

Algorithmen II

Vorlesung am 29.11.2012

Algorithmische Geometrie: Schnitte von Strecken – Sweep-Line

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

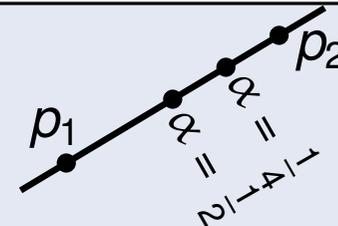


Grundlagen

Definition: Konvexkombination

(Definition 6.1)

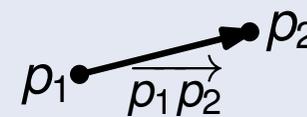
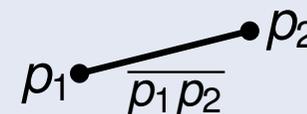
Für zwei Punkte p_1 und p_2 (in der Ebene) und $\alpha \in [0, 1]$ ist der Punkt $p = \alpha p_1 + (1 - \alpha)p_2$ eine *Konvexkombination* von p_1 und p_2 .



Definition: Strecke

(Definition 6.2)

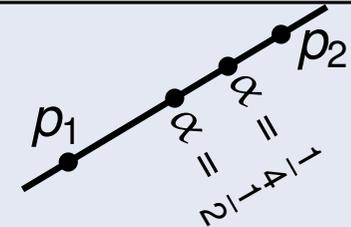
Die Menge aller Konvexkombinationen zweier Punkte p_1, p_2 ist die *Strecke* $\overline{p_1 p_2}$ zwischen p_1 und p_2 . Ist die Reihenfolge von p_1 und p_2 von Bedeutung, so sprechen wir von der *gerichteten Strecke* $\overrightarrow{p_1 p_2}$.



Definition: Konvexkombination

(Definition 6.1)

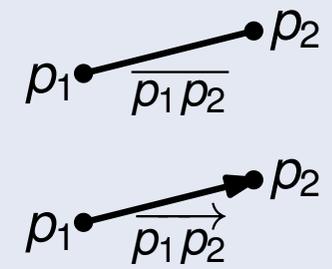
Für zwei Punkte p_1 und p_2 (in der Ebene) und $\alpha \in [0, 1]$ ist der Punkt $p = \alpha p_1 + (1 - \alpha)p_2$ eine *Konvexkombination* von p_1 und p_2 .



Definition: Strecke

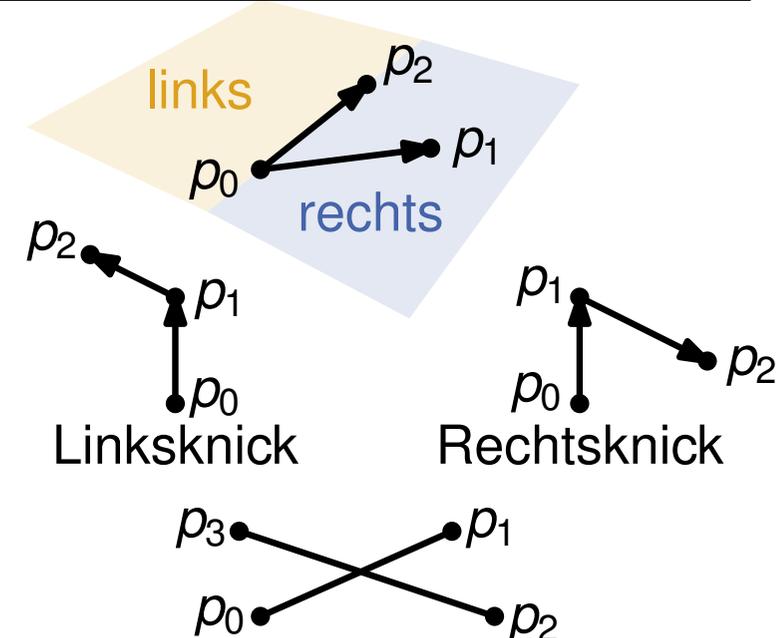
(Definition 6.2)

Die Menge aller Konvexkombinationen zweier Punkte p_1, p_2 ist die *Strecke* $\overline{p_1 p_2}$ zwischen p_1 und p_2 . Ist die Reihenfolge von p_1 und p_2 von Bedeutung, so sprechen wir von der *gerichteten Strecke* $\overrightarrow{p_1 p_2}$.



Grundlegende Fragestellungen:

1. Gegeben drei Punkte p_0, p_1, p_2 in der Ebene. Liegt $\overrightarrow{p_0 p_1}$ rechts oder links von $\overrightarrow{p_0, p_2}$?
2. Bilden die beiden Strecken $\overrightarrow{p_0, p_1}$ und $\overrightarrow{p_1, p_2}$ bei p_1 einen Rechts- oder einen Linksknick?
3. Schneiden sich zwei gegebene Strecken $\overline{p_0 p_1}$ und $\overline{p_2 p_3}$?



Rechts oder Links?

Definition: Kreuzprodukt

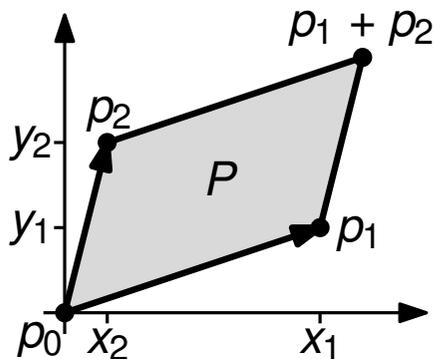
Für Strecken $\overrightarrow{p_0 p_1}$ und $\overrightarrow{p_0 p_2}$ mit $p_0 = (0, 0)$, $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$ ist das *Kreuzprodukt* $p_1 \times p_2$ definiert als

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -(p_2 \times p_1)$$

Behauptung: Fläche eines Parallelprogramms

(Siehe Lineare Algebra)

$|p_1 \times p_2|$ ist die Fläche P des Parallelogramms mit Ecken $(0, 0)$, p_1 , p_2 und $p_1 + p_2$.



