3. Das "Plane-Sweep" Verfahren

3.1 Einleitung

- "Plane Sweep":
  - Allg Ansatz zur Lsg geom Probleme in der Ebene.
  - Die Eingabe wird schrittweise in einer bestimmten Reihenfolge (meist von links nach rechts) betrachtet und dabei wird das Problem schrittweise gelöst.
  - Die Eingabemenge S wird in 3 Mengen zerlegt:
    - S_1: links von SL
      Teilproblem ist für S_1 gelöst
    - S_3: rechts von SL
      Objekte die wir noch nicht kennen
    - $S_2 := S \cap SL$
      dynamische 1-dim Menge ⇒ Lösen des Problems für $S_1 \cup (S \cap SL)$
  ⇒ Reduzierung eines statischen 2-dim Problems auf ein dynamisches 1-dim.

- Schon mit Plane Sweep berechnet:
  - Inkrementelle konv Hülle ⇝ Graham's Scan
  - Triangulierung

3.2 Anwendung I: Schnitt von Segmenten

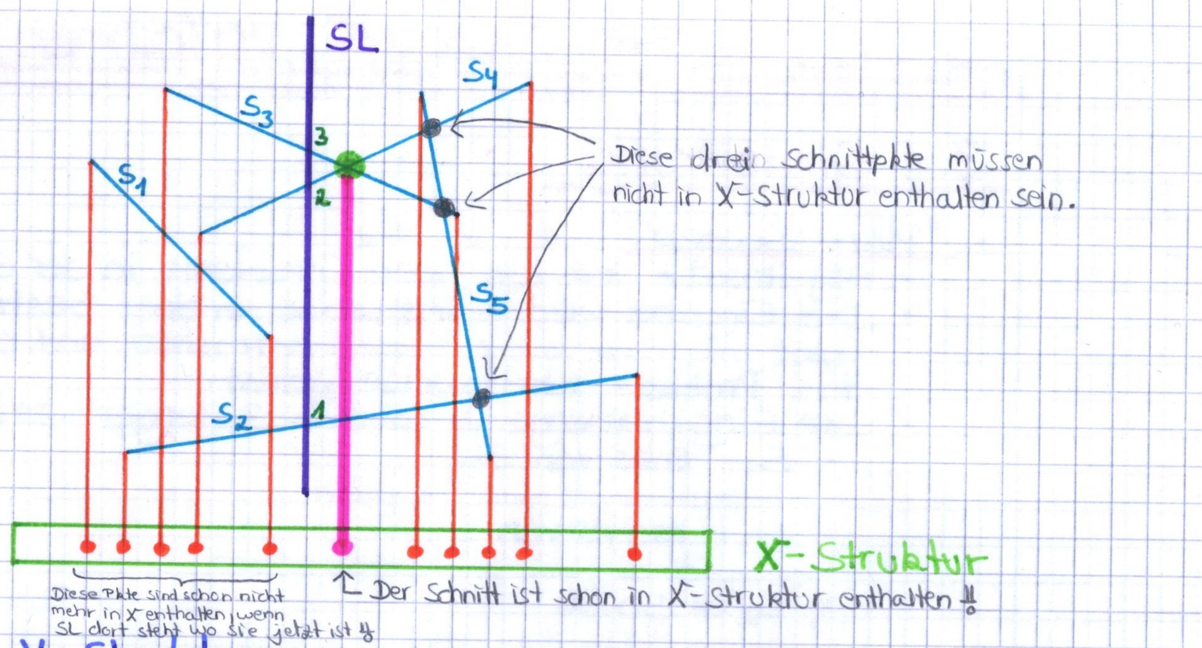
Geg.: Menge S von n Segmenten
Ges.: Alle Schnittpkte

- Trivialer Algorithmus:
  Teste alle Paare → $O(n^2)$ schlecht!!

- Ziel:
  Output-sensitiver Algorithmus, dh Laufzeit abhängig von # Schnittpkte

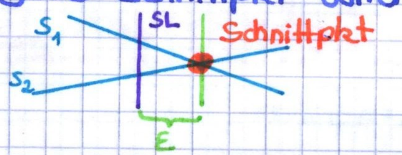
- Da man bei Plane Sweep Verfahren allgemein Events von links nach rechts verarbeitet, benötigt man Datenstrukturen für diese Events: X-Struktur und Y-Struktur

