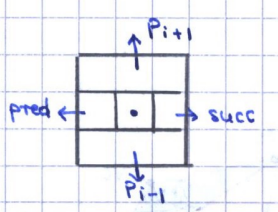
2.2.2. Darstellung im Rechner:

- P_i , $0 \le i \le k$ als doppelt verkettete Liste der Ecken + jede Ecke von P_i zeigt zusätzlich auf ihre Kopie in P_{i+1} und umgekehrt

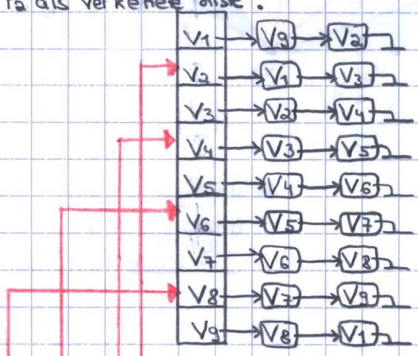


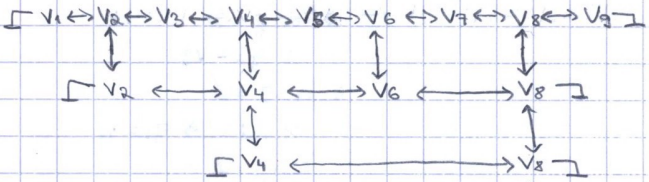
- explizite Speicherung des Kantenbaumes
  → Dynamisierung (Ecken einfügen / löschen)
  (wie bei Suchbäumen)

2.2.3 Beispiel:

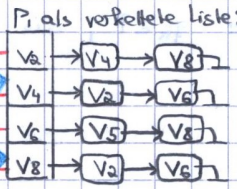
Sei P gegeben wie in 2.1.9.

P_2 als verkettete Liste:





P_1 als verkettete Liste:



P_0 als verkettete Liste:

2.2.4. Anwendung: Schnitt mit einer Geraden:

→ 2.2.4.1. Ziel:

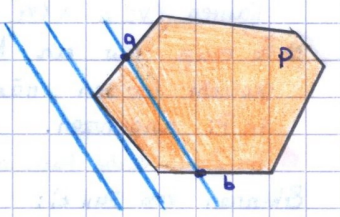
Geg: Hierarchische Darstellung $P_0 .. P_k$ eines konvexen Polygons P und eine Gerade g.

Ausgabe: $P \cap g$

Konvexität → $P \cap g$ = Strecke

→ Es genügt Schnittpkte a, b mit den Randsegmenten zu berechnen.

Fälle: 1) $a \neq b$ (allg. Fall, Ecken möglich)
2) $a = b$ (eine Ecke)
3) ex. nicht!, Schnitt = $\emptyset$



2.2.4.2. Idee für Algorithmus:

1) Schnittest mit P_0:
   2 Fälle: a) $P_0 \cap g = \emptyset$ konst Operationen, weil $|P_0| \le 4$
   b) $P_0 \cap g \neq \emptyset$
   Ann: wir haben Fall a)
   → sei $p_0 \in P_0$ mit minimalen Abstand zu g
   Im Fall b → STOP

d). Durchlaufe die hierarchische Darstellung $P_1, P_2 \ldots$ und finde entweder:

a) $q_i \in P_i$ mit minimalen Abstand, falls $P_i \cap g = \emptyset$

oder

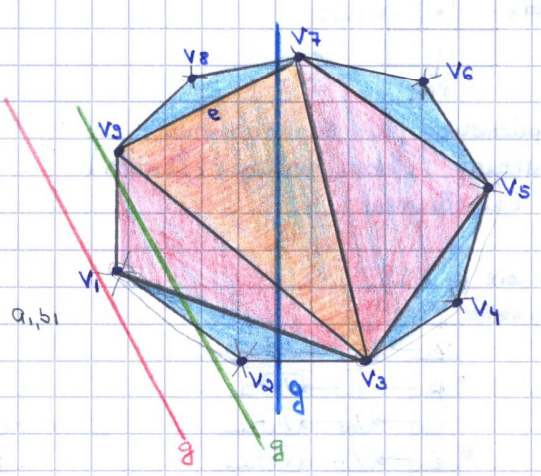
b) Schnittpkte a_i, b_i von P_i mit g.