Definition: Kreuzprodukt

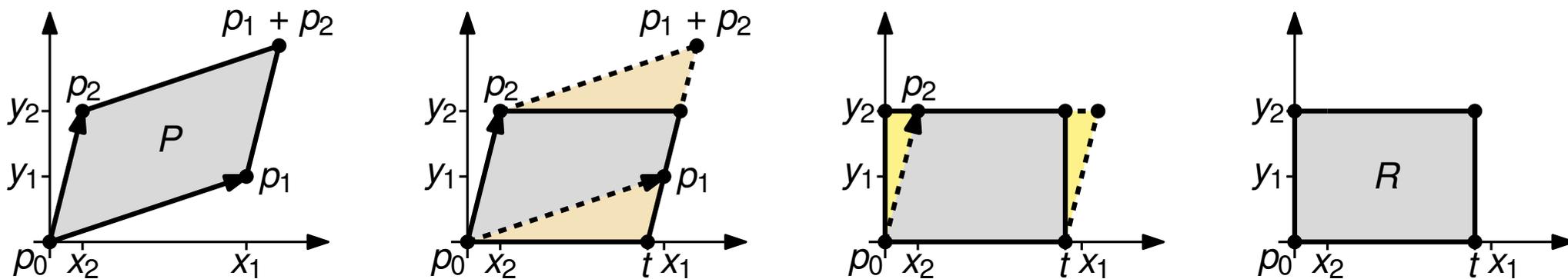
Für Strecken $\overrightarrow{p_0 p_1}$ und $\overrightarrow{p_0 p_2}$ mit $p_0 = (0, 0)$, $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$ ist das *Kreuzprodukt* $p_1 \times p_2$ definiert als

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -(p_2 \times p_1)$$

Behauptung: Fläche eines Parallelprogramms

(Siehe Lineare Algebra)

$|p_1 \times p_2|$ ist die Fläche P des Parallelogramms mit Ecken $(0, 0)$, p_1 , p_2 und $p_1 + p_2$.



Lemma

Für $p_0 = (0, 0)$, p_1 und p_2 liegt $\overrightarrow{p_0 p_1}$ rechts von $\overrightarrow{p_0 p_2}$ genau dann wenn $p_1 \times p_2 > 0$.

Lemma

Für $p_0 = (0, 0)$, p_1 und p_2 liegt $\overrightarrow{p_0 p_1}$ rechts von $\overrightarrow{p_0 p_2}$ genau dann wenn $p_1 \times p_2 > 0$.

- Um nun zu entscheiden, ob $\overrightarrow{p_0 p_1}$ für beliebiges p_0 rechts von $\overrightarrow{p_0 p_2}$ liegt, muss getestet werden ob $(p_1 - p_0) \times (p_2 - p_0) > 0$ gilt.
- Die Strecken $\overrightarrow{p_0 p_1}$ und $\overrightarrow{p_1 p_2}$ bilden einen Rechtsknick genau dann wenn $\overrightarrow{p_0 p_2}$ rechts von $\overrightarrow{p_0 p_1}$ liegt.

Lemma

Für $p_0 = (0, 0)$, p_1 und p_2 liegt $\overrightarrow{p_0 p_1}$ rechts von $\overrightarrow{p_0 p_2}$ genau dann wenn $p_1 \times p_2 > 0$.

- Um nun zu entscheiden, ob $\overrightarrow{p_0 p_1}$ für beliebiges p_0 rechts von $\overrightarrow{p_0 p_2}$ liegt, muss getestet werden ob $(p_1 - p_0) \times (p_2 - p_0) > 0$ gilt.
- Die Strecken $\overrightarrow{p_0 p_1}$ und $\overrightarrow{p_1 p_2}$ bilden einen Rechtsknick genau dann wenn $\overrightarrow{p_0 p_2}$ rechts von $\overrightarrow{p_0 p_1}$ liegt.

Vorteile dieser Methode:

Es müssen Additionen, Subtraktionen, Multiplikationen und Vergleiche durchgeführt werden (insbesondere keine Divisionen, Wurzeln, etc.)

⇒ Bei Ganzzahligen Koordinaten ist das Ergebnis und alle Zwischenschritte Ganzzahlig.

⇒ Keine Fehler durch Ungenauigkeiten bei Gleitkommaarithmetik.

Schneiden sich zwei Strecken?

Definition: Umschließendes Rechteck (bounding box) (Definition 6.4)

Das *Umschließende Rechteck (bounding box)* eines geometrischen Objekts ist das kleinste achsenparallele Rechteck, das diese Objekt enthält.

Notwendige Bedingung: Zwei Strecken können sich nur schneiden, wenn sich auch ihre umschließenden Rechtecke schneiden.

Schneiden sich zwei Strecken?

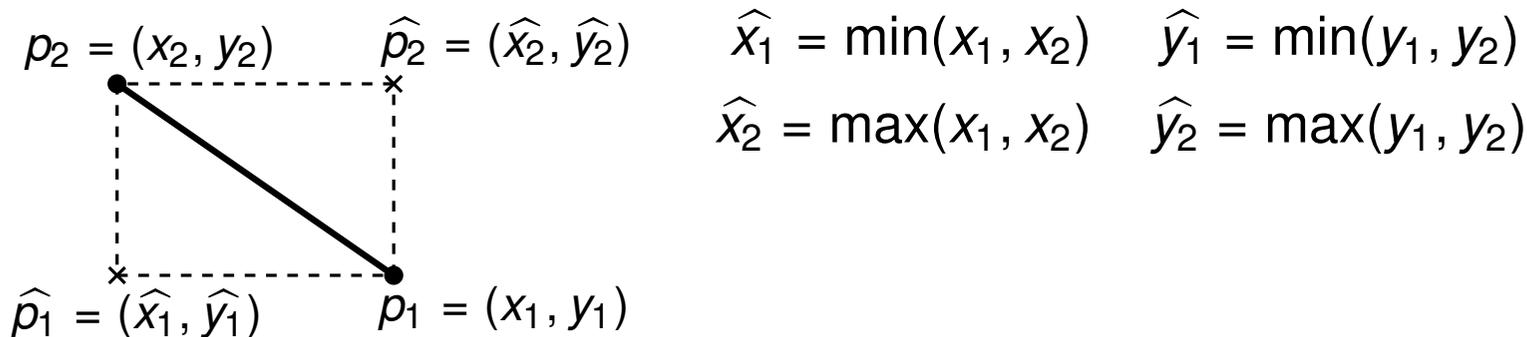
Definition: Umschließendes Rechteck (bounding box)

(Definition 6.4)

Das *Umschließende Rechteck* (*bounding box*) eines geometrischen Objekts ist das kleinste achsenparallele Rechteck, das diese Objekt enthält.

Notwendige Bedingung: Zwei Strecken können sich nur schneiden, wenn sich auch ihre umschließenden Rechtecke schneiden.

Das umschließende Rechteck (\hat{p}_1, \hat{p}_2) zu $\overline{p_1 p_2}$ ist beschrieben durch den linken unteren Punkt $\hat{p}_1 = (\hat{x}_1, \hat{y}_1)$ und den rechten oberen Punkte $\hat{p}_2 = (\hat{x}_2, \hat{y}_2)$ mit:



Schneiden sich zwei Strecken?