- X-Struktur:
  Verwaltung aller Positionen von SL an denen sich $S \cap SL = S_2$ verändert. (S_2 = Folge der von SL geschnittenen Segmente, sortiert nach y-Koord der Schnittpkte mit SL)
  S_2 verändert sich wenn neue Segmente anfangen, alte aufhören oder wenn man einen Schnittpkt von 2 Segmenten erreicht.
  Diese Stellen nennt man Events
  - Statische X-Struktur: (einfache Liste)
    Dh alle Events sind bekannt zB bei konvexe Hülle, Closest Pair, ...
    X-Struktur = sortierte Liste der Eingabepkte
  - Dynamische X-Struktur: (dynamische Warteschlange)
    Dh Events sind nicht alle bekannt, sie werden (zum Teil) während des Sweeps berechnet. zB Segmentschnitt



- **Y-Struktur:**
  Verwaltung der Menge ~~[illegible]~~, dh. der Segmente welche aktuell einen ~~aktuellen~~ Schnittpkte ~~von Segmenten~~ mit SL haben
  Diese Segmente sollen von unten nach oben (dh gem ihrem Schnittpkt mit SL) entlang der SL sortiert werden.

- Dadurch können Schnitttests auf benachbarte Segmente beschrkt werden, nur diese müssen in X-Struktur enthalten sein.
  ⇒ Jeder Schnittpkt wird ε vor ihm erkannt



- **Operationen auf X- bzw Y-Struktur beim Segmentschnitt:**
  - **X-Struktur:**

X.insert (p)	// p = Event-Pkt	$O(\log n)$
X.delete (p)		$O(\log n)$
X.findmin ()	// nächstes Event	$O(1)$

  - **Y-Struktur:**

Y.insert (s)		$O(\log n)$
Y.delete (s)		$O(\log n)$
Y.swap (s_1, s_2)		$O(1)$
Y.succ (s)	// Nachfolger	$O(1)$
Y.pred (s)	// Vorgänger	$O(1)$

- **Implementierung von X- und Y-Struktur:**
  - balancierter, blattorientierter binärer Baum
  - Für Y-Struktur: zusätzliche Verkettung der Blätter

- **Invariante:**
  X-Struktur enthält zu jedem Zeitpkt:
  - alle Endpkte rechts von SL
  - alle Schnittpkte von in Y benachbarten Segmenten rechts von SL

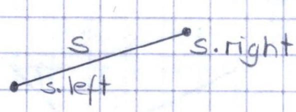
⇒ max $n-1$ Schnittpkte in X-Struktur ($\exists$ n Segmente)
⇒ **Platzbedarf: $O(n)$** wir speichern am Anfang alle linken & rechten Endpkte, das sind aber auch nur linear viele
Ohne Invariante bis zu n^2 Schnittpkte ⇒ Platzbedarf quadratisch, wäre schlecht!!

- **Algorithmus:**

SWEEP(S)

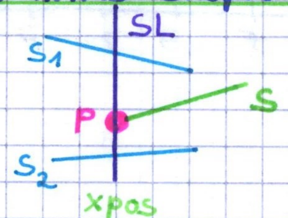
- $\mathcal{X}$-Struktur $\mathcal{X} \leftarrow \emptyset$ — $O(1)$
  $\mathcal{Y}$-Struktur $\mathcal{Y} \leftarrow \emptyset$ — $O(1)$
- double xpos $\leftarrow -\infty$ // xpos ist die Position der SL — $O(1)$
- forall $s \in S$ do
  - $\mathcal{X}$.insert (s.left)
  - $\mathcal{X}$.insert (s.right)

  od — $O(n \log n)$

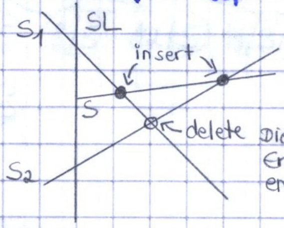


- while ($\mathcal{X} \neq \emptyset$) {
  - $p \leftarrow \mathcal{X}$.findmin()
    xpos $\leftarrow$ p.xcoord() // schiebt SL zum Pkt p

    Fallunterscheidungen gemäß der verschiedenen Events:
  - switch (p) {
  - case linker Endpkt von s: { 6



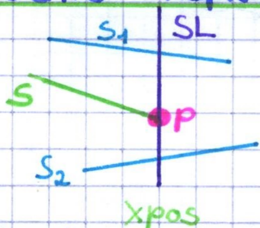
$\mathcal{Y}$.insert (s)
$s_1 \leftarrow \mathcal{Y}$.succ(s) — s_1 exist nicht immer
$s_2 \leftarrow \mathcal{Y}$.pred(s) — s_2 ——— " ———
$\mathcal{X}$.delete ($s_1 \cap s_2$) — $s_1 \cap s_2$ ——— " ———
$\mathcal{X}$.insert ($s_1 \cap s$) — $s_1 \cap s$ ——— " ———
$\mathcal{X}$.insert ($s \cap s_2$) — $s \cap s_2$ ——— " ———



Dieser Schnittpkt wird jetzt gelöscht um Platz zu sparen. Er wird aber später (wenn die SL nahe genug ist) wieder entdeckt.