Im Fall b) → STOP.

Bsp: $P = v_1 .. v_9 = P_2$

$P_1 = v_1 v_2 v_8 v_7 v_9$

$P_0 = v_2 v_7 v_9$



Im Alg.:

1) $P_0 \cap g = \emptyset$
   ⇒ $q_0 = v_9$
2) $P_1 \cap g \neq \emptyset$
   ⇒ Fall b) ⇝ finde a_1, b_1
   STOP

Im Alg.:

1) $P_0 \cap g = \emptyset$
   ⇒ $q_0 = v_9$
2) $P_1 \cap g \neq \emptyset$
   ⇒ Teil a) ⇒ $q_1 = v_1$

   $P_2 \cap g = \emptyset$ -
   ⇒ Teil a) ⇒ $q_2 = v_1$.

Im Alg.:

1) $P_0 \cap g \neq \emptyset$
   ⇒ STOP

Zwei mögliche Ausgänge: (wenn $i = k$)

a) $P_k \cap g = P_n \cap g = \emptyset$ und wir kennen Ecke $q_k \in P$, die g am nächsten

b) $P_i \cap g = (a_i, b_i)$ für $0 \leq i \leq k$

Im Bsp:

- ⇝ Ausgang b) ⇒ $P_1 \cap g = (a_1, b_1)$ für $i = 1$.
- ⇝ Ausgang a) ⇒ $P_n \cap g = \emptyset$, $q_2 = v_1 \in P$.
- ⇝ Ausgang b) ⇒ $P_0 \cap g = (a_0, b_0)$ für $i = 0$

Ausgang a) ⇒ fertig, $P_n \cap g = \emptyset$

Ausgang b) ⇒ Wir müssen Schnittpkte (a_i, b_i) später verfeinern bis (a_k, b_k) gefunden. } Phase 2.

Nun: Übergang von $q_i \to q_{i+1}$ (Fall a)) Bzw. Test, ob Fall b) gilt.

⟶ 2.2.4.3 Laufzeit:

Phase 1: Kandidaten für nächsten Pkt q_{i+1} sind schon genau die Ecken von P_{i+1} die Verfeinerungen der Nachbarkanten von q_i in P_i

Das folgt unmittelbar aus der Konvexität.

- Im Bsp: $i = 0$, $q_0 = v_9$, Kand. für q_1 ist Ecke von P_1 nämlich v_1
  Dies ist die Ecke der Verfeinerungen der Nachbarkanten von $q_0 = v_9$, nämlich der Kanten $(v_9, v_1) \wedge (v_1, v_3)$

⇒ Es muß nur ein Minimum in einer konstanten Zahl von Kandidaten gesucht werden, nämlich in den Verfeinerungen der Kanten, die an q_i in P_i angrenzen.

- Im Bsp: Suche Min. in (v_9, v_1), (v_1, v_3)

Erkennen von Fall b):

Teste in der Kandidatenmenge, ob ein Pkt auf der anderen Seite von g liegt. (Orientation)

⇒ (a_{i+1}, b_{i+1}) in Konst. Zeit denn Konst viele Kand

(Schnittberechnung mit Verfeinerungskanten).

- Im Bsp: $i = 0$, $q_0 = v_9$ ⇒ Kandidatenmenge: (v_9, v_1) und (v_1, v_3)

  v_1 liegt auf der anderen Seite von g, denn $\text{orient}(v_9, v_1, g) > 0$

  ⇒ Teil b) erkannt

  (bez: $\text{orient}(v_9, v_3, g) \leq 0$)

Laufzeit: • $O(1)$ pro Hierarchiestufe
• Haben nur $O(\log n)$ Stufen
$\Rightarrow O(\log n)$ für die erste Phase.

Nach Phase 1 haben wir zwei Fälle:
Fall a) $\Rightarrow$ fertig.
Fall b) $\Rightarrow$ machen weiter zu Phase 2.