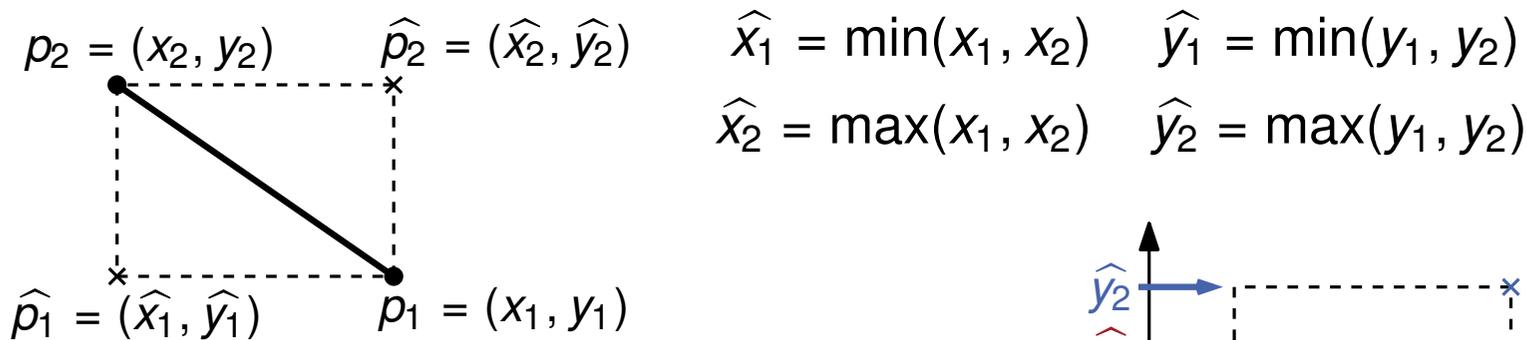
Definition: Umschließendes Rechteck (bounding box)

(Definition 6.4)

Das *Umschließende Rechteck* (*bounding box*) eines geometrischen Objekts ist das kleinste achsenparallele Rechteck, das diese Objekt enthält.

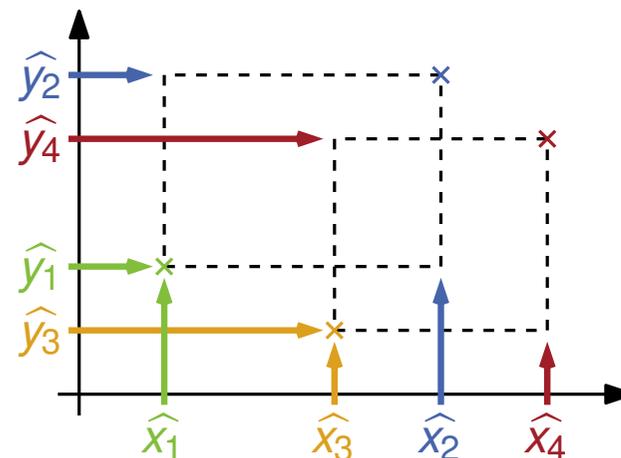
Notwendige Bedingung: Zwei Strecken können sich nur schneiden, wenn sich auch ihre umschließenden Rechtecke schneiden.

Das umschließende Rechteck (\hat{p}_1, \hat{p}_2) zu $\overline{p_1 p_2}$ ist beschrieben durch den linken unteren Punkt $\hat{p}_1 = (\hat{x}_1, \hat{y}_1)$ und den rechten oberen Punkte $\hat{p}_2 = (\hat{x}_2, \hat{y}_2)$ mit:



Zwei Rechtecke (\hat{p}_1, \hat{p}_2) und (\hat{p}_3, \hat{p}_4) schneiden sich genau dann, wenn

$$\hat{x}_2 \geq \hat{x}_3 \wedge \hat{x}_4 \geq \hat{x}_1 \wedge \hat{y}_2 \geq \hat{y}_3 \wedge \hat{y}_4 \geq \hat{y}_1.$$



Schneiden sich zwei Strecken?

Definition

(Definition 6.5)

$\overline{p_1 p_2}$ *berührt* die Gerade durch $\overline{p_3 p_4}$ falls p_1 und p_2 auf unterschiedlichen Seiten von $\overline{p_3 p_4}$ liegen oder einer von ihnen auf der Gerade liegt.

Ob eine Strecke die Gerade durch zwei Punkte berührt kann wieder mithilfe des Kreuzprodukts getestet werden.

Schneiden sich zwei Strecken?

Definition

(Definition 6.5)

$\overline{p_1 p_2}$ berührt die Gerade durch $\overline{p_3 p_4}$ falls p_1 und p_2 auf unterschiedlichen Seiten von $\overline{p_3 p_4}$ liegen oder einer von ihnen auf der Gerade liegt.

Ob eine Strecke die Gerade durch zwei Punkte berührt kann wieder mithilfe des Kreuzprodukts getestet werden.

Lemma

Zwei Strecken schneiden sich genau dann, wenn ihre umschließenden Rechtecke sich schneiden und jede der beiden Strecken die Gerade durch die jeweils andere Strecke berührt.

Vorteile dieser Methode:

Wie zuvor müssen Additionen, Subtraktionen, Multiplikationen und Vergleiche durchgeführt werden (insbesondere keine Divisionen, Wurzeln, etc.)

⇒ Bei Ganzzahligen Koordinaten ist das Ergebnis und alle Zwischenschritte Ganzzahlig.

⇒ Keine Fehler durch Ungenauigkeiten bei Gleitkommaarithmetik.

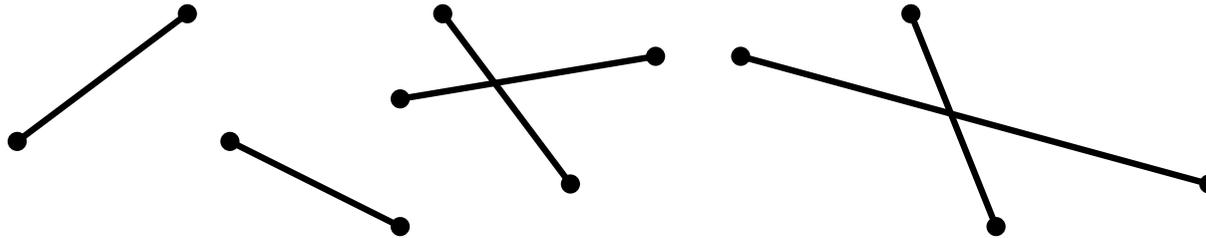
Schnitte von Strecken – Sweep-Line

Problem: Schnitte von Strecken

Gegeben seien n Strecken. Schneiden sich zwei dieser Strecken?

Einfache Lösung:

Teste für jedes Streckenpaar ob sie sich schneiden. $\rightarrow \binom{n}{2} \in O(n^2)$ Paare.