}

- case rechter Endpkt von s: { 4



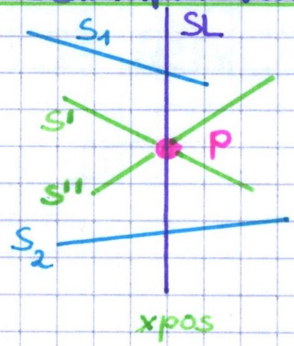
Geht nicht !! Weil man in der $\mathcal{Y}$-Strukt. nur Vorgänger & Nachf. kennt. Und wenn s weg ist, weiß man nicht mehr dass es zw. s_1 und s_2 war, dass man diese beiden jetzt neu "verbinden" muss.

$\mathcal{Y}$.delete(s)
$s_1 \leftarrow \mathcal{Y}$.succ($s_2$)
[$s_2 \leftarrow \mathcal{Y}$.pred($s_1$)]

$s_1 \leftarrow \mathcal{Y}$.succ(s)
$s_2 \leftarrow \mathcal{Y}$.pred(s)
$\mathcal{Y}$.delete(s)
$\mathcal{X}$.insert ($s_1 \cap s_2$) — $s_1 \cap s_2$ liegt <u>recht</u> von SL !!

}

- case schnittpkt von s' und s'': { 8



$s_1 \leftarrow Y.\mathrm{succ}(s')$
$s_2 \leftarrow Y.\mathrm{pred}(s'')$
$Y.\mathrm{swap}(s', s'')$
$X.\mathrm{delete}(s_1 \cap s')$
$X.\mathrm{delete}(s_2 \cap s'')$
$X.\mathrm{insert}(s_1 \cap s'')$
$X.\mathrm{insert}(s_2 \cap s')$
Ausgabe: "$p = s' \cap s''$"

$\boxed{s_2} \to \boxed{s''} \to \boxed{s'} \to \boxed{s_1}$

$\boxed{s_2} \to \boxed{s'} \to \boxed{s''} \to \boxed{s_1}$

}
end switch } $X.\mathrm{delete}(p)$
end while }

- Annahmen:
  - Alle x-Koordinaten von Segment-Anf-, -End- und -Schnittpktn sind paarweise verschieden dh in einem Pkt schneiden sich höchstens 2 Segmente
  - ∄ vertikalen Segmente

- Laufzeit:
  - Alle Operationen auf X und Y benötigen höchstens Zeit $O(\log n)$ (siehe: Operationen auf X bzw Y-Struktur, 2 Seiten vorher)
  - Die Initialisierung vor der while-Schleife benötigt Zeit $O(n \log n)$ Einfügen von 2n Anfangs- bzw Endpktn
  - Die while-Schleife wird $2n+s$ $s \mathrel{\hat{=}} \#$ Schnittpkte mal ausgeführt. Jeder Schleifendurchlauf kostet $O(\log n)$ da in jedem Fall konstant viele Operationen auf X und Y und konstant viele Schnittests ausgeführt werden

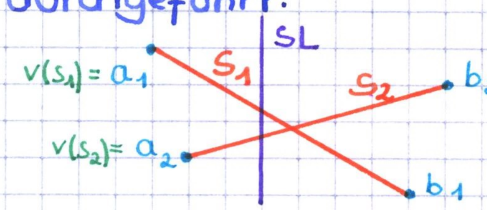
$\Rightarrow$ insgesamt: $O((n+s)\log n) = O(n \cdot \log n + s \cdot \log n)$

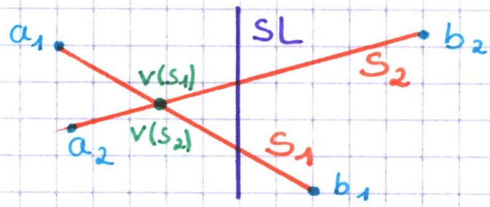
Dies ist wie gewünscht output-sensitiv !!