Phase 2: Geg: $\{a_i, b_i\} = P \cap g$ für $i < k$ (sonst fertig) und es sind keine Ecken (sonst ebenfalls fertig).

Betrachte a_i (b_i analog):

a_{i+1} muss Schnittpkt sein mit einer Verfeinerungskante von e, wobei $a_i = e \cap g$

Im Bsp: Betrachte a_0, $i = 0 < 2$

a_1 ist Schnpkt mit Verfeinerungskante von e, wobei $a_0 = e \cap g$ und der Geraden g.

Die Verfeinerungskanten von e sind (v_9, v_8) und (v_8, v_7)

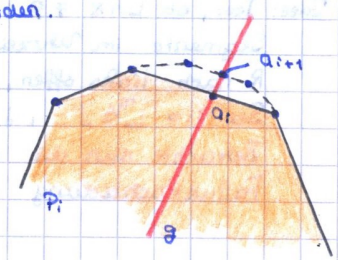
$\Rightarrow a_{i+1}$ kann in Zeit $O(1)$ aus a_i berechnet werden.

(b_{i+1} analog aus b_i)

Weil # Verf.kanten konst. Genauer ≤ 4.

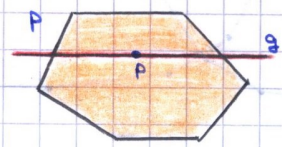
$\Rightarrow$ Laufzeit von Phase 2 auch $O(\log n)$.

$\Rightarrow$ Gesamte Laufzeit also $O(\log n)$!



2.2.5 Satz: Für ein konvexes Polygon P kann:
1) die Hierarchische Darstellung in Zeit $O(n)$ aufgebaut werden.
2) Braucht Platz $O(n)$
3) Für beliebige Gerade g kann $P \cap g$ mit Hilfe der Hierarchischen Darstellung in Zeit $O(\log n)$ Berechnet werden.

2.2.6 Bem.
1) Teil 3) $\Rightarrow$ Alternative $\log n$-Alg. zum Test, ob Pkt $p \in P$ ist.
Idee:



$p \in P \Leftrightarrow p \in P \cap g$
- Betr. Gerade g durch p parallel zu x-Achse
- Berechne hierar. Darst.
- Berechne Schnpkte
- Schaue ob $p_x \in [a_x, b_x]$

$\Rightarrow$ Zeit: $O(1) + O(n) + O(\log n) + O(1) = O(n)$!

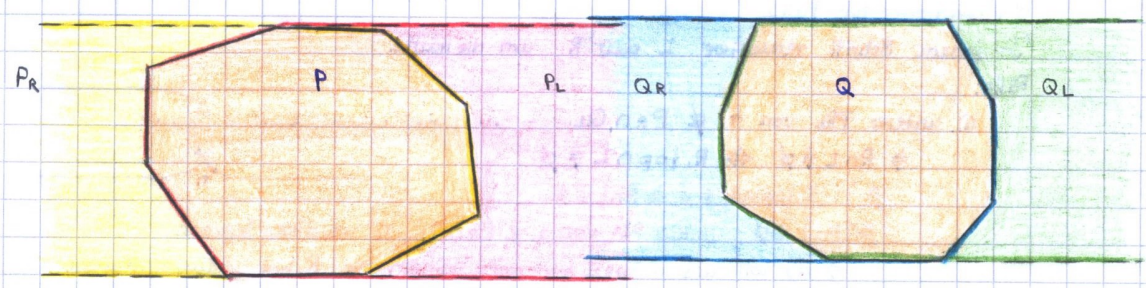
2) Hierar. Darstellung auch im $\mathbb{R}^3$ möglich, d.h. für konvexe Polyeder.
(dazu später mehr...).

2.3 Weiteres Problem auf konvexen Polygonen: Schnitt von zwei konvexen Polygonen.

2.3.1 Ziel: Geg: zwei konvexe Polygone P, Q
Test ob $P \cap Q \neq \emptyset$

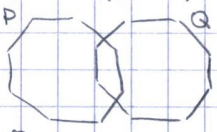
Anm. Wir wissen schon, wie man $P \cap Q$ in Zeit $O(n)$ Berechnet, wobei n = Gesamtzahl der Ecken.