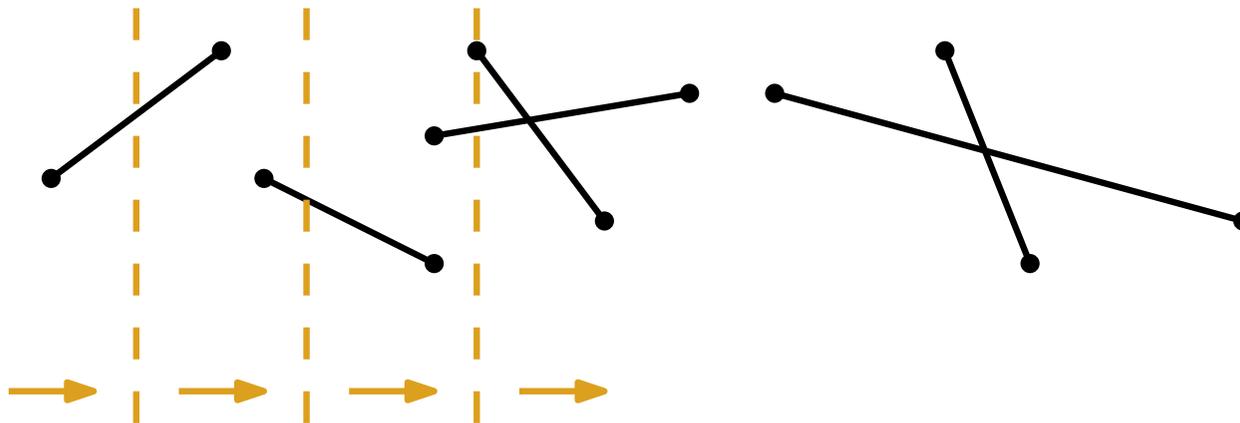


Problem: Schnitte von Strecken

Gegeben seien n Strecken. Schneiden sich zwei dieser Strecken?

Einfache Lösung:

Teste für jedes Streckenpaar ob sie sich schneiden. $\rightarrow \binom{n}{2} \in O(n^2)$ Paare.



Heute:

Sweep-Line-Algorithmus mit Laufzeit $O(n \log n)$.

Definition: Sweep-Line

(Definition 6.6)

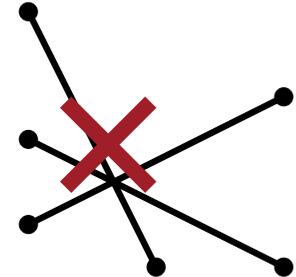
Eine *Sweep-Line* ist eine imaginäre vertikale Linie, die eine Menge geometrischer Objekte von links nach rechts passiert.

Eine Sweep-Line ist durch ihre x -Koordinate eindeutig bestimmt.

Vergleichbare Strecken

Wir machen folgende Annahmen:

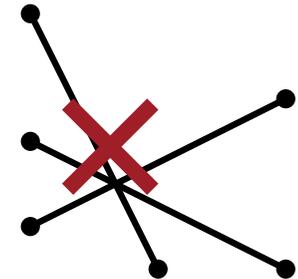
- Keine der Strecken ist vertikal.
- In jedem Punkt schneiden sich höchstens zwei Strecken.



Vergleichbare Strecken

Wir machen folgende Annahmen:

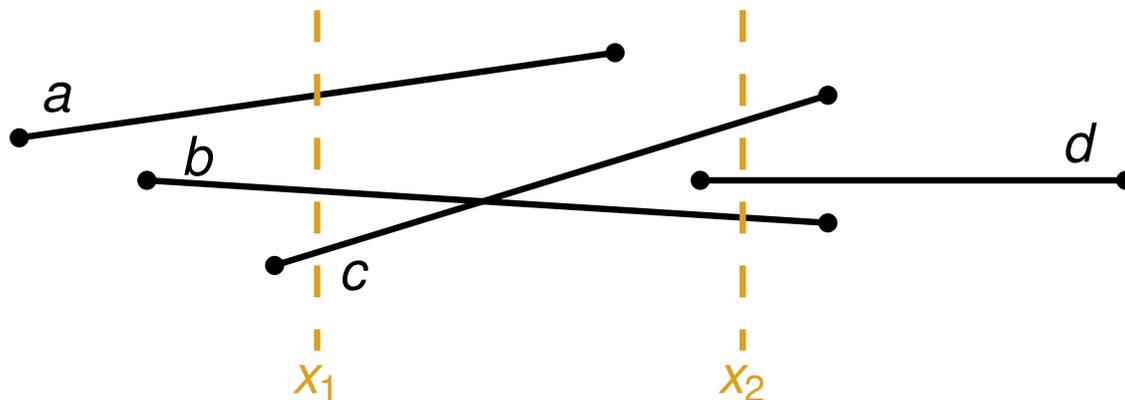
- Keine der Strecken ist vertikal.
- In jedem Punkt schneiden sich höchstens zwei Strecken.



Definition: Vergleichbare Strecken

Zwei Strecken s_1 und s_2 sind bezüglich Sweep-Line x *vergleichbar*, falls beide x schneiden und sich nicht gegenseitig an der Stelle x schneiden.

Es gilt dann: $s_1 <_x s_2$ ($s_2 <_x s_1$) falls s_1 in x unterhalb (*oberhalb*) von s_2 liegt.



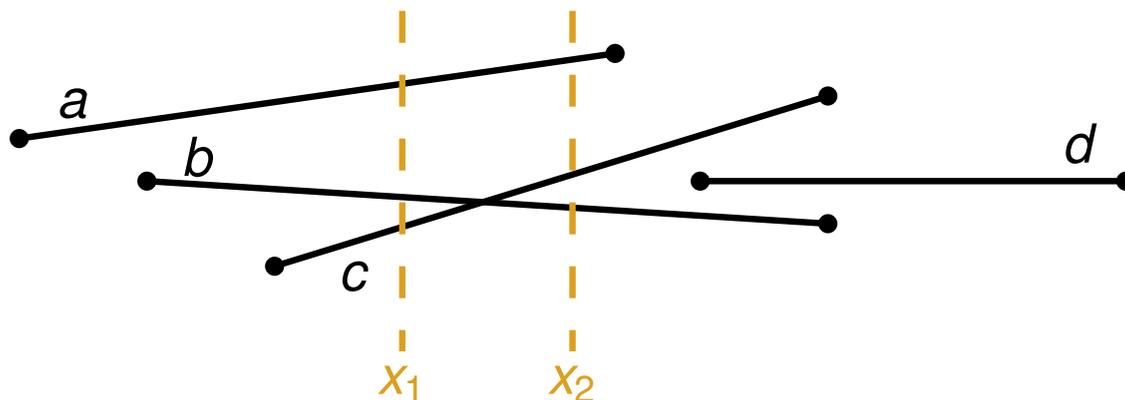
$$c <_{x_1} b <_{x_1} a$$

$$b <_{x_2} d <_{x_2} c$$

Für jedes x ist $<_x$ eine totale Ordnung auf den Strecken, die x schneiden und sich nicht bei x gegenseitig schneiden.