3.2.1 1. Modifikation

orientation-Tests statt "compare" Fkt

- Zur Definition der linearen Ordnung in der Y-Struktur wurde im obigen Algorithmus die "compare"-Fkt verwendet. Anstelle dieser Fkt werden jetzt nur noch orientation-Tests durchgeführt.





Für jedes Segment s in Y werden der letzte Schnittpkt bzw der linke Endpkt als v(s) gespeichert.

1. Fall: $\mathrm{orientation}(v(s_1), b_1, v(s_2)) > 0$
$\Rightarrow s_1 < s_2$

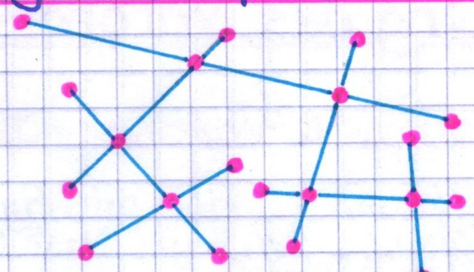
2. Fall: $\mathrm{orientation}(v(s_1), b_1, v(s_2)) < 0$
$\Rightarrow s_1 > s_2$

3. Fall: $\mathrm{orientation}(v(s_1), b_1, v(s_2)) = 0$
$\Rightarrow v(s_1) = v(s_2)$

$$\Rightarrow \mathrm{orientation}(v(s_2), b_1, b_2) \begin{cases} < 0 & \Rightarrow s_1 > s_2 \\ = 0 & \Rightarrow \text{collinear} \\ > 0 & \Rightarrow s_1 < s_2 \end{cases}$$

3.2.2 2. Modifikation   Ausgabe des planaren Graphen

- Ausgabe des planaren Graphen:

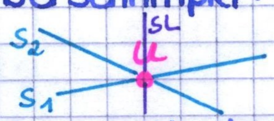


$G = (V, E)$

V: Anf-, End- und Schnittpkte

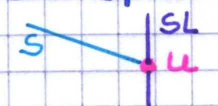
E: Intervalle durch Zerlegung der Segmente durch V

- Erzeugung von Knoten:
  - für jeden Anf- und Endpkt der Segmente
  - für jeden Schnittpkt der Segmente
- Erzeugung von Kanten:
  - bei Schnittpkt:



2 neue Kanten entstehen:
$v(s_1) \leftrightarrow u$ und $v(s_2) \leftrightarrow u$ (jeweils 2 fach gerichtet)

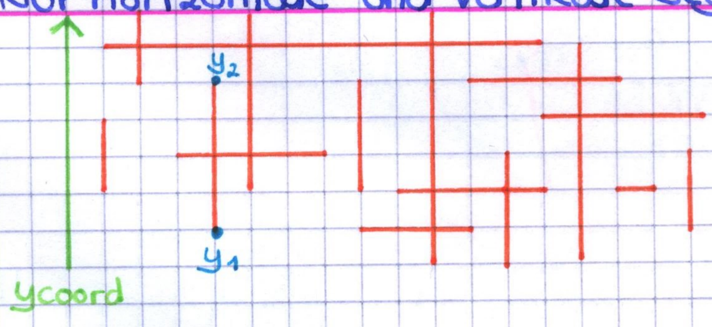
  - bei Endpkt:



1 neue Kante entsteht:
$v(s) \leftrightarrow u$ (2 fach gerichtet)

3.2.3 3. Modifikation   nur horizontale und vertikale Segmente

- Nur horizontale und vertikale Segmente:



$n = 16$ (# Segmente)
$s = 10$ (# Schnittpkte)
$\ell = 9$ (# vertikale Segmente)

- Events:
  - linker Endpkt von Horizontalen
  - rechter ——— " ———
  - vertikales Segment

} Einfügen und löschen
} Test ob Schnittpkte exist

- 1-dim Bereichsabfrage: hat immer Zeit $O(\log n + s)$ s = Ausgabe
  Welche horizontalen Segmente in Y drinstehen

$$\sum_{i=1}^{\ell} \log n + k_i \quad \text{für } \ell \text{ vertikale Segmente}$$

Hier: $\log n + 0 + \log n + 1 + \log n + 1 + \log n + 2 + \log n + 0 + \log n + 3 + \log n + 1 + \log n + 2 + \log n + 0 =$
$= 9 \cdot \log n + 10$

$\Rightarrow O(n \cdot \log n + s)$
Hier zwar $9 = \ell \neq n$ aber es könnte ja sein, dass nur vertikale Segmente existieren!! (dann wäre $\ell = n$)

- Laufzeit:
  $O(n \cdot \log n + s)$