2.3.2 Idee:
1) Betrachte ein einfacheres Problem: Test, ob sich zwei polygonale Ketten schneiden.
- Zerlege P in linken (P_L) & rechten (P_R) Rand durch Zerschneiden an Extremalpkten gemäß yx-Sortierung der Ecken.
- Erweitere durch entsprechende horizontale Strahlen.



Beobachtung: $P \cap Q \neq \emptyset \Leftrightarrow P_L \cap Q_R \neq \emptyset \wedge P_R \cap Q_L \neq \emptyset$

Beweis: $P \cap Q \neq \emptyset \Rightarrow P_R \cap Q_L \neq \emptyset \wedge P_L \cap Q_R \neq \emptyset$ klar.

Für 

mit $P_R \cap Q_L \neq \emptyset \Leftrightarrow P \cap Q \neq \emptyset$
(hierbei gilt auch $P_L \cap Q_R \neq \emptyset$)

Für 

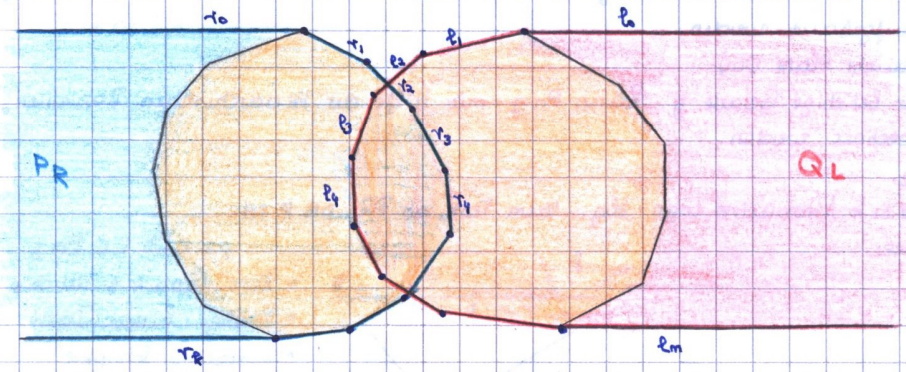
mit $Q_R \cap P_L \neq \emptyset \Leftrightarrow P \cap Q \neq \emptyset$
(hierbei gilt auch $Q_L \cap P_R \neq \emptyset$)

$\Rightarrow$ falls $P_R \cap Q_L \neq \emptyset \wedge P_L \cap Q_R \neq \emptyset \Rightarrow P \cap Q \neq \emptyset$

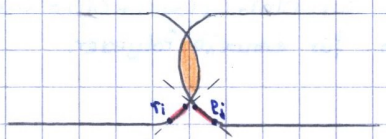
2). Lösung des einfacheren Problems:

Bea: Wir wissen nicht, ob P linkes und Q rechtes Polygon oder umgekehrt.

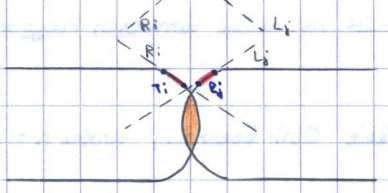
Idee: Test, ob $L \cap R \neq \emptyset$ in Zeit $O(\log n)$, wobei $R = r_0 .. r_k$ gegeben durch die Segmente im Uhrzeigersinn (r_0, r_k Strahlen) und $L = \ell_0 .. \ell_m$ analog gg. Uhrzeigersinn.
R nach links offen, L nach rechts offen.
Sei $i := \lfloor m/2 \rfloor$, $j := \lfloor k/2 \rfloor$
Betrachte nun die mittleren Segmente r_i und ℓ_j.
Fallunterscheidung gemäß der Lage von ℓ_j und r_i auf Geraden R_i und L_j.



