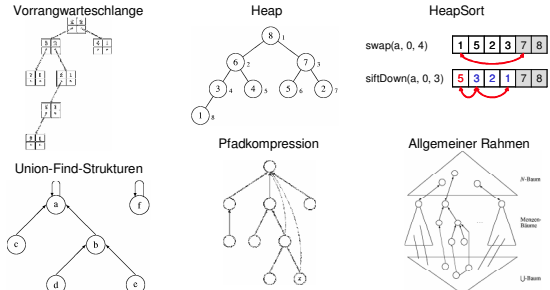


# Geometrische Algorithmen

Thomas Röfer

- Motivation
- Scan-line-Prinzip
- Konvexe Hülle
- Distanzprobleme
- Voronoi-Diagramm

# Rückblick „Manipulation von Mengen“



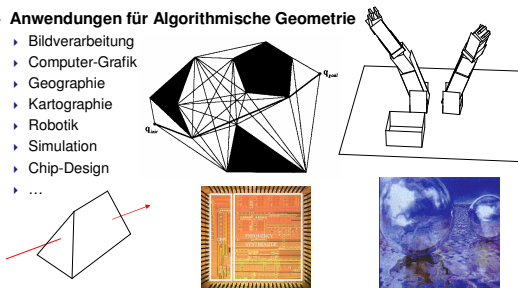
PI-2: Geometrische Algorithmen

2

# Motivation

## Anwendungen für Algorithmische Geometrie

- Bildverarbeitung
- Computer-Grafik
- Geographie
- Kartographie
- Robotik
- Simulation
- Chip-Design
- ...



PI-2: Geometrische Algorithmen

3

# Scan-Line-Prinzip

## Motivation

- Probleme im zweidimensionalen Raum wie „welche Strecke schneidet welche andere Strecke“ oder „welche Strecke sieht welche andere Strecke“ lassen sich naiv nur mit einem Aufwand von  $O(N^2)$  oder höher lösen
- Mit dem Scan-Line-Prinzip lässt sich der Aufwand aber auf  $O(N \log N)$  drücken

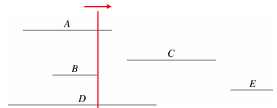
## Ansatz

- Die (End-)Punkte der Strecken werden nach der  $x$ -Koordinate sortiert
- Dann wandert die Scan-Line von Punkt zu Punkt (links nach rechts) und verwaltet einen nach  $y$ -Koordinaten sortierten Puffer der offenen Strecken

## Algorithmus

- $Q$  ist objekt- und problemabhängige Folge von Haltepunkten in aufsteigender  $x$ -Reihenfolge
- Setze  $L = \emptyset$
- Solange  $Q$  nicht leer
  - wähle nächsten Haltepunkt aus  $Q$  und entferne ihn aus  $Q$
  - $update(L)$  und gib (problemabhängige) Teilantwort aus

## Beispiel: Sichtbarkeitsproblem

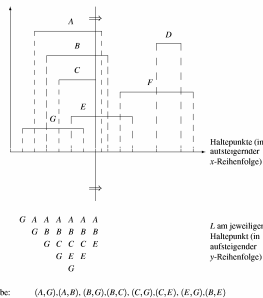


PI-2: Geometrische Algorithmen

4

# Scan-Line-Prinzip – Sichtbarkeit

- $Q$  ist Folge der  $2N$  Anfangs- und Endpunkte von Elementen in  $S$  in aufsteigender  $x$ -Reihenfolge
- $L$  ist Menge der aktiven Liniensegmente in aufsteigender  $y$ -Reihenfolge. Anfangs  $L = \emptyset$
- Solange  $Q$  nicht leer
  - $p$  ist nächster Haltepunkt aus  $Q$
  - Wenn  $p$  Startpunkt eines Segments  $s$ 
    - Füge  $s$  in  $L$  ein
    - Bestimme die Nachbarn  $s'$  und  $s''$  von  $s$  in  $L$  und gib  $(s, s')$  und  $(s, s'')$  aus
  - Ansonsten ist  $p$  Endpunkt von  $s$ 
    - Bestimme die Nachbarn  $s'$  und  $s''$  von  $s$  in  $L$  und gib  $(s', s'')$  aus
    - Entferne  $s$  aus  $L$
- Aufwand
  - Sortieren:  $O(2N \log 2N)$
  - Durchlaufen:  $2N O(\log N)$

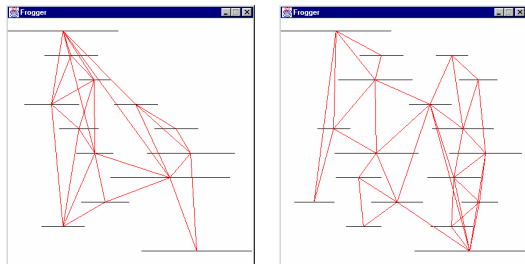


Ausgabe:  $(A, G), (A, B), (B, G), (B, C), (C, G), (C, E), (E, G), (B, E)$

PI-2: Geometrische Algorithmen

5

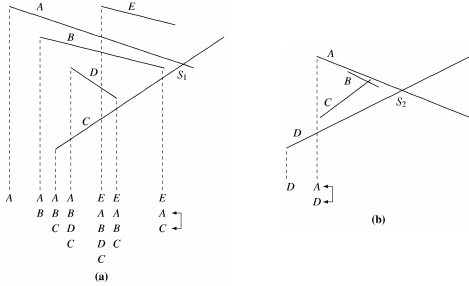
# Scan-Line-Prinzip – Sichtbarkeit



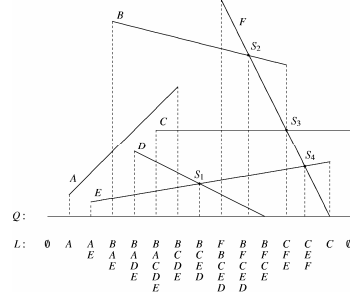
PI-2: Geometrische Algorithmen

6

### Scan-Line-Prinzip – Schnittpunkte

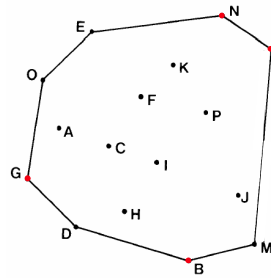


### Scan-Line-Prinzip – Schnittpunkte



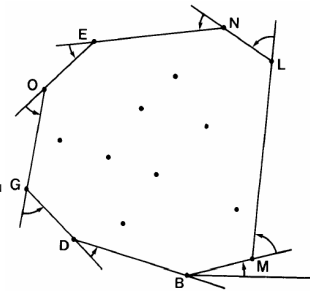
### Konvexe Hülle

- Definition
  - Die *Konvexe Hülle* zu einer Punktmenge  $P$  ist das kleinste Polygon  $G$ , das alle Punkte aus  $P$  umschließt
- Eigenschaften
  - Jede Verbindungsgerade zwischen zwei Punkten aus  $P$  liegt ebenfalls innerhalb der konvexen Hülle
  - Jede Gerade außerhalb der konvexen Hülle, die gegen die Hülle geschoben wird, wird zuerst einen der Eckpunkte berühren
- Garantierte Hüllpunkte
  - Punkte mit minimalen oder maximalen  $x$ - oder  $y$ -Koordinaten



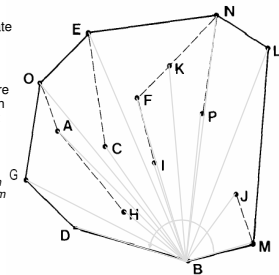
### Konvexe Hülle – Package Wrapping

- Algorithmus
  - $p_0$  ist der Punkt mit minimaler  $y$ -Koordinate
  - Setze  $p = p_0$
  - Nimm an, die vorherige Kante wäre waagrecht gewesen
  - Suche den Punkt  $p'$ , bei dem der Winkel zwischen  $(p, p')$  und der vorherigen Kante minimal ist
  - Füge  $p'$  zur konvexen Hülle hinzu
  - Setze  $p = p'$
  - Fahre mit 4. fort, solange  $p \neq p_0$
- Aufwand
  - $n$  mal nach dem Punkt mit dem kleinsten Winkel suchen:  $O(n^2)$



