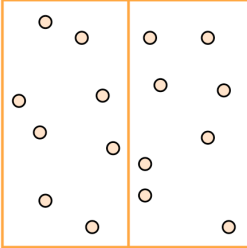
53

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




### kD-Bäume



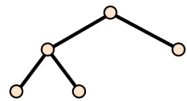
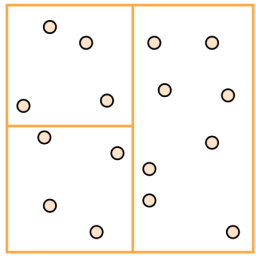


54

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




### kD-Bäume

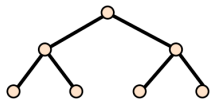
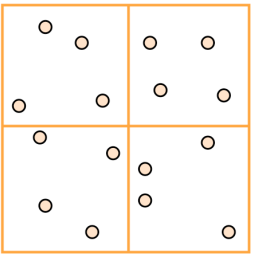



55

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




### kD-Bäume

56

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



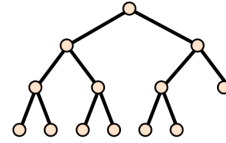
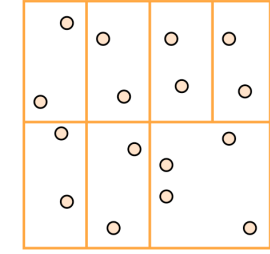
**kD-Bäume**

57 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

**kD-Bäume**


58 **Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

### kD-Bäume

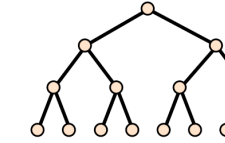
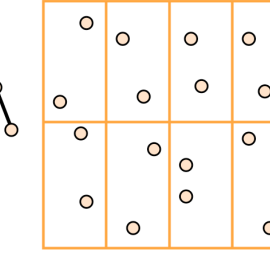



59

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




### kD-Bäume

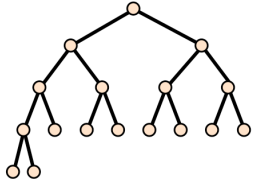
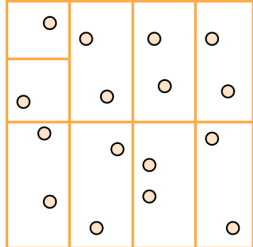



60

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




### kD-Bäume

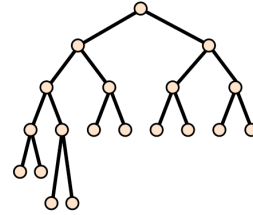
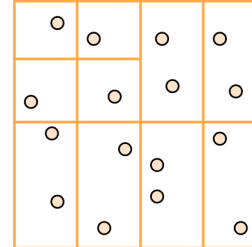



61

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




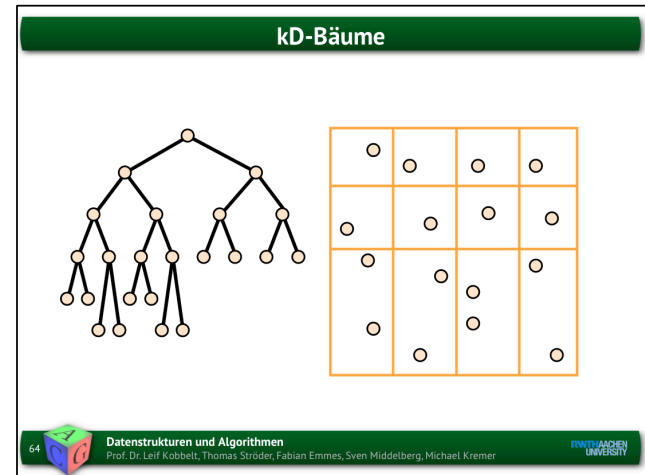
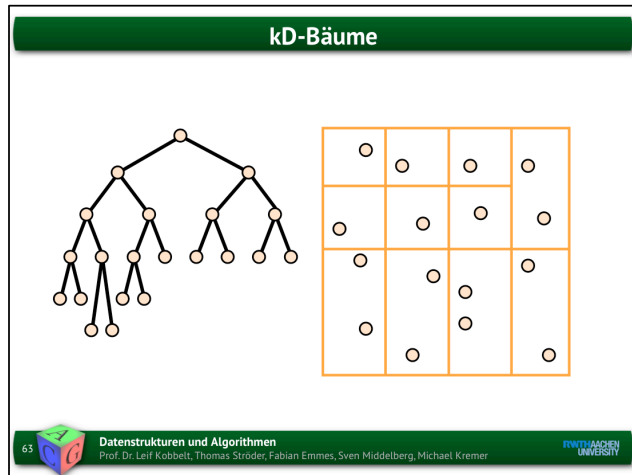
### kD-Bäume

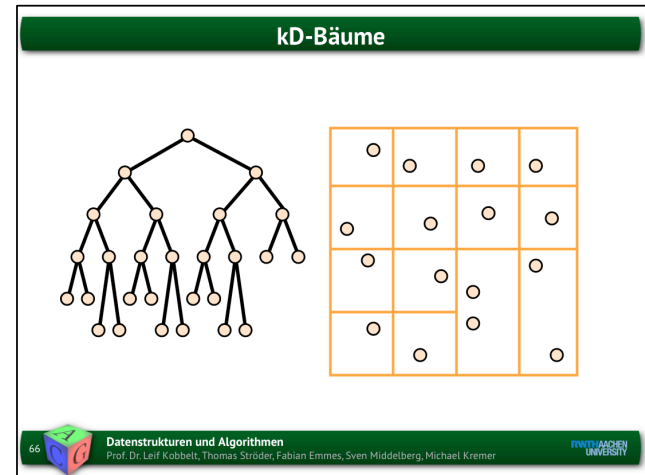
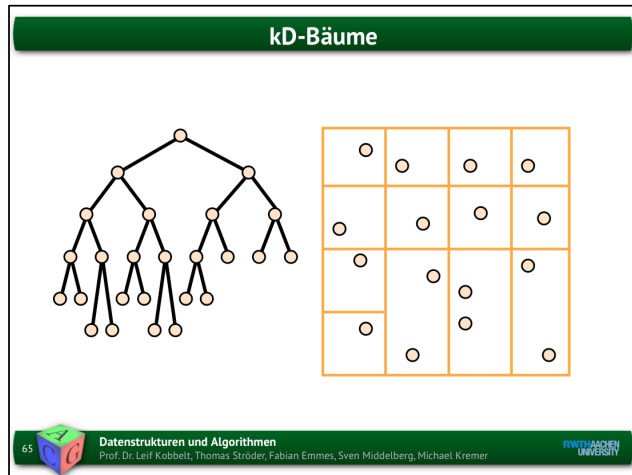
62

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

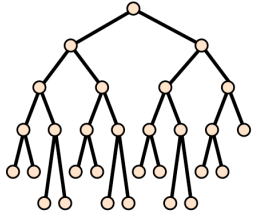
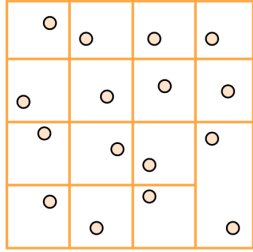









### kD-Bäume

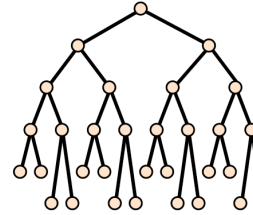
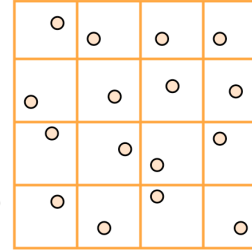



67

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer




### kD-Bäume

68

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer



## kD-Baum Suche

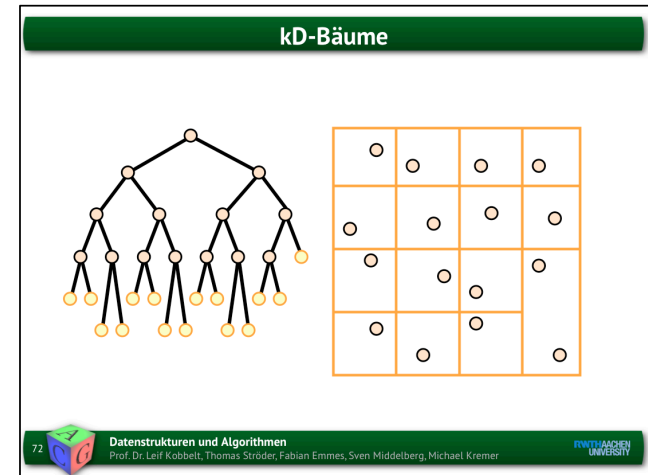
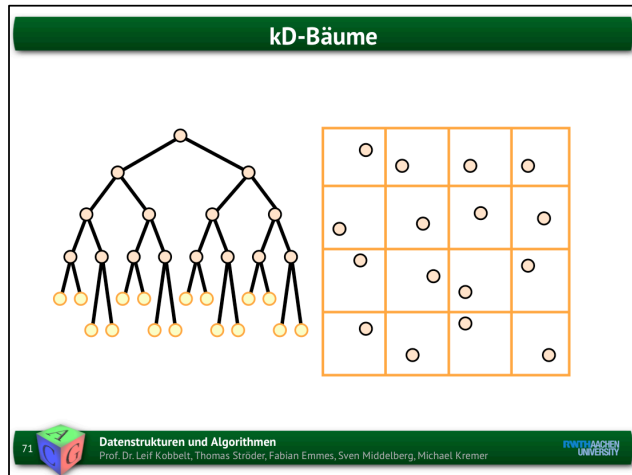
- In welcher Zelle liegt der Anfragepunkt
- In welchen Zellen muß nach dem nächsten Punkt gesucht werden
- Effizienter Ausschluß weit entfernter Zellen