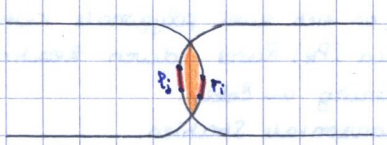
Es gibt nun mehrere Fälle:

1) 

← $P_R \cap Q_L$ oberhalb von den unteren Endpkten von r_i und ℓ_j

2) 

← $P_R \cap Q_L$ unterhalb von den oberen Endpkten von r_i und ℓ_j

3) 

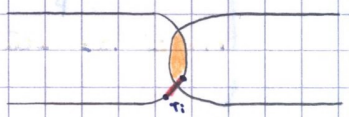
← Endpkte von r_i und $\ell_j \in P_R \cap Q_L$
(Bea: hierbei müssen sich die Polygone nicht notw. schneiden!)

Wir betrachten hier Fall 1

In jedem Schritt reduziere L oder R um die Hälfte.

Fälle:

a). unterer Pkt von $r_i \notin P_R \cap Q_L$



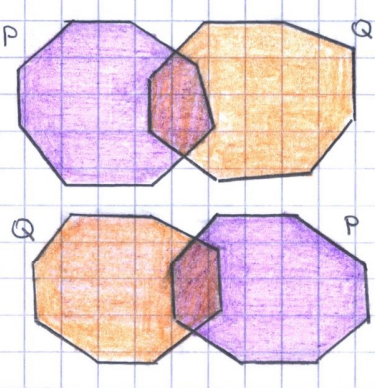
$\Rightarrow P \cap L \neq \emptyset \Leftrightarrow R.top \cap L \neq \emptyset$

"$\Leftarrow$": $R.top \cap L \neq \emptyset \Rightarrow R \cap L \neq \emptyset$

"$\Rightarrow$" $P \cap L \neq \emptyset$, unterer Pkt von $r_i \notin P_R \cap Q_L$, r_i mittleres Segment
$\Rightarrow$ da wir uns im Fall 1 befinden, können wir R.bot rausschmeissen.
$\Rightarrow R.top \cap L \neq \emptyset$.

Beobachtung: $P \cap Q \neq \emptyset \Leftrightarrow P_L \cap Q_R \neq \emptyset \wedge P_R \cap Q_L \neq \emptyset$.

Denn:



Hier: $P \cap Q \neq \emptyset \Leftrightarrow P_R \cap Q_L \neq \emptyset$

Beo: Man weiß in der Regel nicht, welches Polygon rechts und welches links liegt.

Hier: $P \cap Q \neq \emptyset \Leftrightarrow P_L \cap Q_R \neq \emptyset$

$\Rightarrow$ Insgesamt gilt also: falls $P_L \cap Q_R \neq \emptyset \wedge P_R \cap Q_L \neq \emptyset \Rightarrow P \cap Q \neq \emptyset$

„$\Leftarrow$" klar

$\Rightarrow$ die obige Beh.

2) Lösung des einfacheren Problems:

Idee: Test, ob $L \cap R \neq \emptyset$ in Zeit $O(\log n)$, wobei $R = \{r_0 \dots r_k\}$ und $L = \{\ell_0 \dots \ell_m\}$ gegeben durch die Segmente im Uhrzeigersinn (für R) und gegen Uhrzeigersinn (für L).

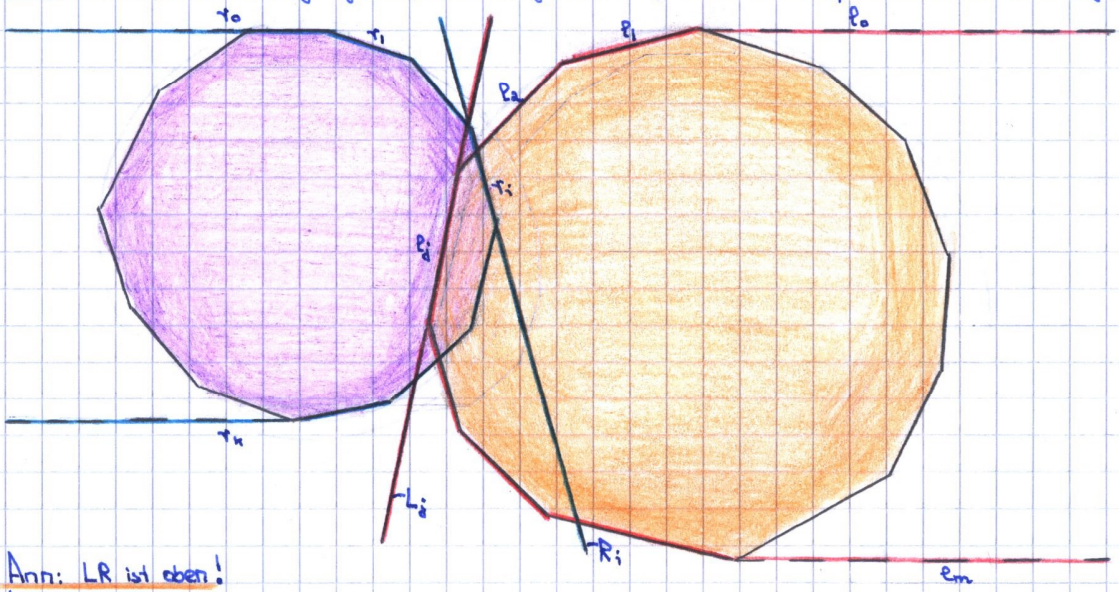
(r_0 und r_k bzw ℓ_0 und ℓ_m sind Strahlen).