Beobachtung: Sich schneidende Strecken

- Für zwei Strecken s_1, s_2 die sich schneiden gibt es Sweep-Lines x_1 und x_2 , sodass $s_1 <_{x_1} s_2$ und $s_2 <_{x_2} s_1$.
- x_1 und x_2 können sogar so gewählt werden, dass s_1 und s_2 bezgl. $<_{x_1}$ und $<_{x_2}$ direkt aufeinanderfolgen.



$$c <_{x_1} b <_{x_1} a$$

$$b <_{x_2} c <_{x_2} a$$

Folgerung

Es ist ausreichend, alle Streckenpaare auf Schnitt zu prüfen, die bezüglich $<_x$ irgendeiner Sweep-Line x direkt aufeinanderfolgen.

Sweep-Line Algorithmen verwenden typischerweise zwei Datenmengen:

1. **Sweep-Line Zustand:** Objekte (und ggf. Beziehungen zwischen ihnen), die von der aktuellen Sweep-Line geschnitten werden.
2. **Event-Point Schedule:** Folge sortierter x -Koordinaten; die „Haltepunkte“ (Event-Points) der Sweep-Line. Der Sweep-Line Zustand ändert sich nur an solchen Event-Points.

Sweep-Line Algorithmen verwenden typischerweise zwei Datenmengen:

1. **Sweep-Line Zustand:** Objekte (und ggf. Beziehungen zwischen ihnen), die von der aktuellen Sweep-Line geschnitten werden.
2. **Event-Point Schedule:** Folge sortierter x -Koordinaten; die „Haltepunkte“ (Event-Points) der Sweep-Line. Der Sweep-Line Zustand ändert sich nur an solchen Event-Points.

Für unser Problem:

1. **Sweep-Line Zustand:** Alle Strecken, die die aktuelle Sweep-Line schneiden, von unten nach oben sortiert.
2. **Event-Point Schedule:** Die Endpunkte aller Strecken, von links nach rechts sortiert.

Events: An jedem Event-Point taucht eine neue Strecke auf oder eine schon vorhandene verschwindet. Folgendes muss getan werden:

- Neue Strecke in den Sweep-Line Zustand einfügen bzw. alte herauslöschen.
- Überprüfen ob sich neuerdings benachbarte Streckenpaare schneiden.

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$

Sweep-Line Zustand

Sortiere Endpunkte der Strecken von links nach rechts **Event-Point Schedule**

if zwei Endpunkte sind gleich **then** gib TRUE zurück

foreach Endpunkt p in der sortierten Liste **do**

if p linker Endpunkt von Strecke s **then**

 INSERT(T, s)

if s schneidet ABOVE(T, s) **then** gib TRUE zurück

if s schneidet BELOW(T, s) **then** gib TRUE zurück

if p rechter Endpunkt von Strecke s **then**

if ABOVE(T, s) schneidet BELOW(T, s) **then** gib TRUE zurück

 DELETE(T, s)

gib FALSE zurück

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$	Sweep-Line Zustand	$O(1)$
--------------------------	--------------------	--------

Sortiere Endpunkte der Strecken von links nach rechts	Event-Point Schedule	$O(n \log n)$
---	----------------------	---------------

if zwei Endpunkte sind gleich then gib TRUE zurück		$O(n)$
--	--	--------

foreach Endpunkt p in der sortierten Liste **do**

if p linker Endpunkt von Strecke s **then**

 INSERT(T, s)

if s schneidet ABOVE(T, s) **then** gib TRUE zurück

if s schneidet BELOW(T, s) **then** gib TRUE zurück

if p rechter Endpunkt von Strecke s **then**

if ABOVE(T, s) schneidet BELOW(T, s) **then** gib TRUE zurück

 DELETE(T, s)

gib FALSE zurück

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$	Sweep-Line Zustand	$O(1)$
--------------------------	--------------------	--------

Sortiere Endpunkte der Strecken von links nach rechts	Event-Point Schedule	$O(n \log n)$
---	----------------------	---------------

if zwei Endpunkte sind gleich then gib TRUE zurück		$O(n)$
--	--	--------

foreach Endpunkt p in der sortierten Liste **do**

if p linker Endpunkt von Strecke s **then**

 INSERT(T, s)

if s schneidet ABOVE(T, s) **then** gib TRUE zurück

if s schneidet BELOW(T, s) **then** gib TRUE zurück

if p rechter Endpunkt von Strecke s **then**

if ABOVE(T, s) schneidet BELOW(T, s) **then** gib TRUE zurück

 DELETE(T, s)

gib FALSE zurück

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Benutze für T einen AVL-Baum (oder ähnliches): Operationen INSERT(T, s), DELETE(T, s), ABOVE(T, s) und BELOW(T, s) benötigen $O(\log n)$ Zeit.

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$ Sweep-Line Zustand $O(1)$

Sortiere Endpunkte der Strecken von links nach rechts Event-Point Schedule $O(n \log n)$

if zwei Endpunkte sind gleich **then** gib TRUE zurück $O(n)$

foreach Endpunkt p in der sortierten Liste **do**

if p linker Endpunkt von Strecke s **then** $O(\log n)$

INSERT(T, s)

if s schneidet ABOVE(T, s) **then** gib TRUE zurück

if s schneidet BELOW(T, s) **then** gib TRUE zurück

if p rechter Endpunkt von Strecke s **then** $O(\log n)$

if ABOVE(T, s) schneidet BELOW(T, s) **then** gib TRUE zurück

DELETE(T, s)

gib FALSE zurück

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Benutze für T einen AVL-Baum (oder ähnliches): Operationen INSERT(T, s), DELETE(T, s), ABOVE(T, s) und BELOW(T, s) benötigen $O(\log n)$ Zeit.

Sweep-Line Algorithmus

INTERSECT(S)

$T \leftarrow \emptyset$	Sweep-Line Zustand	$O(1)$
Sortiere Endpunkte der Strecken von links nach rechts	Event-Point Schedule	$O(n \log n)$
if zwei Endpunkte sind gleich then gib TRUE zurück		$O(n)$
foreach Endpunkt p in der sortierten Liste do		$O(n \log n)$
if p linker Endpunkt von Strecke s then		$O(\log n)$
INSERT(T, s)		
if s schneidet ABOVE(T, s) then gib TRUE zurück		
if s schneidet BELOW(T, s) then gib TRUE zurück		
if p rechter Endpunkt von Strecke s then		$O(\log n)$
if ABOVE(T, s) schneidet BELOW(T, s) then gib TRUE zurück		
DELETE(T, s)		
gib FALSE zurück		$O(1)$

(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Benutze für T einen AVL-Baum (oder ähnliches): Operationen INSERT(T, s), DELETE(T, s), ABOVE(T, s) und BELOW(T, s) benötigen $O(\log n)$ Zeit.