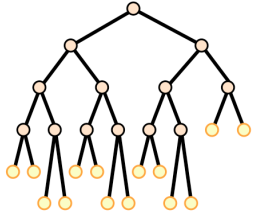
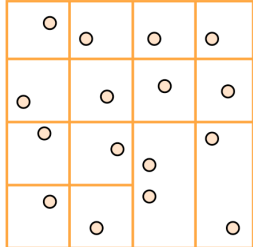
## kD-Baum Suche

- Jedes kD-Baum Blatt enthält einen der gegebenen Punkte  $p_i$
- Jeder innere kD-Baum Knoten enthält eine Ebene, die die Punkte im linken und rechten Teilbaum separiert.





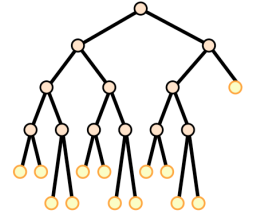
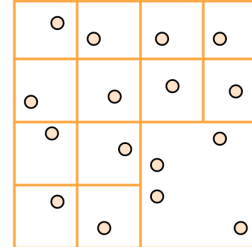
### kD-Bäume

73

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

### kD-Bäume

74

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

## kD-Baum Suche

- Traverse(p,n)
  - if n is a leaf-node then
    - return ( dist(p,point[n]), point[n] )
  - else
    - if p lies in positive half-space of n then
      - (dist,near) ← Traverse(p,left[n])
    - else
      - (dist,near) ← Traverse(p,right[n])



## kD-Baum Suche

- Traverse(p,n)
  - if n is a leaf-node then
    - return ( dist(p,point[n]), point[n] )
  - else
    - if  $p \cdot \text{normal}[n] - \text{offset}[n] \geq 0$  then
      - (dist,near) ← Traverse(p,left[n])
    - else
      - (dist,near) ← Traverse(p,right[n])



### kD-Baum Suche

77

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer


### kD-Baum Suche

78

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer


### kD-Baum Suche

The diagram shows two nodes from a kD-tree. Each node is represented by a circle containing a yellow dot (the internal node) and a grey dot (the leaf node) connected by a line. A vertical orange line is drawn to the left of the yellow dot in both nodes, representing a splitting plane.

79 **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

### kD-Baum Suche

The diagram shows two nodes from a kD-tree, similar to the previous slide. In addition to the vertical orange line, dashed orange lines extend from the yellow dots to the left, representing the search path or the region being explored.

80 **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



## kD-Baum Suche

- `Traverse(p,dist,near,n)`
  - if `n` is a leaf-node then
    - if `dist(p,point[n]) < dist` then
      - return `(dist(p,point[n]), point[n])`
    - else
      - return `(dist, near)`
  - else
    - if `p·normal[n] - offset[n] ≥ -dist` then
      - `(dist,near) = Traverse(p,dist,near,left[n])`
    - if `p·normal[n] - offset[n] ≤ dist` then
      - `(dist,near) = Traverse(p,dist,near,right[n])`
    - return `(dist,near)`



## Aufwandsabschätzung

- Bei regelmäßig verteilten Datenpunkten ist der erwartete Aufwand  $O(\log n)$
- Der Algorithmus funktioniert auch für allgemeinere **BSP-Bäume**, bei denen in jedem Knoten eine beliebige separierende Ebene definiert wird.



## k nächste Punkte

- Geg. Punktmenge  $p_1, \dots, p_n$
- Anfrage:  $(x,y)$  [ hier:  $\mathbb{R}^2$ , allg.:  $\mathbb{R}^k$  ]
- Suche die  $k$  besten Kandidaten
- Verwalte die aktuell  $k$  Besten in einem Heap
- Neue Kandidaten verdrängen die alten



## k nächste Punkte

- Nearest( $p, neighbors, n$ )
    - if  $n$  is a leaf-node then
      - if  $dist(p, point[n]) < neighbors.top()$  then
        - remove\_top(neighbors)
        - add(neighbors, dist( $p, point[n]$ ), point[n])
else
      - if  $p.normal[n] - offset[n] \geq -neighbors.top()$  then
        - neighbors  $\leftarrow$  Traverse( $p, neighbors, left[n]$ )
      - if  $p.normal[n] - offset[n] \leq neighbors.top()$  then
        - neighbors  $\leftarrow$  Traverse( $p, neighbors, right[n]$ )
- return neighbors



## Aufwandsabschätzung

- Fast der gleiche Aufwand wie bei der einfachen Suche:  $O(\log n \times \log k)$



## Range Queries

- Bereichsabfragen
  - Suche alle Punkte in einem Kreis  $(x,y,r)$
  - Suche alle Punkte in einem Intervall  $[a,b] \times [c,d]$
  - ...
- Einfache Modifikation des Suchalgorithmus



## Range Queries

- `Traverse(p,maxdist,n)`
  - if `n` is a leaf-node then
    - if `dist(p,point[n]) < maxdist` then
      - output `point[n]`
    - else
      - if `p-normal[n]-offset[n] ≥ -maxdist` then
        - `Traverse(p,maxdist,left[n])`
      - if `p-normal[n]-offset[n] ≤ maxdist` then
        - `Traverse(p,maxdist,right[n])`



## Abstandsdiagramme

- Bei häufigen Nachbarschafts-Anfragen kann der Zugriff durch Vorberechnung von Abstandsdiagrammen beschleunigt werden
- Diese zerlegen den  $\mathbb{R}^n$  (hier den  $\mathbb{R}^2$ ) in kleine Bereiche, für die die Nachbarschaften jeweils vorberechnet werden können



Gewünscht: Graph-Datenstruktur, in der jeder Punkt mit seinen (geometrischen) Nachbarn verbunden ist. Dann ist die Suche an sich wenig komplex. Nur der Aufbau des Graphen ist etwas schwieriger.

### Voronoi-Gebiete

- Geg. Punktmenge  $p_1, \dots, p_n$
- Das **Voronoi-Gebiet** zu einem Punkt  $p_i$  enthält den Bereich des  $\mathbb{R}^2$ , der näher an  $p_i$  als an jedem anderen  $p_j$  liegt
- $V(p_i) = \{ q \mid \forall j \neq i : \text{dist}(q, p_i) < \text{dist}(q, p_j) \}$

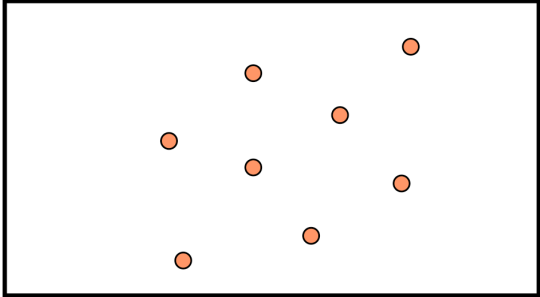
89

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG

Jeder Punkt soll Informationen darüber haben, was seine Nachbarn sind.  
 Beispiel: Funkmasten. Für jeden Punkt in der Ebene will man wissen, welcher der geometrisch nächste Funkmast ist, weil dieser lokal die größte Feldstärke hat.

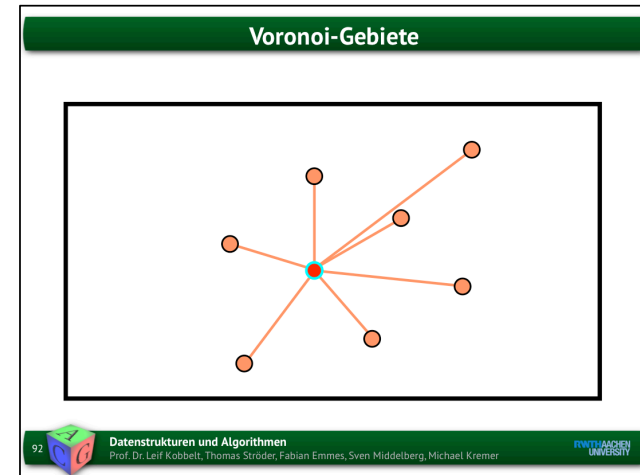
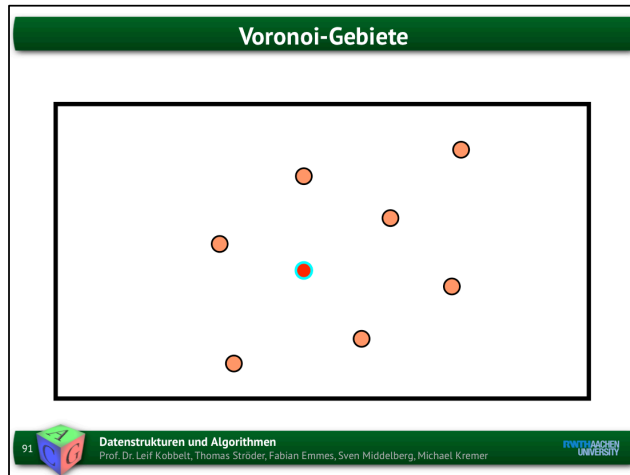
### Voronoi-Gebiete



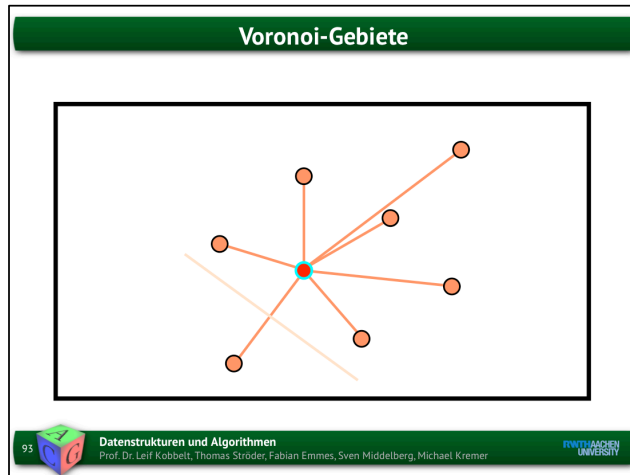
90

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

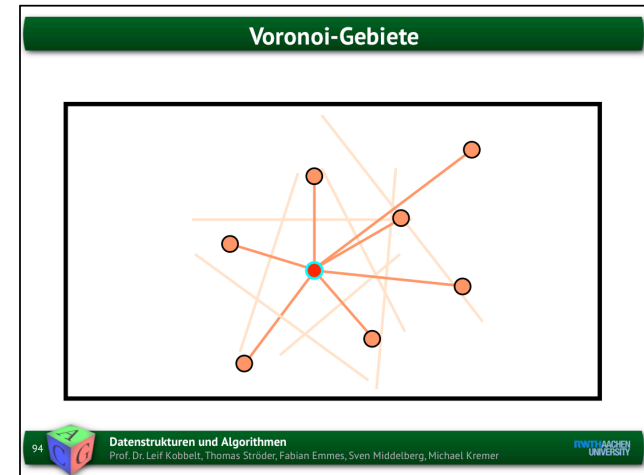
FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG



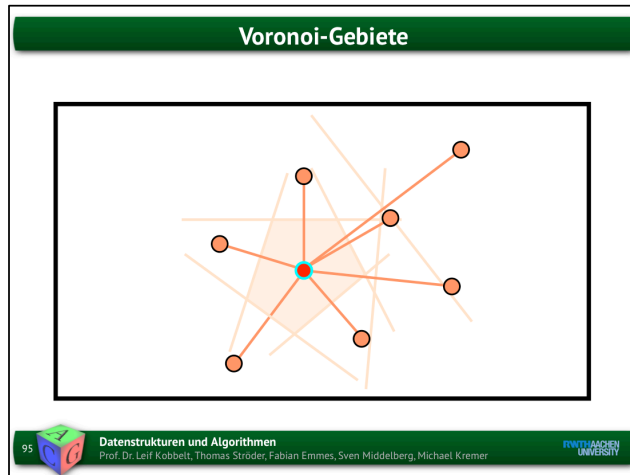
Bilde Gerade zu jedem anderen Punkt.



Die Senkrechte teilt den Raum in zwei Halbräume, für die gilt:  
 Alle Punkte oberhalb der Geraden sind näher am oberen Punkt (rot)  
 und alle Punkte unterhalb dieser Geraden sind näher zum unteren Punkt.



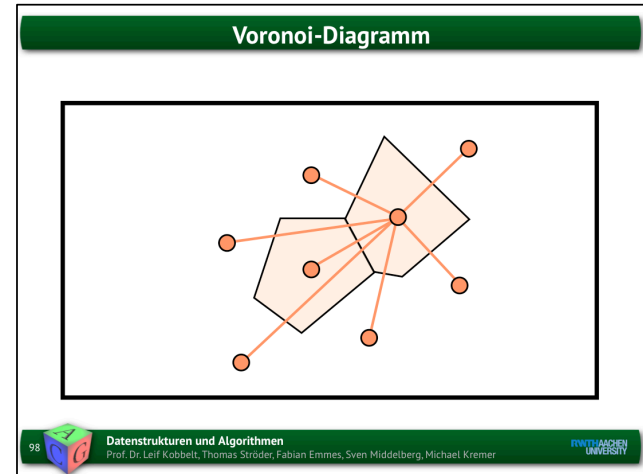
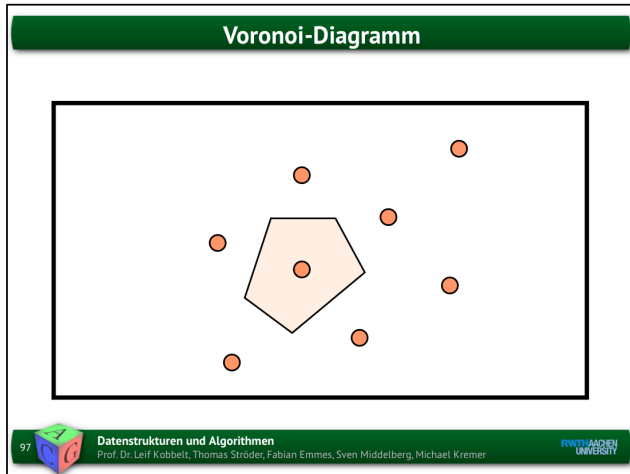
Diese Halbräume bilden wir für alle möglichen Punkte.

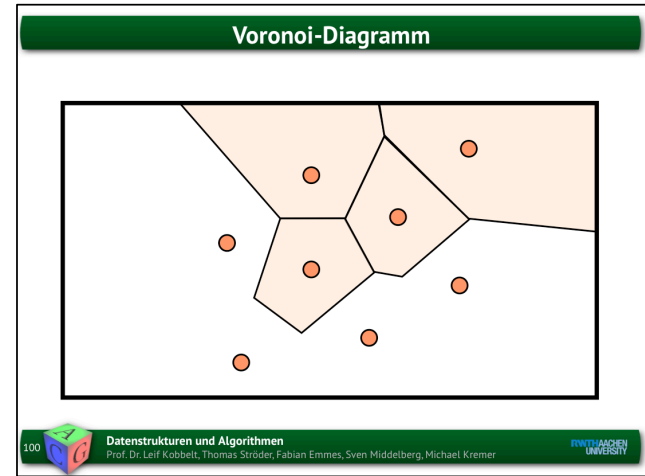
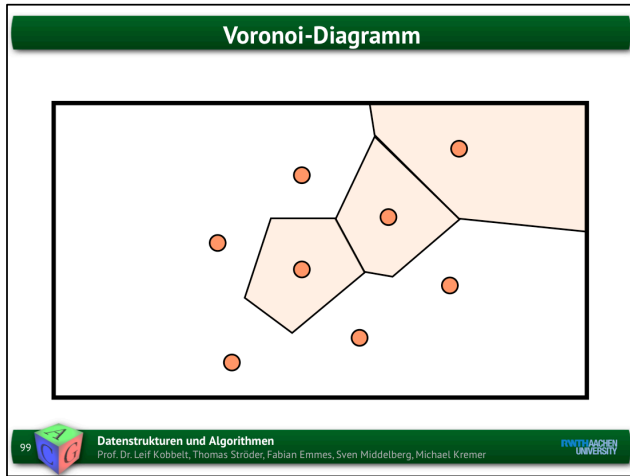


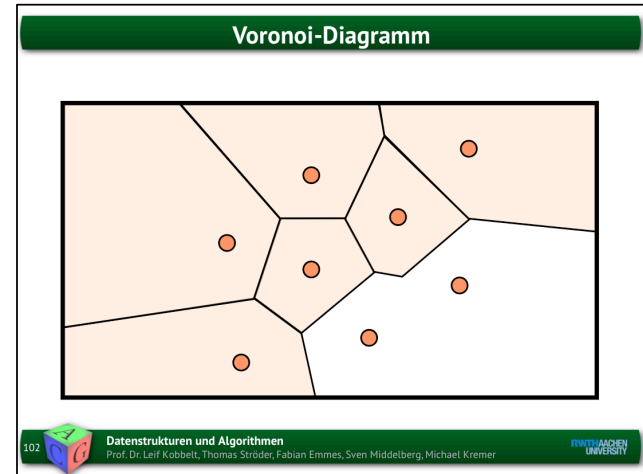
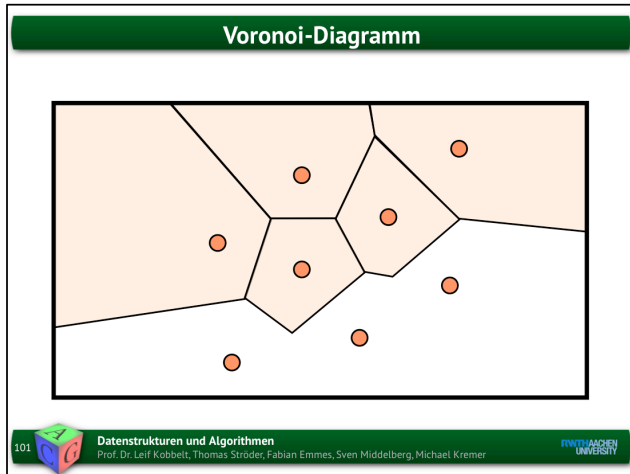
Das Voronoi-Gebiet für den roten Punkt ist dann die Schnittmenge aller Halbräume der adjazenten Punkte.  
 Da ein Halbraum immer konvex ist, ist der Schnitt zweier Halbräume auch konvex.  
 Daraus folgt, dass eine  
 Voronoi-Zelle (-Gebiet) immer konvex ist.

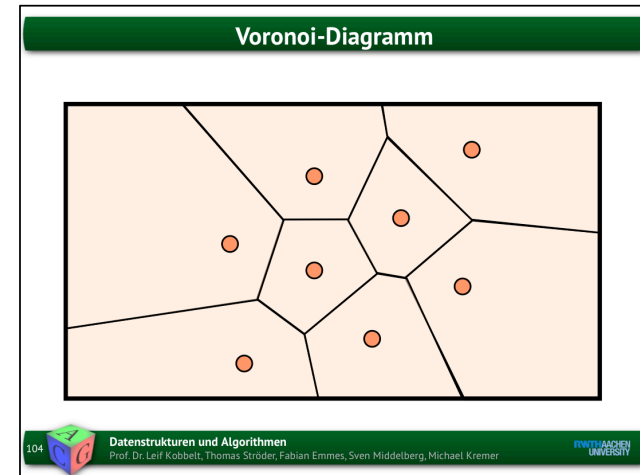
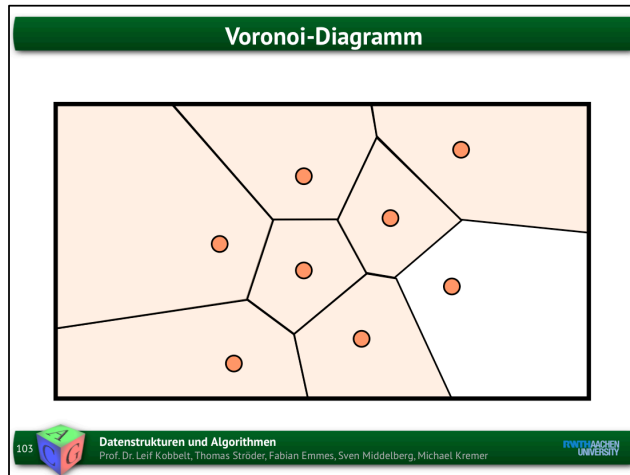
- ### Voronoi-Gebiete
- Polygonale Gebiete
  - Schnitt von Halbräumen
  - Konvexe Gebiete
- 96
Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer
FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG











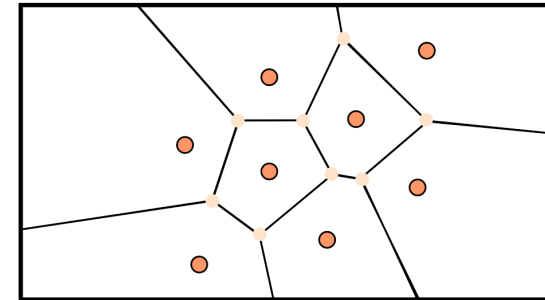
Die Menge aller Voronoi-Zellen (Voronoi-Diagramm). Nun kann man schnell bestimmen, zu welchem der orangenen Punkte ein beliebiger Punkt in dieser Ebene am nächsten ist.

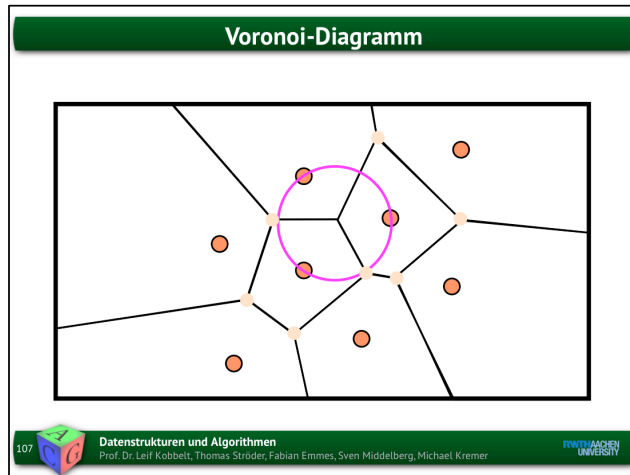
## Voronoi-Diagramm

- Ein VD ist eine (planare) Graph-Struktur, die den  $\mathbb{R}^2$  in konvexe Regionen unterteilt
- Die Voronoi-Kanten liegen auf den Mittelsenkrechten der Verbindungsstrecken
- Im allg. Treffen an einem Voronoi-Vertex genau 3 Kanten zusammen
- Voronoi-Vertices sind Umkreismittelpunkte

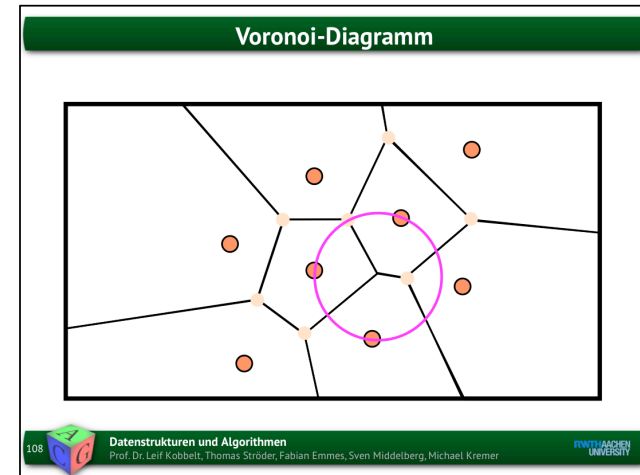


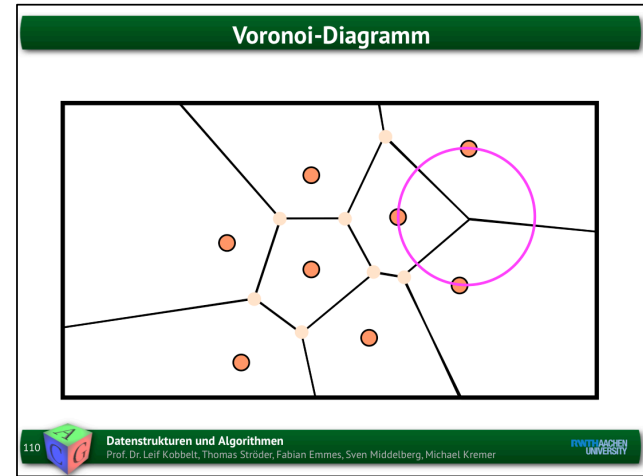
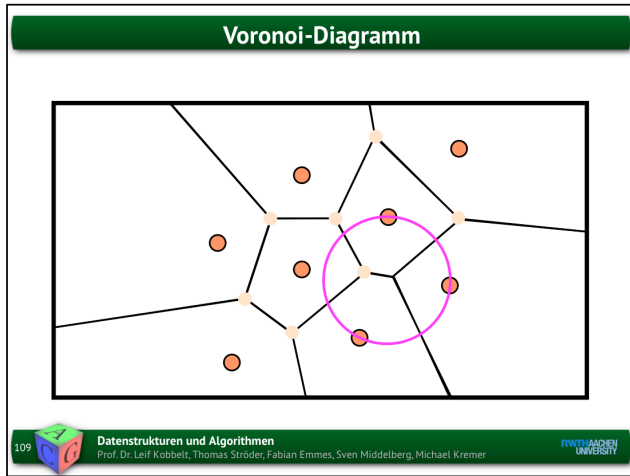
## Voronoi-Diagramm

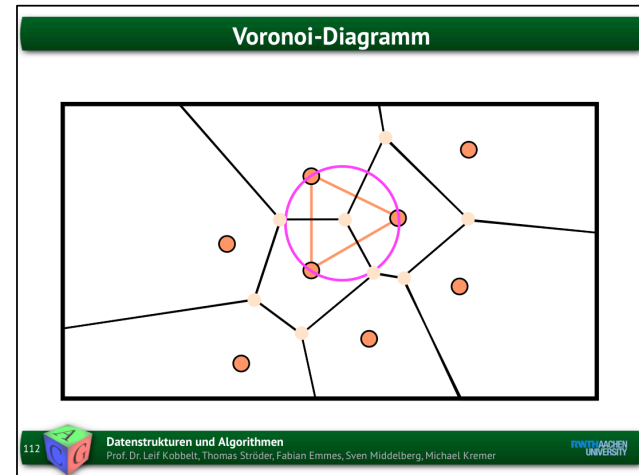
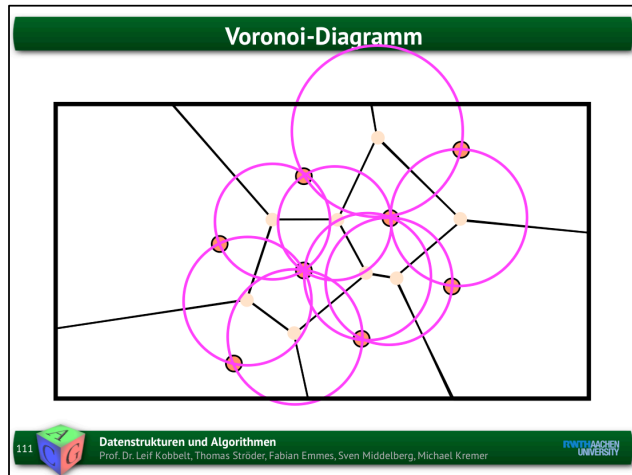




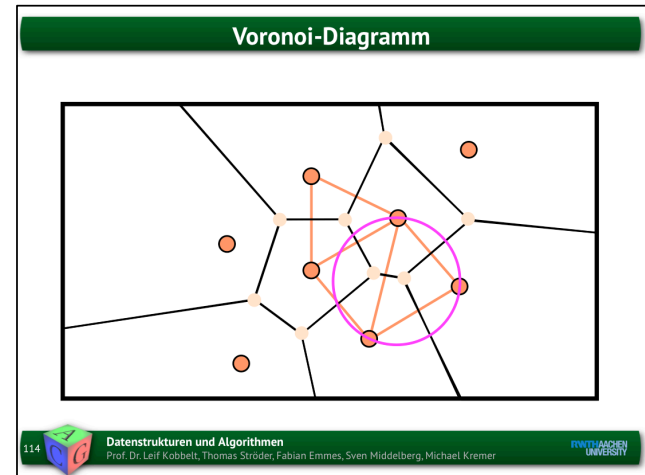
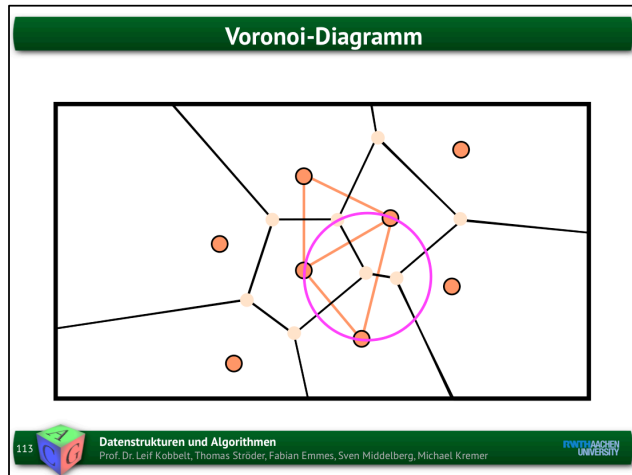
An den Eckpunkten des Diagramms schneiden sich immer genau drei Kanten. Diese Eckpunkte bilden den Mittelpunkt eines Kreises, auf dem die drei umgebenden (orangenen) Punkte liegen. Es gibt den Spezialfall, in dem ein solcher Eckpunkt Valenz vier hat. In diesem Fall liegen alle vier umgebenden Punkte auf einem Kreis mit dem Eckpunkt als Mittelpunkt.

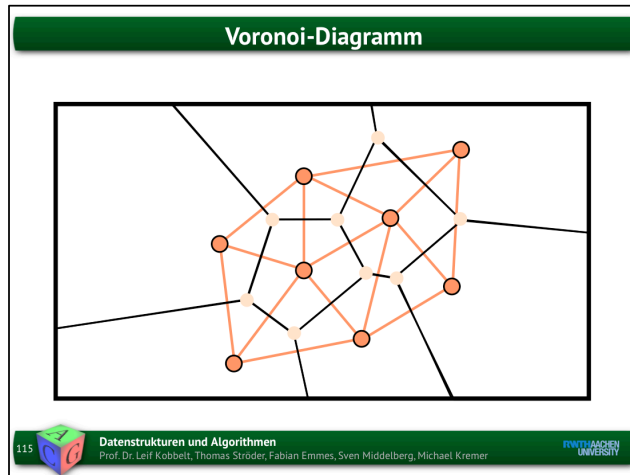




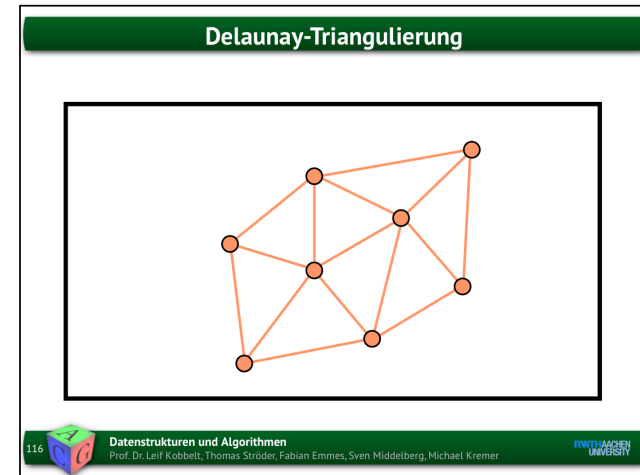








Die Delaunay-Triangulierung wird gebildet, in dem je zwei benachbarte Punkte durch eine Kante verbunden werden. Diese Triangulierung und das Voronoi-Diagramm sind dementsprechend dual zueinander.

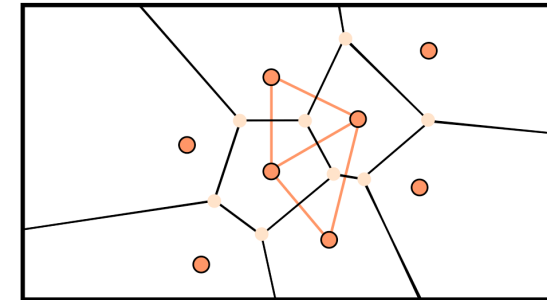


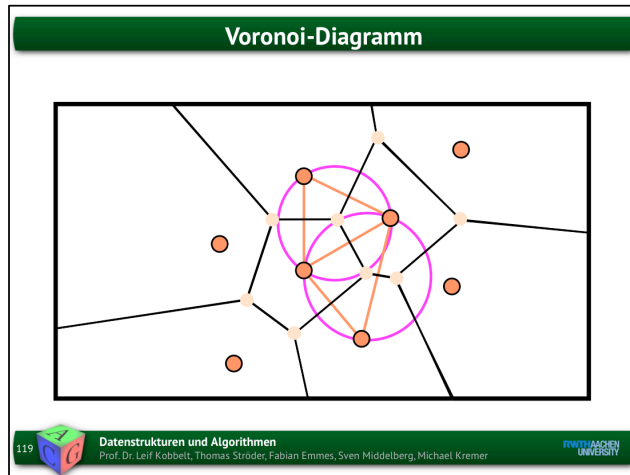
## Delaunay-Triangulierung

- Eigenschaften (1):
  - Dualer Graph zum Voronoi-Diagramm
  - Zerlegt die konvexe Hülle der Punkte in disjunkte Dreiecke (mit den  $p_i$  als Ecken)
  - Die Region innerhalb eines Dreiecks ist einem der drei Eckpunkte am nächsten
  - Der Umkreis jedes Dreiecks enthält keinen weiteren Punkt  $p_j$

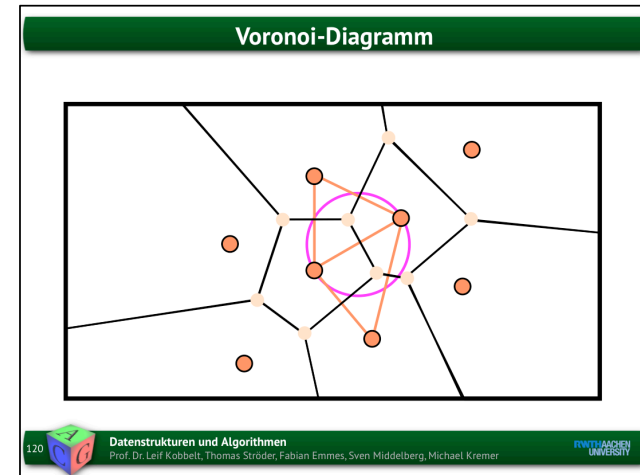


## Voronoi-Diagramm





Diese Kreise (die je drei Punkte schneiden) enthalten keinen weiteren orangenen Punkt.



## Delaunay-Triangulierung

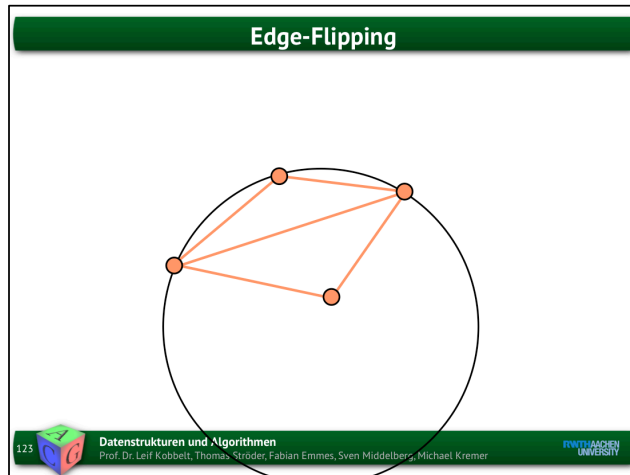
- Eigenschaften(2):
  - Der Umkreis jeder Kante enthält keinen weiteren Punkt  $p_j$
  - Die DT ist eindeutig definiert
  - Maximaler minimaler Innenwinkel
  - Einfacher zu handhaben als VD, da keine allg. Graph-Struktur verarbeitet werden muß



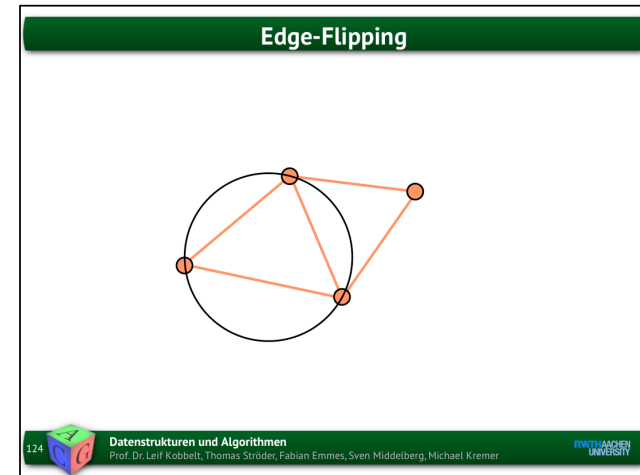
## Delaunay-Triangulierung

- Inkrementeller Algorithmus zur Erzeugung ...
  - Für nur drei Punkte  $p_1, p_2, p_3$  trivial
  - Füge weitere Punkte  $p_4, \dots$  ein und stelle die Delaunay-Eigenschaften wieder her

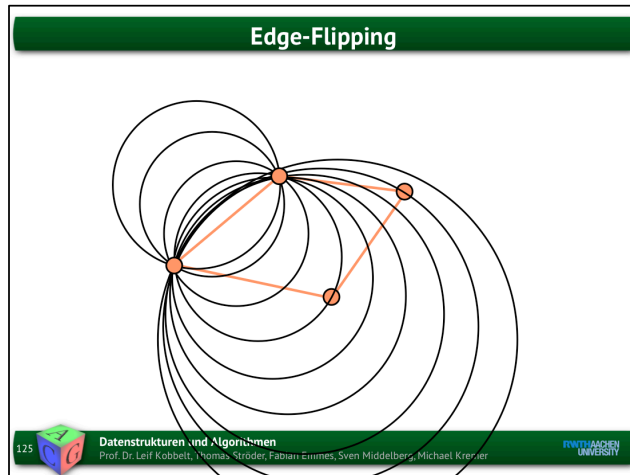




Kreis, der drei Punkte schneidet, enthält einen weiteren Punkt im Inneren. => Delaunay-Eigenschaft nicht erfüllt.  
Reparieren durch Edge-Flip.



Nun ist die Delaunay-Eigenschaft wieder erfüllt.



Kreisschar, die zwei Punkte schneidet und monoton größer wird zu beiden Seiten.

### Inkrementeller Algorithmus



- Geg: Punktmenge  $p_1, p_2, \dots, p_n$
- Zur Vermeidung von Spezialfällen:  
Ergänze drei zusätzliche Punkte  
 $q_1, q_2, q_3$ ,  
so dass alle  $p_i$  innerhalb des Dreiecks  
 $[q_1, q_2, q_3]$  liegen  $\Rightarrow$  Einfügen nur im  
Inneren
- $q_1, q_2, q_3$  können leicht entfernt werden

126 RWTH AACHEN UNIVERSITY



Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Man fügt drei Hilfspunkte außerhalb der Punktwolke ein, so dass sie ein Dreieck bilden, das alle Punkte der Punktwolke enthält. Dann wird inkrementell unterteilt...

### Inkrementeller Algorithmus

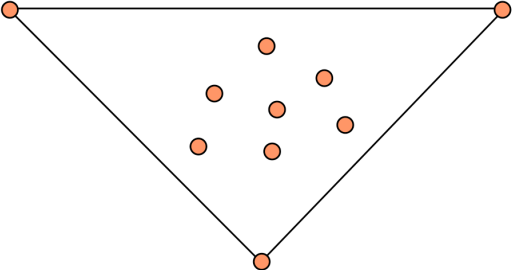
127  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

### Inkrementeller Algorithmus



128  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



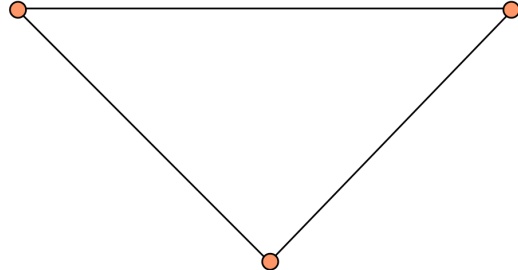
### Inkrementeller Algorithmus





The diagram shows a triangle with three vertices marked by orange circles. Inside the triangle, there are seven additional orange circles representing data points. The points are distributed in a way that suggests they are being processed or filtered by an algorithm.

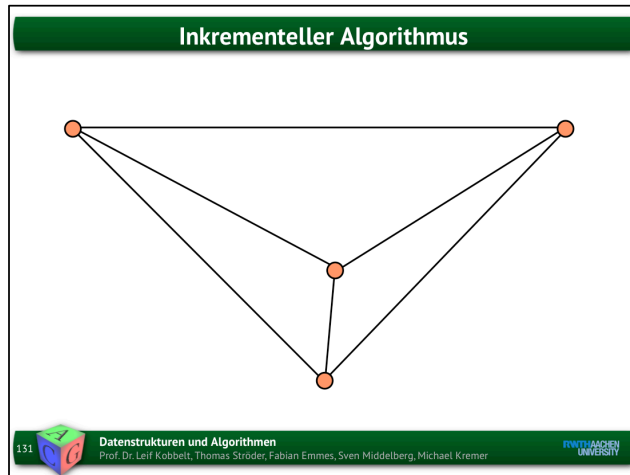
129  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  **FRONTMANNEN**  
UNIVERSITY

### Inkrementeller Algorithmus

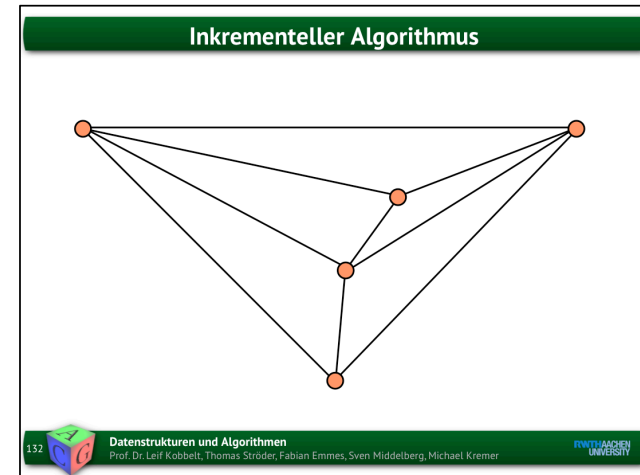


The diagram shows a triangle with three vertices marked by orange circles. The interior of the triangle is empty, indicating that the data points from the previous step have been removed or are no longer relevant to the current state of the algorithm.

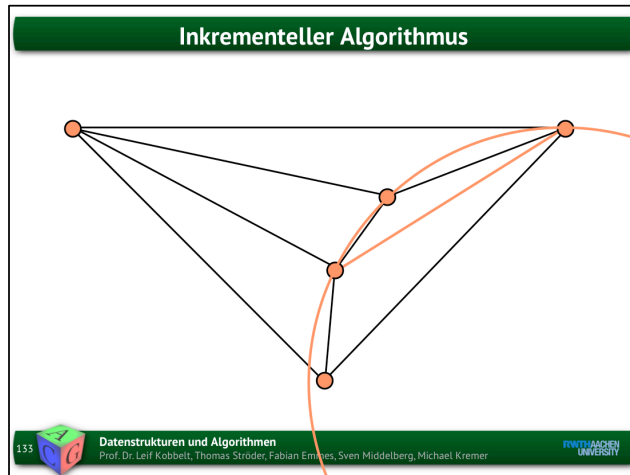
130  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  **FRONTMANNEN**  
UNIVERSITY



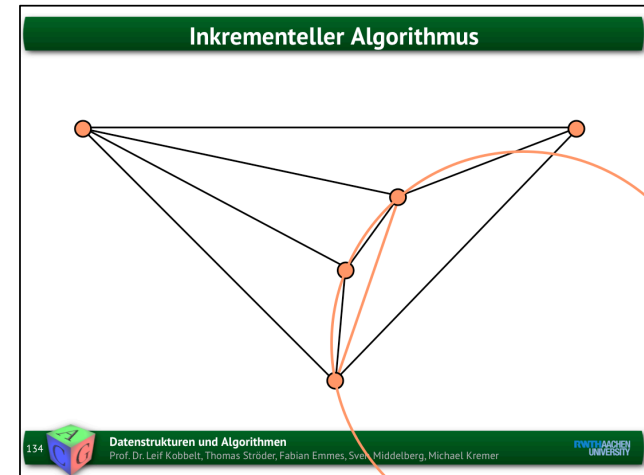
Es wird für jedes neu hinzugekommene Dreieck geprüft, ob Delaunay-Eigenschaft erfüllt ist. Hier ist sie erfüllt.



Hier nicht.





Die orangene Kante muss geflippt werden.





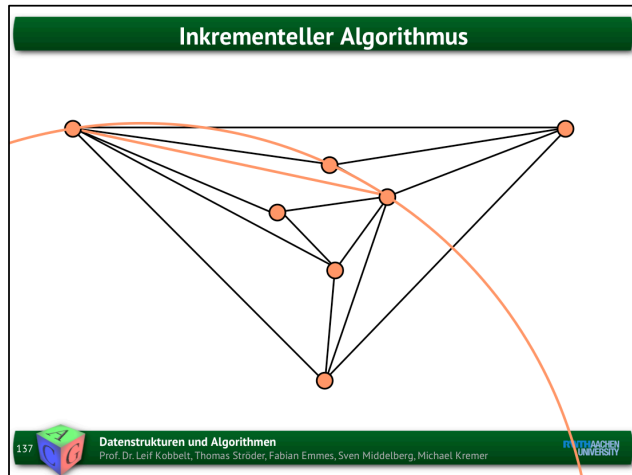
Delaunay-Eigenschaft wiederhergestellt.

**Inkrementeller Algorithmus**

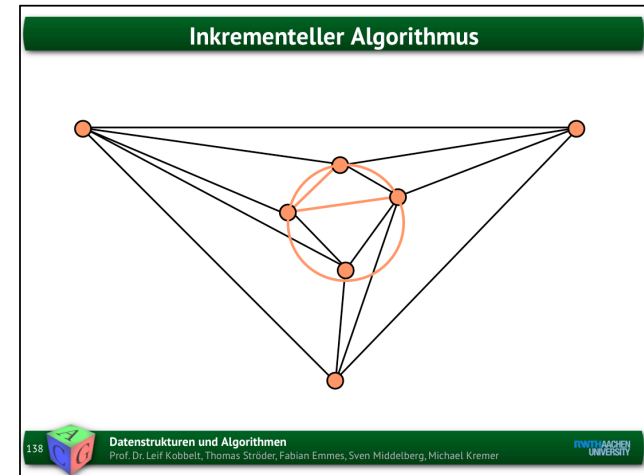
135  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

**Inkrementeller Algorithmus**

136  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 





D.-Eig. wieder verletzt. Kanten-Flip.





Und so weiter...



**Inkrementeller Algorithmus**

139  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



**Inkrementeller Algorithmus**

140  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



**Inkrementeller Algorithmus**

141  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



**Inkrementeller Algorithmus**

142  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

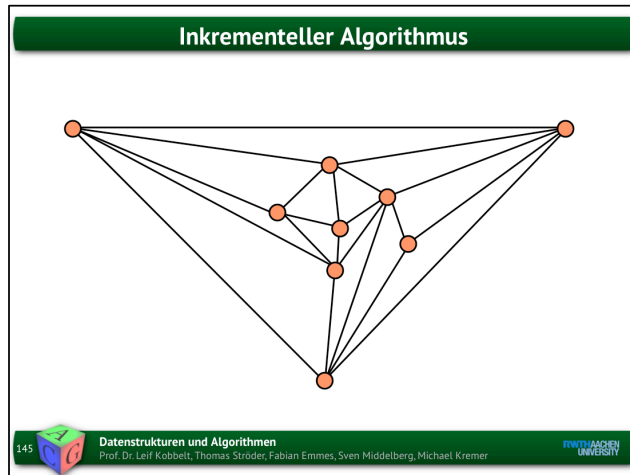
### Inkrementeller Algorithmus

143  Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  RWTH AACHEN  
UNIVERSITY

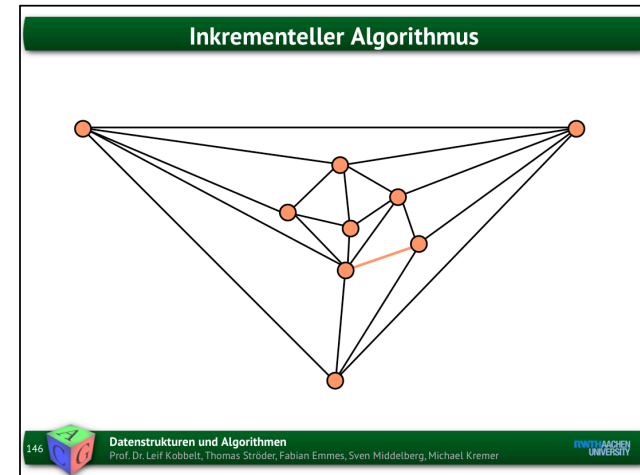
### Inkrementeller Algorithmus

144  Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer  RWTH AACHEN  
UNIVERSITY









Jeder Punkt, der hinzugefügt wird, hat die Valenz drei, da er immer im Inneren eines Dreiecks hinzugefügt wird.





### Inkrementeller Algorithmus

147  Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



### Inkrementeller Algorithmus

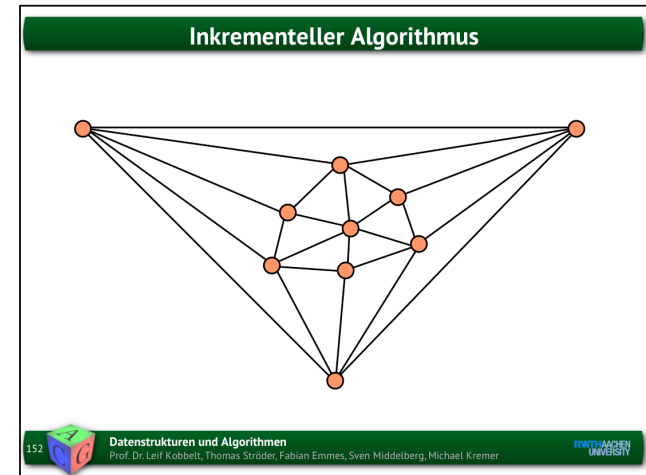
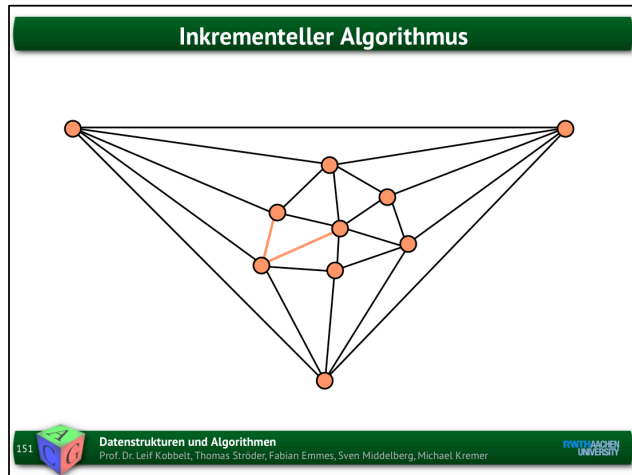
148  Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

### Inkrementeller Algorithmus

149  Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

### Inkrementeller Algorithmus

150  Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



### Inkrementeller Algorithmus

153 FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG

Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

Am Ende schmeißt man die drei Hilfspunkte inkl. inzidenter Kanten wieder weg und ist fertig mit der Delaunay-Triangulierung.  
 Man könnte mit Hilfe dieses Algorithmus auch abschätzen, wieviele Dreiecke eine Punktwolke, die 1 Mio. Vertices enthält, beinhaltet. Für jeden Vertex, den wir neu hinzufügen, werden zwei neue Dreiecke dem Graphen hinzugefügt. Daraus ergibt sich, dass die Anzahl der Dreiecke in einem Dreiecksnetz ungefähr doppelt so groß ist, wie die Anzahl der Vertices.

### Inkrementeller Algorithmus

- Für jeden Punkt  $p_i$  ]  $O(n)$ 
  - Suche das entsprechende Dreieck ]  $O(n)$
  - Füge den neuen Punkt ein ]  $O(1)$
  - Flippe Kanten bis die Delaunay-Bedingungen wieder hergestellt sind. ]  $O(k)$

154 FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG

Datenstrukturen und Algorithmen  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

$k$  konstant und unabhängig von der Komplexität des Netzes.

## Aufwandsabschätzung

- Der maximale Knotengrad  $k$  (Valenz) kann normalerweise als beschränkt angenommen werden (zumindest als unabhängig von  $n$ )
- Gesamtaufwand  $O(n^2)$
- Beschleunigung durch bessere Suche



Leider trotzdem ein Gesamtaufwand von  $O(n^2)$ , da für jeden Punkt sein zugehöriges Dreieck gesucht werden muss.  
Dies können wir noch beschleunigen.

## Beschleunigte Suche

- Standard Ansatz:  
Verwalte die Dreiecke in einem Suchbaum, um schneller auf die Elemente zuzugreifen
- Verbesserung:  $O(\log n)$  statt  $O(n)$

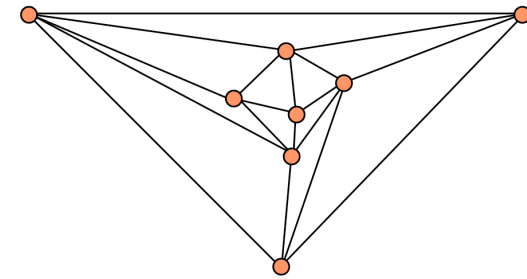


## Beschleunigte Suche



- Am Anfang existiert nur ein Dreieck
- Bei jedem Einfügeschritt werden  $m$  Dreiecke entfernt und  $m+2$  Dreiecke ergänzt





## Inkrementeller Algorithmus



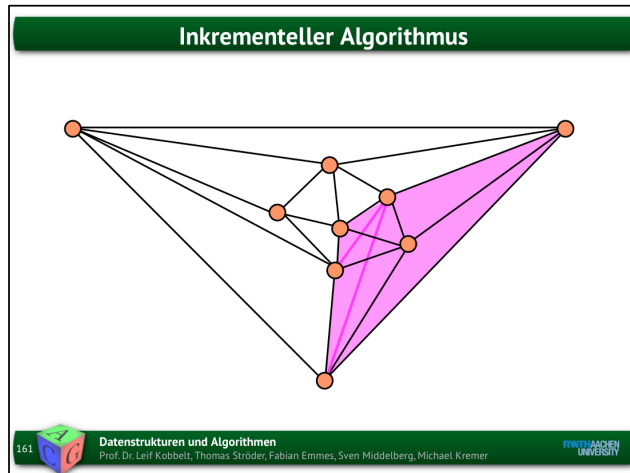
### Inkrementeller Algorithmus

159  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

### Inkrementeller Algorithmus

160  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 







- ### Beschleunigte Suche
- Am Anfang existiert nur ein Dreieck
  - Bei jedem Einfügeschritt werden  $m$  Dreiecke entfernt und  $m+2$  Dreiecke ergänzt
  - $m+2 \leq k$  ... die maximale Valenz
  - $m+2 : m \geq c > 1$  ... minimaler Verfeinerungsfaktor
- 162

**Datenstrukturen und Algorithmen**  
 Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer

$k$  = maximale Valenz im Graphen (i.d.R. beschränkt durch konstante Zahl).  
 Der unten stehende Quotient besagt, dass die Anzahl der Dreiecke im Graphen streng monoton wächst.



### Inkrementeller Algorithmus

The diagram shows a graph with 8 nodes (orange circles) and several edges. A pink shaded region highlights a specific part of the graph, consisting of a central node connected to three other nodes, which are in turn connected to a fourth node. This region is part of a larger structure that includes two additional nodes on the left and one on the right, all connected to the central node of the pink region.

163  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



### Inkrementeller Algorithmus

The diagram shows a graph with 8 nodes (orange circles) and several edges. A pink shaded region highlights a specific part of the graph, consisting of a central node connected to three other nodes, which are in turn connected to a fourth node. This region is part of a larger structure that includes two additional nodes on the left and one on the right, all connected to the central node of the pink region.

164  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



### Inkrementeller Algorithmus

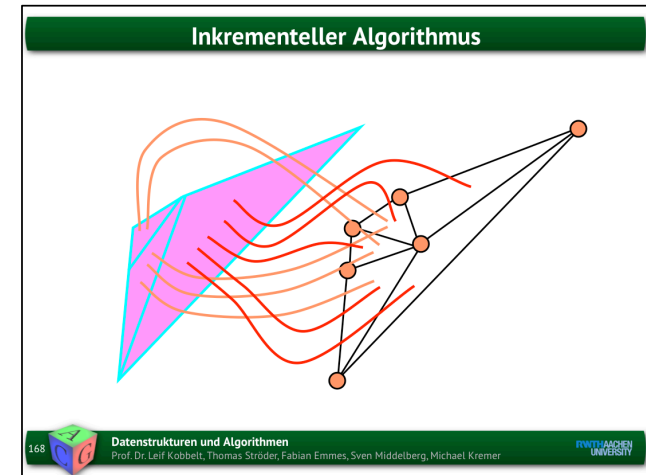
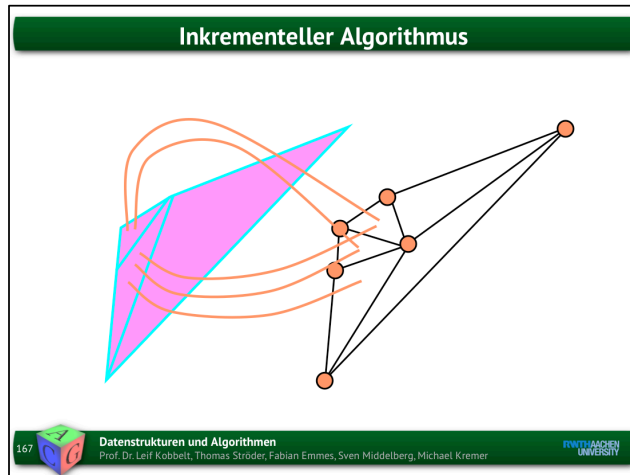
The diagram illustrates an incremental algorithm. On the left, a pink polygon is shown with a cyan outline. On the right, a graph structure is shown with orange nodes and black edges. The graph consists of a central node connected to three other nodes, which are further connected to a fourth node on the right. The pink polygon is positioned to the left of the graph, and its right edge is aligned with the leftmost edge of the graph.

165  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 

### Inkrementeller Algorithmus

The diagram illustrates an incremental algorithm. On the left, a pink polygon is shown with a cyan outline. On the right, a graph structure is shown with orange nodes and black edges. The graph consists of a central node connected to three other nodes, which are further connected to a fourth node on the right. The pink polygon is positioned to the left of the graph, and its right edge is aligned with the leftmost edge of the graph. Orange arrows indicate the flow of information or the algorithm's progress, starting from the top-left corner of the pink polygon and moving towards the graph structure.

166  **Datenstrukturen und Algorithmen**  
Prof. Dr. Leif Kobbelt, Thomas Stroder, Fabian Emmes, Sven Middelberg, Michael Kremer 



Bilde Baum mit Mapping von allen Dreiecken auf das übergeordnete Dreieck der jeweils größeren Triangulierung.  
Das große Hilfsdreieck vom Anfang bildet dabei die Wurzel.

## Suchbaum für Dreiecke

- Seien die Dreiecke, die im Laufe des Algorithmus generiert werden die Knoten des Suchbaumes
- Durch den minimalen Verfeinerungsfaktor  $m+2 : m \geq c > 1$  ist garantiert, dass die Anzahl von Knoten von Level zu Level exponentiell steigt



## Suchbaum für Dreiecke

- Obwohl diese Datenstruktur keinen (zyklenfreien) Baum darstellt, ergibt sich trotzdem ein gerichteter Graph, dessen Pfade maximal logarithmische Länge haben, wenn die Punkte hinreichend gleichmäßig verteilt sind (und die maximale Valenz beschränkt ist).
- Kosten:
  - Aufbau des Baumes  $O(n)$
  - Suche im Baum  $O(\log n)$



## Inkrementeller Algorithmus

- Für jeden Punkt  $p_i$  }  $O(n)$ 
  - Suche das entsprechende Dreieck }  $O(\log n)$
  - Füge den neuen Punkt ein }  $O(1)$
  - Flippe Kanten bis die Delaunay-Bedingungen wieder hergestellt sind. }  $O(k)$
- Gesamtaufwand:  $O(n \log n)$