R ist nach links offen und L nach rechts offen.

Sei $i := \lfloor m/2 \rfloor$ und $j := \lfloor k/2 \rfloor$

Betrachte nun die mittleren Segmente r_i und ℓ_j

Nun: Fallunterscheidung gemäß der Lage von ℓ_j und r_i auf Geraden R_i und L_j:



Ann: LR ist oben!

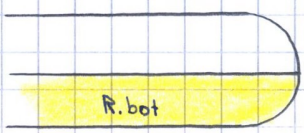
In jedem Schritt reduziere L oder R um die Hälfte.

Fälle:

a) unterer Pkt von r_i ist nicht in LR

$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R.top \cap L \neq \emptyset$

wobei:



b). unterer Pkt von ℓ_j ist nicht in LR.

$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R.L.top \neq \emptyset$

c) beide unteren Pkte sind in LR

(i) falls u_{r_i} unterhalb von u_{ℓ_j}

$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R.top \cap L \neq \emptyset$

(ii) falls u_{ℓ_j} unterhalb von u_{r_i}

$\Rightarrow R \cap L \neq \emptyset \Leftrightarrow R \cap L.top \neq \emptyset$

Hierbei: $u_{r_i} \hat{=}$ unterer Endpkt von r_i

$u_{\ell_j} \hat{=}$ unterer Endpkt von ℓ_j.

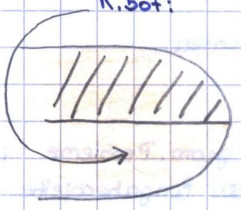
⇒ In Zeit $O(1)$ wird entweder L oder R auf die Hälfte reduziert.
Wiederhole nun bis l_j sich mit r_i schneidet oder bis nur noch konstante Größe von L und R.

2.3.3. Laufzeit: sei $k + m + 2 =: n$
⇒ der eigentliche Test, ob $L \cap R \neq \emptyset$ in Zeit $O(\log n)$.

Kleines Beispiel zur Veranschaulichung:

Bea: R.top:

R.bot:



b) unterer Pkt von l_j nicht $\in L \cap L$
$\Rightarrow R \cap L \neq 0 \Leftrightarrow R \cap L.top \neq \emptyset$
$\Rightarrow$ Man teilt also so weiter bis nur ein Segment übrig bleibt.

c) beide unteren Pkte $\in L \cap L$
$\Rightarrow$ Ketten schneiden sich.

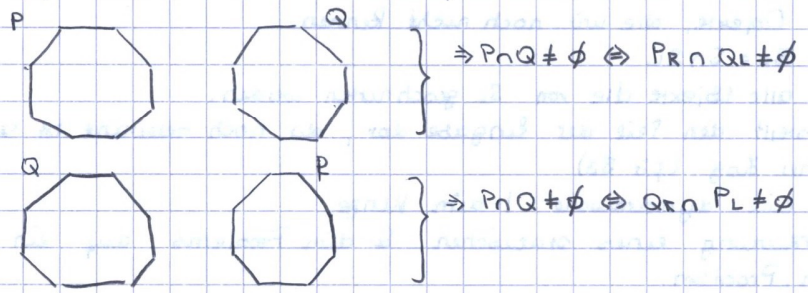
$\Rightarrow$ In Zeit $O(1)$ wird entweder L oder R auf die Hälfte reduziert.