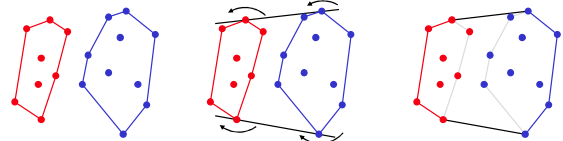
### Konvexe Hülle – Graham Scan

- Algorithmus
  - $p_0$  ist der Punkt mit minimaler  $y$ -Koordinate
  - Berechne für jeden Punkt  $p_i$  den Winkel zwischen der Waagerechten und der Strecke  $(p_0, p_i)$
  - Sortiere die Punkte danach. Falls mehrere den gleichen Winkel haben, werden sie in der Reihenfolge ihres Abstands abgelegt
  - Lege  $p_0, p_1$  und  $p_2$  auf einem Stack  $S$  ab
  - Für alle weiteren Punkte  $p_i$  in der Reihenfolge der Sortierung
    - Solange  $p_i$  auf oder rechts von der Geraden  $(nextToTop(S), top(S))$  liegt, wird  $top(S)$  vom Stack  $S$  entfernt
    - Danach wird  $p_i$  auf dem Stack  $S$  abgelegt
- Aufwand
  - Sortierung:  $O(N \log N)$
  - Scan:  $O(2N)$



### Konvexe Hülle – Divide and Conquer

- Algorithmus
  - Falls eine Punktmenge nur ein Element enthält, ist dieser Punkt die konvexe Hülle
  - Ansonsten zerlege die Punktmenge so in zwei Hälften, dass alle Punkte der einen Menge links von allen Punkten der zweiten Menge liegen
  - Berechne die konvexen Hüllen für beide Teilmengen
  - Bestimme die obere und untere Tangente der beiden Hüllen
  - Die Geraden zwischen den Punkten auf diesen Tangenten verbinden die beide Hüllen
  - Die dazwischen (innerhalb) liegenden Punkte der beiden Teilhüllen werden gelöscht

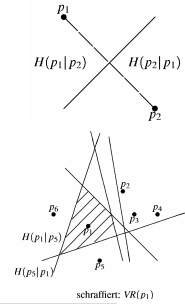


## Distanzprobleme

- Dichtestes Punktepaar**
  - Gegeben: Eine Menge  $P$  von  $N$  Punkten in der Ebene
  - Gesucht: Ein Paar  $p_1, p_2$  von Punkten aus  $P$  mit minimaler Distanz
  - Naive Lösung: Berechne die Distanz zwischen allen Punktpaaren und wähle das Minimum:  $O(n^2)$
- Alle nächsten Nachbarn**
  - Gegeben: Eine Menge  $P$  von  $N$  Punkten in der Ebene
  - Gesucht: Für jeden Punkt  $p_i \in P$  ein nächster Nachbar  $p_j \in P \setminus \{p_i\}$ , so dass  $d(p_i, p_j) = \min_{p_k \in P \setminus \{p_i\}} d(p_i, p_k)$
  - Naive Lösung: Paare jeden Punkt mit allen anderen und wähle das Paar mit der minimalen Distanz:  $O(n^2)$
- Lösung im Eindimensionalen**
  - Sortiere alle Werte und betrachte dann nur noch aufeinander folgenden Werte
  - $O(N \log N)$

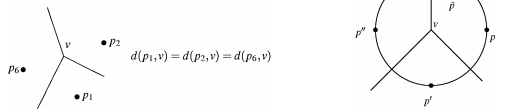
## Voronoi-Diagramm

- Definition**
  - Das Voronoi-Diagramm für eine Menge von Punkten in der Ebene teilt die Ebene in Gebiete gleicher nächster Nachbarn
  - Besteht die Menge aus lediglich zwei Punkten, so wird die Einteilung gerade durch die Mittelsenkrechte auf der Verbindungsstrecke der beiden Punkte realisiert
- Formal**
  - Der geometrische Ort aller Punkte, die näher an  $p_1$  liegen als an  $p_2$ , ist die Halbebene  $H(p_1 | p_2)$
  - Die Voronoi-Region  $VR(p)$  enthält die geometrischen Orte aller Punkte der Ebene, die näher an  $p \in P$  liegen als an jedem anderen Punkt aus  $P$
  - $VR(p) = \bigcap_{p' \in P \setminus \{p\}} H(p | p')$



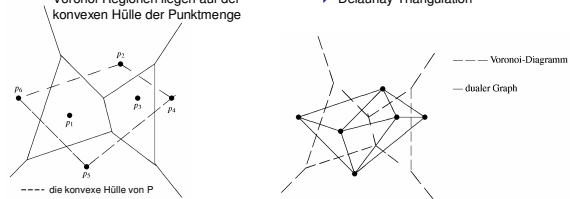
## Voronoi-Diagramm – Eigenschaften

- Definition**
  - Ein Voronoi-Diagramm ist ein Graph, dessen Knoten *Voronoi-Knoten* und dessen Kanten *Voronoi-Kanten* genannt werden
- Eigenschaften**
  - Jeder Knoten  $v$  ist gleich weit von den Punkten  $p_i$  entfernt, der Regionen  $VR(p_i)$  an  $v$  angrenzen
  - Unter der Annahme, dass keine vier Punkte in  $P$  auf einem Kreis liegen, liegen um jeden Knoten  $v$  drei Punkte in einem Abstand  $r$ . Innerhalb des Radius  $r$  kann es keinen weiteren Punkt  $\tilde{p}$  geben

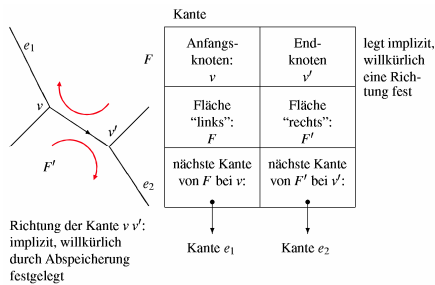


## Voronoi-Diagramm – Eigenschaften

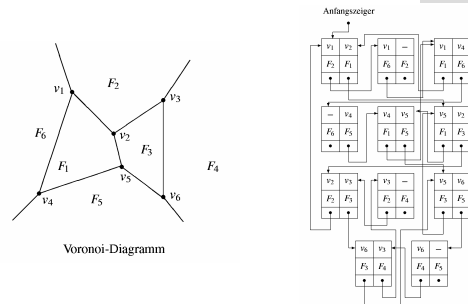
- Eigenschaften**
  - Eine Voronoi-Regionen sind beschränkt, andere unbeschränkt
  - Die Punkte der unbeschränkten Voronoi-Regionen liegen auf der konvexen Hülle der Punktmenge
- Dualer Graph**
  - Ein Graph, in dem alle Punkte angrenzender Voronoi-Regionen verbunden sind
  - Delaunay-Triangulation



## Voronoi-Diagramm – Datenstruktur



## Voronoi-Diagramm – Datenstruktur



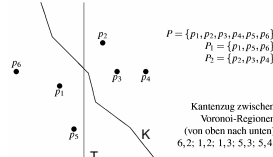
## Voronoi-Diagramm – Datenstruktur

```
class Edge
{
    Node startNode,
    endNode;
    Surface leftSurface,
    rightSurface;
    Edge startEdge,
    endEdge;
}
```

```
void surround(Surface s, Edge e
             EdgeAction a)
{
    Edge e2 = e;
    do
    {
        a.action(e2);
        if (e2.leftSurface == s)
            e2 = e2.startEdge;
        else
            e2 = e2.endEdge;
    }
    while (e2 != e);
}
```

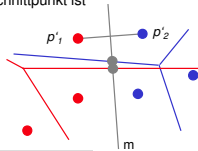
## Voronoi-Diagramm – Konstruktion

- ▶ **Gegeben**
  - ▶ Eine Menge  $P$  von  $N$  Punkten in der Ebene
- ▶ **Gesucht**
  - ▶ Das Voronoi-Diagramm  $VD(P)$  für  $P$ , als doppelt verkettete Kantenliste
- ▶ **Falls  $|P| = 1$** 
  - ▶  $VD(P)$  ist die gesamte Ebene
- ▶ **Ansonsten**
  - ▶ **Divide**
    - ▶ Teile  $P$  durch eine vertikale Trennlinie  $T$  in zwei etwa gleich große Teilmengen  $P_1$  (links von  $T$ ) und  $P_2$  (rechts von  $T$ )
  - ▶ **Conquer**
    - ▶ Berechne  $VD(P_1)$  und  $VD(P_2)$  rekursiv
  - ▶ **Merge**
    - ▶ Berechne den  $P_1$  und  $P_2$  trennenden Kantenzug  $K$ , der Teil von  $VD(P)$  ist
    - ▶ Schneide den rechts von  $K$  liegenden Teil von  $VD(P_1)$  ab, sowie den links von  $K$  liegenden Teil von  $VD(P_2)$
    - ▶ Vereine  $VD(P_1)$ ,  $VD(P_2)$  und  $K$  zu  $VD(P)$

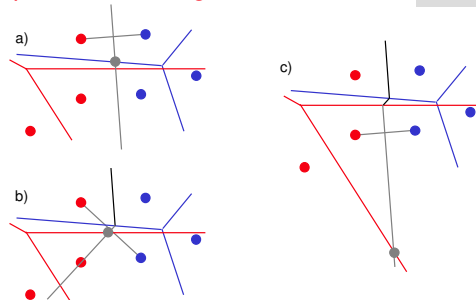


## Voronoi-Diagramm – Kantenzug $K$

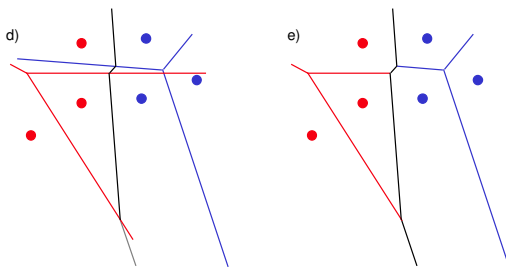
1. Setze  $K = \emptyset$
2. Ermittle die beiden oberen Tangentialpunkte  $p'_1 \in P_1$  und  $p'_2 \in P_2$
3. Bestimme die Mittelsenkrechte  $m$  der Verbindungsstrecke von  $p'_1$  und  $p'_2$
4. Setze  $k = (x, \infty)$ , so dass  $k$  auf  $m$  liegt
5. Schneide  $m$  mit den beiden unteren unendlichen Kanten in  $VR(p'_1)$  und  $VR(p'_2)$
6. Wenn es keinen Schnitt gibt, füge  $(k, (x, \infty))$  als unendliche Kante in  $K$  ein, wobei  $(x, \infty)$  auf  $m$  liegt. Gib  $K$  zurück
7. Ansonsten füge  $(k, s)$  in  $K$  ein, wobei  $s$  der obere Schnittpunkt ist
8. Setze  $k = s$
9. Der zum oberen Schnitt gehörige Punkt  $p'_i$  ( $i = 1, 2$ ) wird durch seinen an dieselbe Kante angrenzenden Nachbarn  $p''_i$  ersetzt
10. Bestimme die neue Mittelsenkrechte  $m$  der Verbindungsstrecke von  $p'_i$  und  $p'_2$
11. Fahre mit 5. fort



## Beispiel – Kantenzug $K$



## Beispiel – Kantenzug $K$



## Voronoi-Diagramm – Aufwand

- ▶ **Aufwand**
  - ▶ Sortieren der Punkte nach x-Koordinaten:  $O(N \log N)$
  - ▶ Aufteilen der sortierten Punkte auf zwei Hälften:  $O(1)$
  - ▶ Bestimmung von  $K$ :  $O(N)$
  - ▶ Zusammenführen von  $VD(P_1)$ ,  $VD(P_2)$  und  $K$ :  $O(N)$
  - ▶ Rekursion:  $2 \cdot O(N/2)$ ,  $\log N$  mal  $\rightarrow O(N \log N)$
- ▶ **Dichtestes Punkte Paar**
  - ▶ Konstruiere das Voronoi-Diagramm  $VD(P)$  der Punktmenge  $P$ :  $O(N \log N)$
  - ▶ Durchlaufe alle Kanten und ermittle das Minimum der Distanz benachbarter Punkte sowie das Punktepaar, das diese Distanz hat:  $O(N)$
- ▶ **Alle nächsten Nachbarn**
  - ▶ Konstruiere das Voronoi-Diagramm  $VD(P)$  der Punktmenge  $P$ :  $O(N \log N)$
  - ▶ Durchlaufe die Kantenliste für  $VD(P)$  so, dass der Reihe nach für jeden Punkt  $p$  alle Voronoi-Kanten von  $VR(p)$  betrachtet werden. Dabei wird für jeden Punkt ein nächster Nachbar unter allen Punkten mit benachbarter Voronoi-Region ermittelt