Sweep-Line Algorithmus

INTERSECT(S)	$O(n \log n)$
$T \leftarrow \emptyset$	Sweep-Line Zustand $O(1)$
Sortiere Endpunkte der Strecken von links nach rechts	Event-Point Schedule $O(n \log n)$
if zwei Endpunkte sind gleich then gib TRUE zurück	$O(n)$
foreach Endpunkt p in der sortierten Liste do	$O(n \log n)$
if p linker Endpunkt von Strecke s then	$O(\log n)$
INSERT(T, s)	
if s schneidet ABOVE(T, s) then gib TRUE zurück	
if s schneidet BELOW(T, s) then gib TRUE zurück	
if p rechter Endpunkt von Strecke s then	$O(\log n)$
if ABOVE(T, s) schneidet BELOW(T, s) then gib TRUE zurück	
DELETE(T, s)	
gib FALSE zurück	$O(1)$

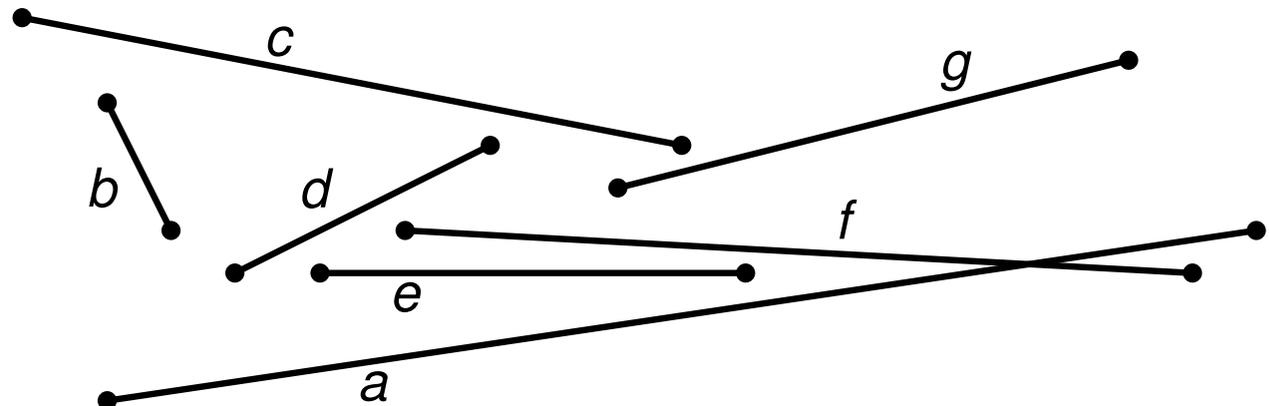
(ABOVE(T, s) (BELOW(T, s)) ist die Strecke unmittelbar über (unter) s in T)

Benutze für T einen AVL-Baum (oder ähnliches): Operationen INSERT(T, s), DELETE(T, s), ABOVE(T, s) und BELOW(T, s) benötigen $O(\log n)$ Zeit.

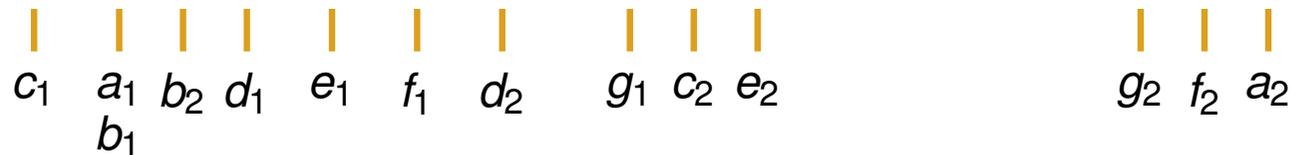
⇒ Gesamtlaufzeit von $O(n \log n)$

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand



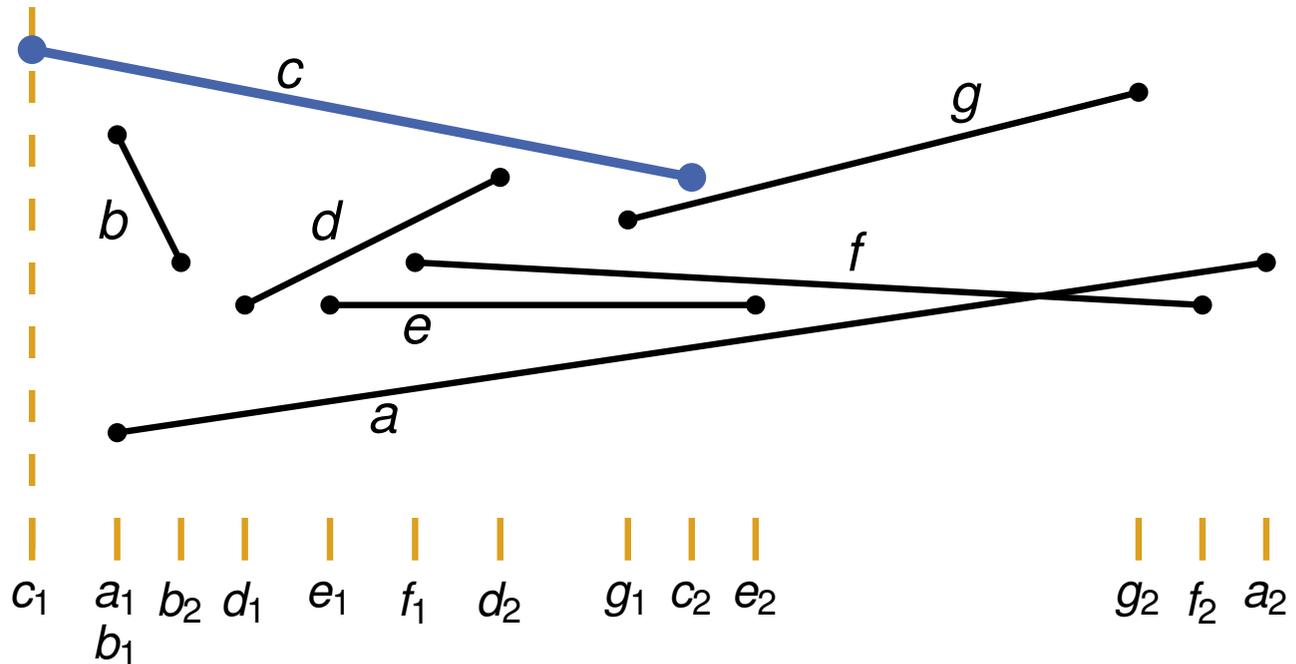
Event-Point Schedule →



Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

↓
+ c



Sweep-Line Algorithmus – Beispiel

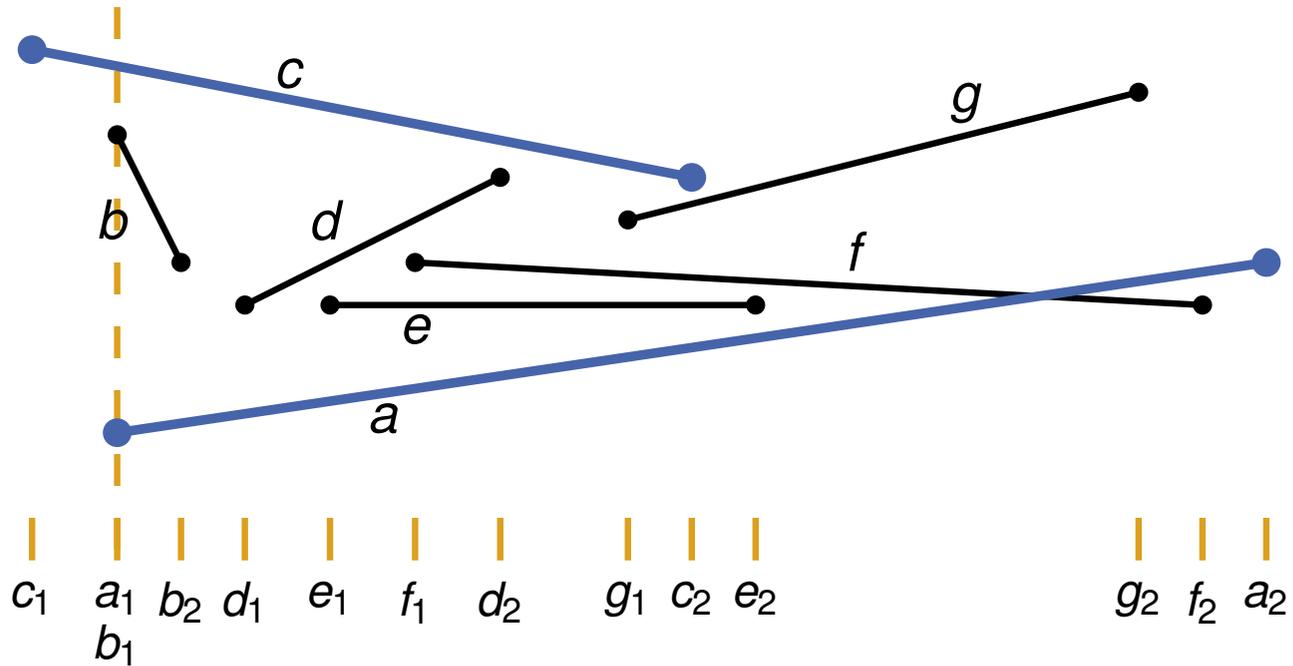
Sweep-Line Zustand



c

+ a

Event-Point Schedule →



a schneidet c nicht

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

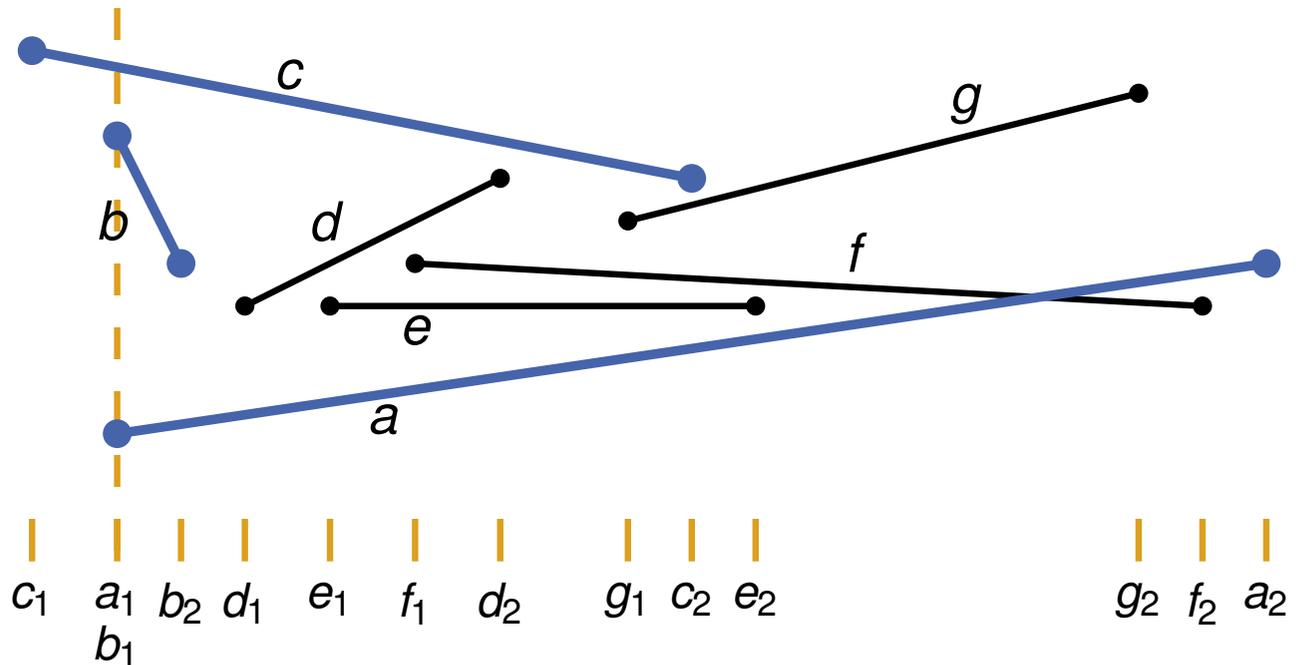


c

+ b

a

Event-Point Schedule →



b schneidet weder a noch c