Wiederhole bis r_i sich mit l_j schneidet oder bis nur noch konstante Größe von L und R also bis nur jeweils ein Segment übrig geblieben ist.

2.3.3. Laufzeit:
Sei $k+m =: n$. $\Rightarrow$ Test ob $L \cap R \neq \emptyset$ in Zeit $O(\log n)$.

Zusatz: Bea: $P \cap Q \neq \emptyset \Leftrightarrow P_L \cap Q_R \neq \emptyset \wedge P_R \cap Q_L \neq \emptyset$

Skizze:



$\Rightarrow P \cap Q \neq \emptyset \Leftrightarrow P_R \cap Q_L \neq \emptyset \wedge Q_R \cap P_L \neq \emptyset$

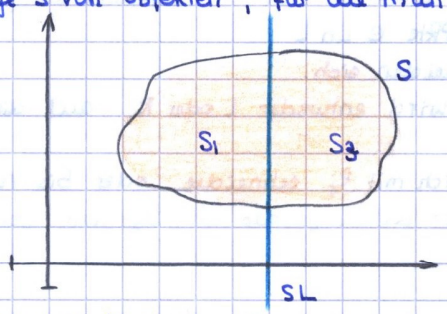
Ferner: $P \cap Q \neq \emptyset \Leftrightarrow L \cap R \neq \emptyset$
$\Rightarrow$ Haben unser Problem auf ein einfacheres zurückgeführt und es mit Zeit $O(\log n)$ gelöst.
Wg. Äquivalenz $\Rightarrow$ Test ob $P \cap Q \neq \emptyset$ in Zeit $O(\log n)$.

Kapitel III: Das Plane Sweep Verfahren.

3.1. Einführung:

3.1.1. Idee:

Allg. Ansatz zur Lsg. geom. Probleme in der Ebene, die inkrementell durch schrittweises Betrachten der Eingabeobjekte in einer bestimmten Reihenfolge gelöst werden können.
d.h. meistens von links nach rechts (xy-Ordnung).
Genauer: Man schiebt / bewegt eine sog. Sweepline, also eine senkrechte Gerade SL über die Menge S von Objekten, für die man ein Problem lösen möchte.



Im Allg. zerlegen wir S in drei Mengen:
- S_1 links von SL
  (Teilproblem für S ist gelöst).
- S_3 rechts von SL
  Objekte, die wir noch nicht kennen.
- $S_2 = S \cap SL$
  alle Objekte die von SL geschnitten werden.

3.1.2. Bem:
S_2 stellt den Teil der Eingabe dar, der noch relevant ist zur Berechnung restliches Lsg (für S_3)
S_2 ist eine dynamische 1-dim. Menge.
⇒ Reduzierung eines statischen 2-dim. Problems auf ein dynamisches 1-dim. Problem.
(die dynam. 1-dim. Probleme werden balancierte Suchbäume sein.).

3.1.3. Bem:
Wir haben spezielle Varianten des SL-Verfahrens kennen gelernt:
- Inkrementelle konv. Hülle
  13.8. Grahams Scan für beide Hüllen gleichzeitig
- Triangulierung.
  Übung.

3.2. Line Segment Intersection.
Schnitt von Geradensegmenten / Strecken.

3.2.1. Problem:
Geg: Menge S von n Segmenten
Ges: alle Schnittpkte.
(Bzw. Unterteilung der Ebene in Graph)
Grundoperation: $s_i \cap s_j \quad \forall s_i, s_j \in S$
(dies geht in konst. Zeit O(1) mit orientation-Test).

3.2.2. Triviale Lsg:
Teste alle Paare nacheinander
⇒ Laufzeit $O(n^2)$.

3.2.3. Ziel:
outputsensitiver Algorithmus, d.h. Laufzeit hängt von Zahl der Schnittpkte ab.

3.2.4. Idee für einen Plane Sweep Algorithmus: