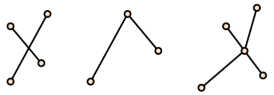
2.7 Geometrische Algorithmen

2.7 Geometrische Algorithmen

Wir wollen folgendes Problem lösen: Gegeben sei eine Menge von Liniensegmenten in der Ebene.
Wir wollen nun alle möglichen Schnitte aller Segmente bestimmen.

Schnitt von Liniensegmenten

- Problemstellung
  - Gegeben: Eine Menge S von n abgeschlossenen Liniensegmenten in der Ebene
  - Gesucht: Die Schnittpunkte der Segmente

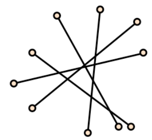


abgeschlossene Segmente → die Endpunkte gehören dazu!

Komplexität

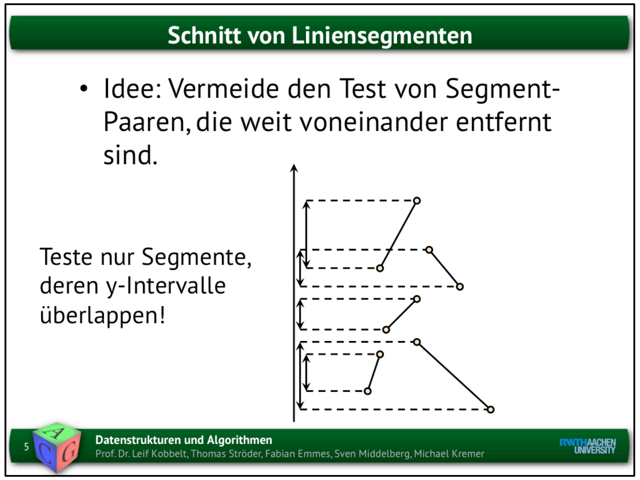
- Brute-Force
  Teste alle Paare von Segmenten → $O(n^2)$
- Dies ist im Worst-case zwar optimal …

n^2 Schnittpunkte!

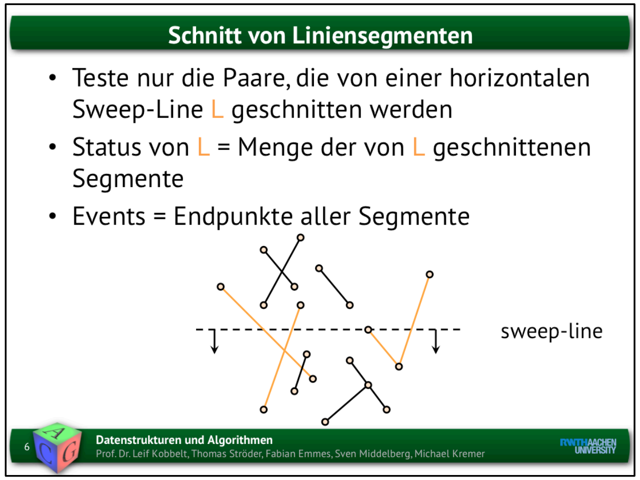


- … aber wir hätten lieber einen "output-sensitiven" Algorithmus

Aufwand von n^2, wenn jede Kante jede andere schneidet. Das muss aber nicht immer so sein.
Welche geometrischen Eigenschaften kann man ausnutzen, um dies zu beschleunigen?



Sortiere alle Kanten nach y-Bereich. Man kann nun relativ schnell ausschließen, welche Kanten sich überhaupt schneiden könnten.



„Sweep-Line-Algorithmus“

Sweep-Line „läuft“ über den Raum, der die Segmente enthält.

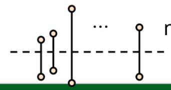
Aktiviere Kante eines Segments, wenn die Sweep-Line geschnitten wird.
Sobald die Sweep-Line nicht mehr geschnitten wird, werden diese Kanten wieder deaktiviert.
Man muss dann nur Schnitte zwischen aktivierten Kanten berechnet werden.

Schnitt von Liniensegmenten

- Ist das Event der obere Endpunkt eines Segments, so wird das Segment gegen alle Segmente im Status getestet und dann dem Status hinzugefügt.
- Ist das Event der untere Endpunkt eines Segments, so wird es aus dem Status entfernt.

➡Es werden nur Paare getestet, die von einer horizontalen Linie geschnitten werden.

- Leider ist dieser Algorithmus immer noch nicht output-sensitiv.

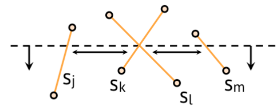


n Segmente

Schnitt von Liniensegmenten

- Verfeinerung:
  - Ordne die Segmente im Status von links nach rechts
  - Status ändert sich nun auch an Schnittpunkten → Schnittpunkte sind auch Events
  - Teste bei Status-Änderung (Einfügen, Ausfügen oder Swap) nur noch gegen die Nachbarsegmente

neue Nachbarn



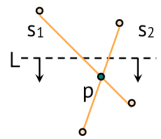
Erlaube als Event auch den Schnitt zweier Kanten.
Nur benachbarte Kanten können sich als nächstes schneiden.
Bei einem Schnitt werden die Nachbarschaftsbeziehungen zwischen den betroffenen Segmenten getauscht.

Schnitt von Liniensegmenten

- Verfeinerung:
  - Ordne die Segmente im Status von links nach rechts
  - Status ändert sich nun auch an Schnittpunkten → Schnittpunkte sind auch Events
  - Teste bei Status-Änderung (Einfügen oder Swap) nur noch gegen die Nachbarsegmente
    Finden wir immer noch alle Schnittpunkte?

Schnitt von Liniensegmenten

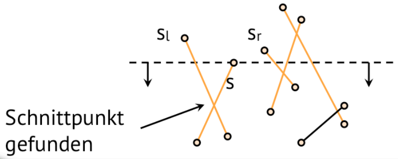
- Lemma
  Es seien s_1 und s_2 zwei nicht-horizontale Segmente, die sich in einem inneren Punkt p schneiden und es gebe kein weiteres Segment durch diesen Punkt. Dann gibt es ein Event über p, wo s_1 und s_2 Nachbarn werden und auf Schnitt getestet werden.



Es sei L eine horizontale Linie über p, so dass kein Event zwischen L und p liegt → s_1 und s_2 sind benachbart. Zu Beginn des Algorithmus waren sie aber nicht benachbart, also gibt es ein Event q bei dem s_1 und s_2 Nachbarn wurden und auf Schnitt getestet wurden.

Ablauf (grob)

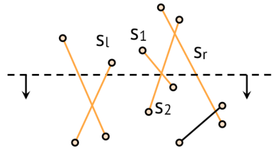
- Eine Horizontale Sweep Line L wird von oben nach unten über die Ebene bewegt, Halt an Events
- Event: Oberer Endpunkt eines Segments s
  → Das Segment s muss gegen seine zwei Nachbarn s_l und s_r getestet werden. Nur Schnittpunkte unterhalb von L sind von Interesse.



Drei Arten von Events: „Anfang“, ...

Ablauf (grob)

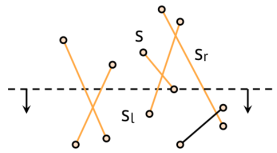
- Eine Horizontale Sweep Line L wird von oben nach unten über die Ebene bewegt, Halt an Events
- Event: Schnittpunkt zwischen Segmenten s_1 und s_2
  → Die Reihenfolge von s_1 und s_2 wird getauscht und die Segmente jeweils gegen (höchstens) einen neuen Nachbarn getestet.



... „Schnitt“ und...

Ablauf (grob)

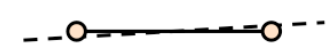
- Eine Horizontale Sweep Line L wird von oben nach unten über die Ebene bewegt, Halt an Events
- Event: Unterer Endpunkt eines Segments s
  ➞ Die Nachbarn von s werden Nachbarn und müssen auf Schnitt getestet werden.



... „Ende“.

Details

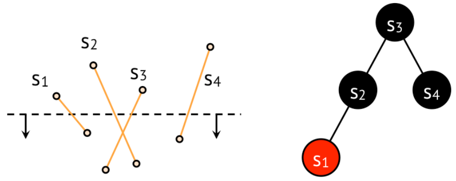
- Event-Schlange Q als balancierter binärer Suchbaum
  - Ordnung:
    $p < q \Leftrightarrow p_y < q_y$ oder ($p_y = q_y$ und $p_x < q_x$)
    Insbesondere wird der linke Endpunkt eines horizontalen Segments zuerst abgearbeitet
  - Operationen
    - Liefere nächstes Event
    - Prüfe, ob Event enthalten ist
    - ...



Events, die weiter oben liegen, werden früher bearbeitet. Danach gilt Sortierung von links nach rechts.

Datenstrukturen

- Status T als balancierter binärer Suchbaum
  - Segmente sind entlang der Sweep-Line L geordnet
  - Operationen: Finde Segment links/auf/rechts von einem Punkt q

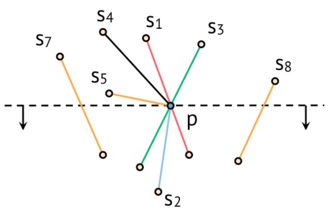


Man kann zum Beispiel einen Rot-Schwarz-Baum als Datenstruktur für die Events nehmen.

Algorithmus

- FindIntersections(S)
  Q ← empty
  insert segment endpoints into Q
  T ← empty
  while Q is not empty do
    determine next event point p in Q
    delete p from Q
    HandleEventPoint(p)

Algorithmus



U(p) = segments with upper endpoint p = $\{s_2\}$

L(p) = segments in T with lower endpoint p = $\{s_4,s_5\}$

C(p) = segments in T that contain p in their interior = $\{s_1,s_3\}$

Algorithmus

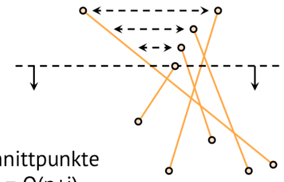
- HandleEventPoint(p)
  - determine U(p), L(p), C(p)
  - if #L(p)+#U(p)+#C(p) > 1 then
    - report p as intersection
  - delete L(p), C(p) from T, insert U(p), C(p) into T
  - if U(p) ∪ C(p) is empty then
    - s_l, s_r ← left, right neighbors of p in T
    - FindNewEvent(s_l,s_r,p)
  - else
    - s' ← leftmost segment of U(p) ∪ C(p)
    - s_l ← left neighbor of s' in T
    - FindNewEvent(s_l, s', p)
    - s" ← rightmost segment of U(p) ∪ C(p)
    - s_r ← right neighbor of s" in T
    - FindNewEvents(s", s_r, p)

Algorithmus

- FindNewEvent(s_l,s_r,p)
  if s_l and s_r intersect below L or
  on L and to the right of p then
  r ← $s_l \cap s_r$
  if r ∉ Q then
  insert(r, Q)

Erweiterung

- Um Speicherplatz zu sparen, wird der Schnittpunkt zweier Segmente s_1 und s_2 aus Q entfernt, sobald s_1 und s_2 nicht mehr benachbart sind → Speicher(Q)=O(n)



n Segmente, i Schnittpunkte
→ $\text{Speicher}_{\text{naiv}}(Q) = O(n+i)$

Aufwand

- Ohne Beweis: Der Algorithmus FindIntersections
  - ist korrekt und
  - findet die i Schnittpunkte von n Segmenten in $O((n+i) \log n)$ Schritten mit einem Speicheraufwand von $O(n)$.
- Anschaulich
  - Wir haben $O(n+i)$ Events
  - Status enthält max. n Segmente → $O(\log n)$
  - Queue enthält max. $n+i$ Segmente → $O(\log(n+i)) = O(\log(n+n^2)) = O(\log n)$

2.7 Geometrische Algorithmen

Ende des Vorlesungsstoffs.

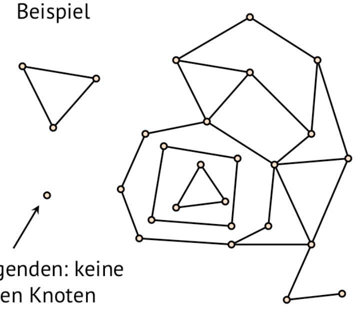
Planare Unterteilungen

- Eine planare Unterteilung S der Ebene besteht aus
  - Knoten $V = \{v_1,\ldots,v_n\} \subseteq R^2$
  - Kanten $E \subseteq V \times V$

  so dass sich das Innere der Kanten nicht schneidet.
- Maximal zusammenhängende Gebiete heißen Facetten. Eine Facette ist also ein durch Kanten und Knoten begrenztes Polygon.
- Komplexität(S) = Zahl der Knoten + Zahl der Kanten + Zahl der Facetten

Planare Unterteilungen

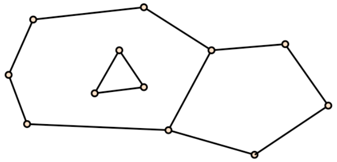
- Beispiel



im Folgenden: keine isolierten Knoten

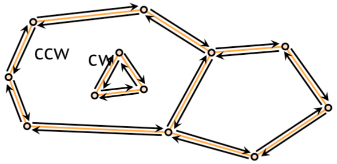
DCEL

- Doubly-connected edge list
  - Ersetze jede Kante durch zwei Halbkanten.
  - Konvention: Die Halbkanten seien so orientiert, dass die zugehörige Facette links von der Halbkante liegt.



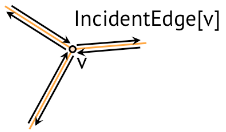
DCEL

- Doubly-connected edge list
  - Ersetze jede Kante durch zwei Halbkanten.
  - Konvention: Die Halbkanten seien so orientiert, dass die zugehörige Facette links von der Halbkante liegt.



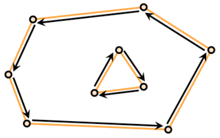
DCEL

- Zu jedem Knoten v speichern wir
  - Coordinates[v]
    Die Koordinaten von v
  - IncidentEdge[v]
    Eine bel. Halbkante mit v als Ursprung



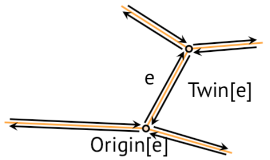
DCEL

- Zu jeder Facette f speichern wir
  - OuterComponent[f]
    Eine Halbkante des Rands von f (=NIL für die äußerste Facette)
  - InnerComponents[f]
    Eine Liste, die eine Halbkante von jedem Loch in f enthält.
  - Label[f]
    Bel. Beschriftung von f



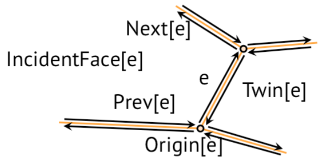
DCEL

- Zu jeder Halbkante e speichern wir
  - Origin[e]
    Den Ursprungsknoten
  - Twin[e]
    Die Nachbarhalbkante



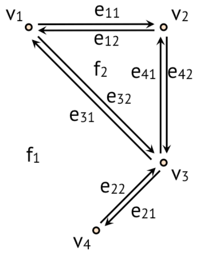
DCEL

- Zu jeder Halbkante e speichern wir
  - IncidentFace[e]
    Die angrenzende Facette
  - Prev[e], Next[e]
    Vorgänger- und Nachfolgehalbkante



DCEL

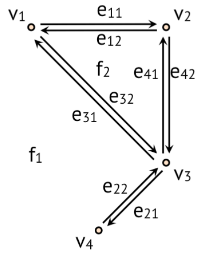
- Beispiel



Knoten	Coordinates	IncidentEdge
v_1	(0,4)	e_{11}
v_2	(2,4)	e_{42}
v_3	(2,2)	e_{21}
v_4	(1,1)	e_{22}

DCEL

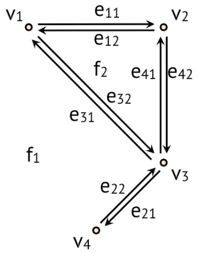
- Beispiel



Facette	OuterComponent	InnerComponents
f_1	NIL	e_{11}
f_2	e_{41}	NIL

DCEL

- Beispiel



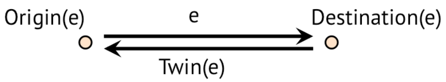
Halfedge	Origin	Twin	Incident Face	Next	Prev
e_{11}	v_1	e_{12}	f_1	e_{42}	e_{31}
e_{12}	v_2	e_{11}	f_2	e_{32}	e_{41}
e_{21}	v_3	e_{22}	f_1	e_{22}	e_{42}
e_{22}	v_4	e_{21}	f_1	e_{31}	e_{21}
e_{31}	v_3	e_{32}	f_1	e_{11}	e_{22}
e_{32}	v_1	e_{31}	f_2	e_{41}	e_{12}
e_{41}	v_3	e_{42}	f_2	e_{12}	e_{32}
e_{42}	v_2	e_{41}	f_1	e_{21}	e_{11}

DCEL

- Konstanter Speicheraufwand für Knoten und Kanten
- Speicheraufwand für Facetten nicht konstant, da abhängig von der Zahl der Löcher, aber ≤ der Zahl der Kanten
- Speicheraufwand: O(Komplexität(S))

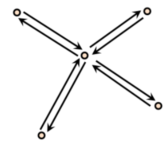
DCEL: Operationen

- Destination(e) = Origin(Twin(e))



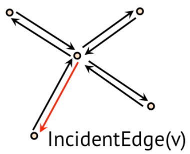
DCEL: Operationen

- EnumerateNeighbors(v)
  e ← IncidentEdge(v)
  repeat
    // do something with Destination(e)
    e ← Next(Twin(e))
  until e = IncidentEdge(v)



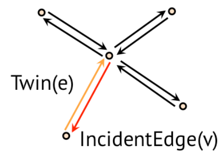
DCEL: Operationen

- EnumerateNeighbors(v)
  e ← IncidentEdge(v)
  repeat
  // do something with Destination(e)
  e ← Next(Twin(e))
  until e = IncidentEdge(v)



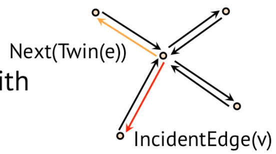
DCEL: Operationen

- EnumerateNeighbors(v)
  e ← IncidentEdge(v)
  repeat
  // do something with Destination(e)
  e ← Next(Twin(e))
  until e = IncidentEdge(v)



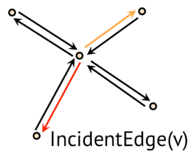
DCEL: Operationen

- EnumerateNeighbors(v)
  e ← IncidentEdge(v)
  repeat
    // do something with Destination(e)
    e ← Next(Twin(e))
  until e = IncidentEdge(v)



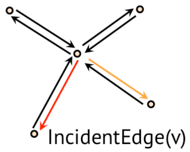
DCEL: Operationen

- EnumerateNeighbors(v)
  e ← IncidentEdge(v)
  repeat
    // do something with Destination(e)
    e ← Next(Twin(e))
  until e = IncidentEdge(v)



DCEL: Operationen

- EnumerateNeighbors(v)
  e ← IncidentEdge(v)
  repeat
  // do something with Destination(e)
  e ← Next(Twin(e))
  until e = IncidentEdge(v)



DCEL: Operationen

- EnumerateEdges(f)
  holes ← OuterComponent(f) ∪ InnerComponents(f)
  for all e_0 in holes do
  e ← e_0
  repeat
  // do something with e
  e ← Next(e)
  until e = e_0

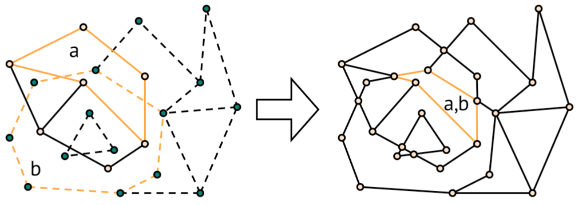
2.7 Geometrische Algorithmen

Überlagerungen

- Problem
  - Gegeben seien zwei Unterteilungen S_1, S_2 der Ebene als DCELs
  - Gesucht ist die Überlagerung $O(S_1,S_2)$ von S_1 und S_2 als DCEL.
  - Jede Facette f aus $O(S_1,S_2)$ soll mit den Labels der Facetten aus S_1 und S_2 beschriftet werden, die f enthalten

Überlagerungen

- Beispiel



Überlagerungen

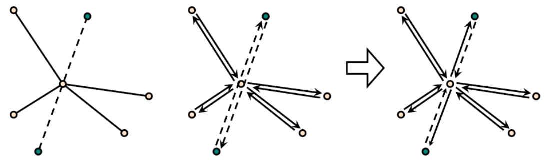
- Ablauf
  - Bestimme zunächst nur die Knoten und Halbkanten von $O(S_1,S_2)$ einschließlich korrekter Verzeigerung
  - In einem zweiten Schritt werden die Facetten bestimmt und beschriftet

Überlagerungen

- Bestimmung der Knoten und Halbkanten
  - Setze $D = S_1 \cup S_2$, D ist allerdings keine gültige Unterteilung der Ebene
  - Transformiere D mittels eines Plane-Sweep Algorithmus in eine DCEL $O(S_1,S_2)$, die die Überlagerung von S_1 und S_2 darstellt
  - Wird ein Event erreicht, so muss D entsprechend aktualisiert werden

Überlagerungen

- Beispiel: Knoten aus S_1 liegt exakt auf einer Kante von S_2



- Erzeuge neue Kanten, setze Zeiger entsprechend um

Überlagerungen

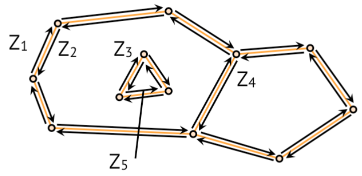
- Ähnlich
  - Knoten aus S_1 liegt auf Knoten aus S_2
  - Kante aus S_1 schneidet Kante aus S_2

Facetten

- Die Knoten und Kanten sind nun korrekt berechnet
- Bestimme und beschrifte die Facetten aus O(S1,S2)

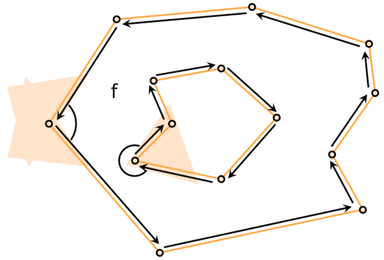
Facetten

- Bestimme zunächst alle Zyklen von Halbkanten
  - Beschreibt ein Zyklus einen äußeren Rand oder einen Lochrand?
  - Welche Zyklen begrenzen die gleiche Facette?



Facetten

- Beschreibt ein Zyklus Z einen äußeren Rand oder einen Lochrand?

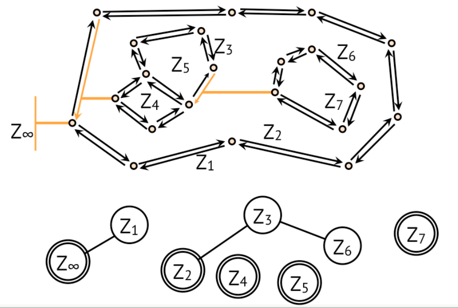


Facetten

- Beschreibt ein Zyklus Z einen äußeren Rand oder einen Lochrand?
  - Die (unbekannte) angrenzende Facette liegt links von den Halbkanten.
  - Betrachte den linkesten Knoten im Zyklus und berechne den Winkel α innerhalb der angrenzenden Facette.
  - Ist α kleiner 180 Grad, so handelt es sich um einen äußeren Rand.
  - Ist α größer als 180 Grad, so handelt es sich um einen Lochrand.

Facetten

- Welche Zyklen begrenzen die gleiche Facette?



Facetten

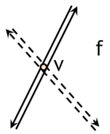
- Welche Zyklen begrenzen die gleiche Facette?
  - Erzeuge einen Graph G
    - Die Zyklen sind die Knoten
    - Zusätzlich gibt es einen Knoten für den imaginären Rand der Konfiguration
    - Zwei Zyklen/Knoten Z_1, Z_2 werden verbunden, wenn Z_1 ein Lochrand ist und Z_2 eine Halbkante e direkt links vom linkesten Knoten in Z_1 enthält
  - Die Halbkante e kann schon während des Plane-Sweeps bestimmt werden!

Facetten

- Welche Zyklen begrenzen die gleiche Facette?
  - Die Zusammenhangskomponenten von G sind die Facetten

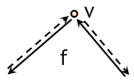
Facetten

- Welche Beschriftung erhält eine Facette f?
  - Es sei v ein Knoten von f der als Schnittpunkt von Kanten aus S_1 und aus S_2 entstanden ist ➞ aus den IncidentFace Zeigern können die angrenzenden Facetten und deren Beschriftung bestimmt werden



Facetten

- Welche Beschriftung erhält eine Facette f?
  - Ist v ein Knoten von f (oBdA) aus S_1 der kein Schnittpunkt ist, so kennen wir zunächst nur die Facette aus S_1, die f enthält
  - Um die Facette aus S_2 die f enthält zu bestimmen, müssen wir herausfinden, in welcher Facette von S_2 der Knoten v liegt ➞ einfache Modifikation des Plane-Sweep Algorithmus



Fazit

- Es sei S_1 eine Unterteilung mit Komplexität n_1 und S_2 eine Unterteilung mit Komplexität n_2. Die Überlagerung von S_1 und S_2 kann in

  $$O(n \log n + k \log n)$$

  Schritten berechnet werden, wobei $n = n_1 + n_2$ und k die Komplexität der Überlagerung ist.