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

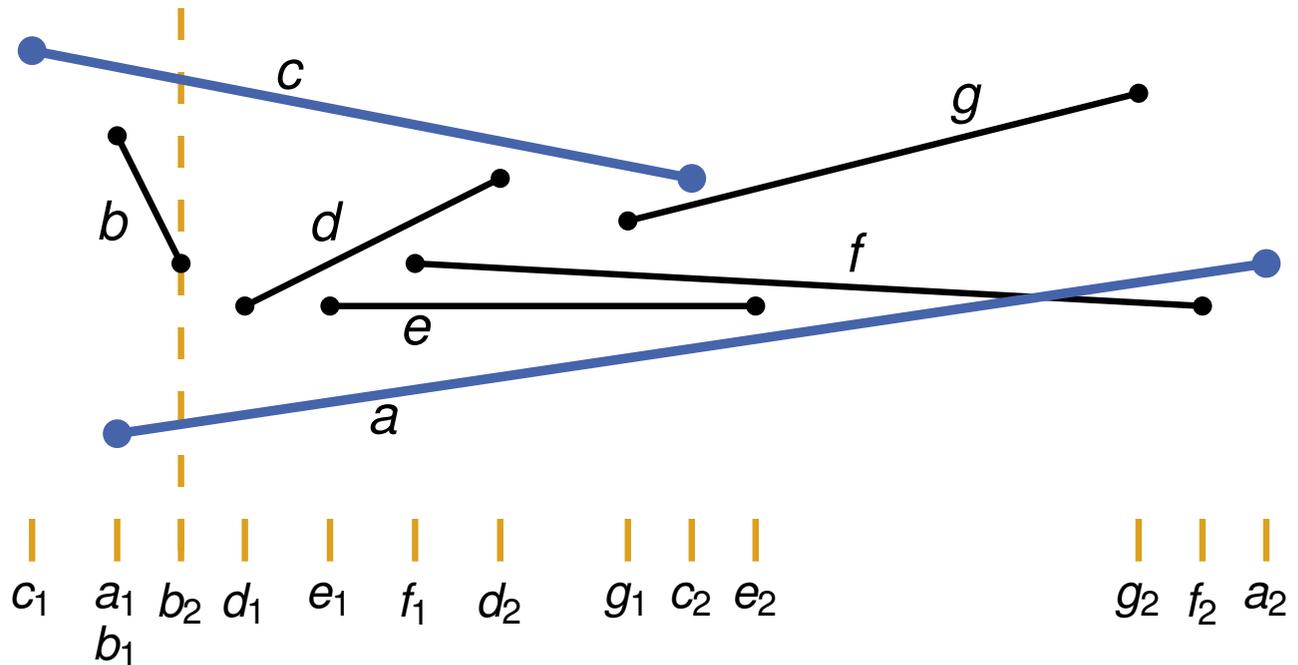


c

~~*b*~~

a

Event-Point Schedule →



a schneidet *c* nicht

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

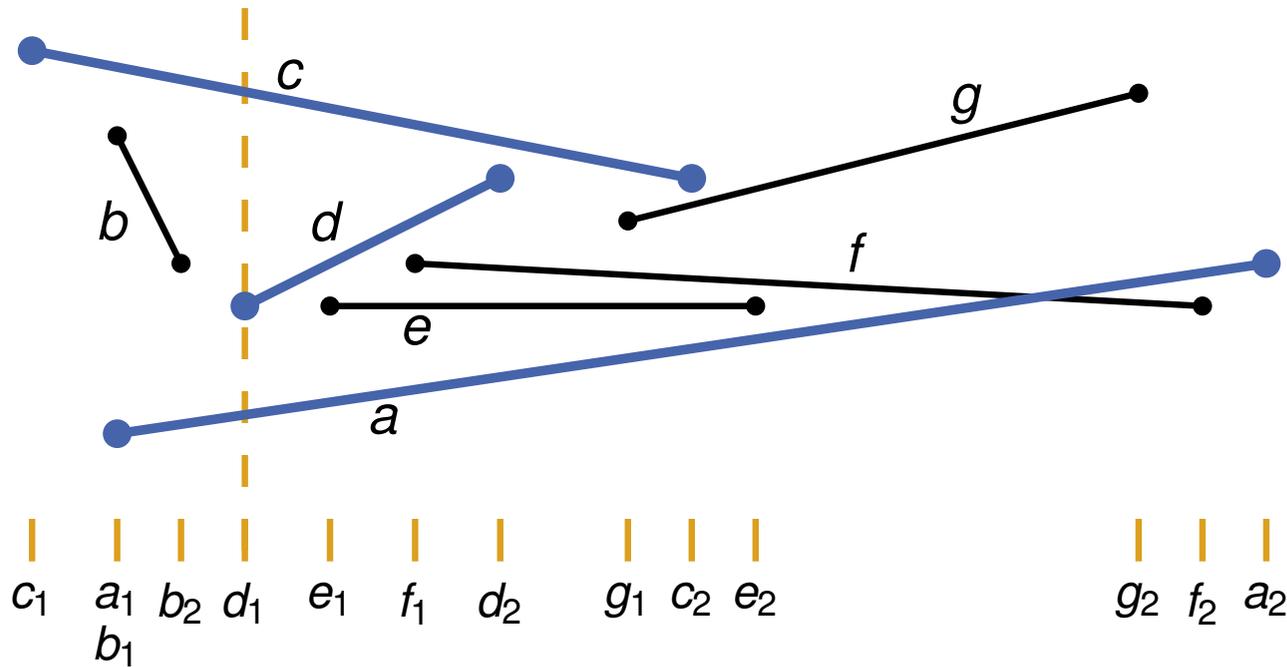


c

+ *d*

a

Event-Point Schedule →



d schneidet weder *a* noch *c*

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand



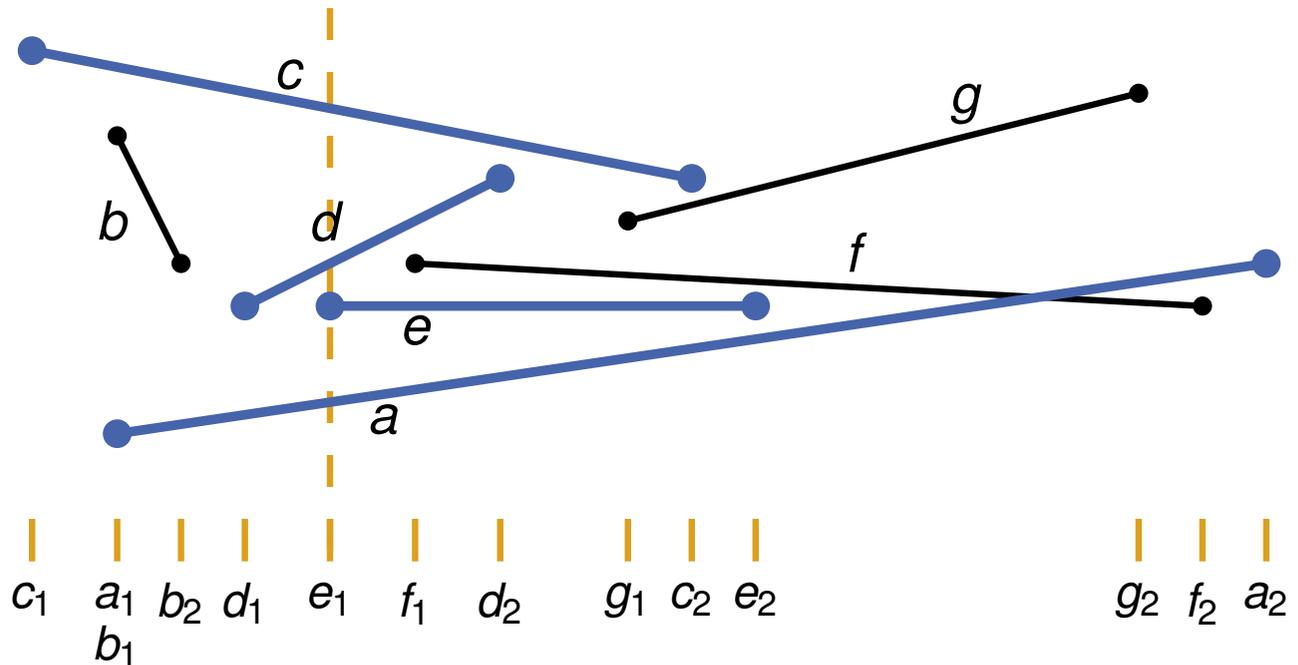
c

d

+ *e*

a

Event-Point Schedule →



e schneidet weder *d* noch *a*

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand



c

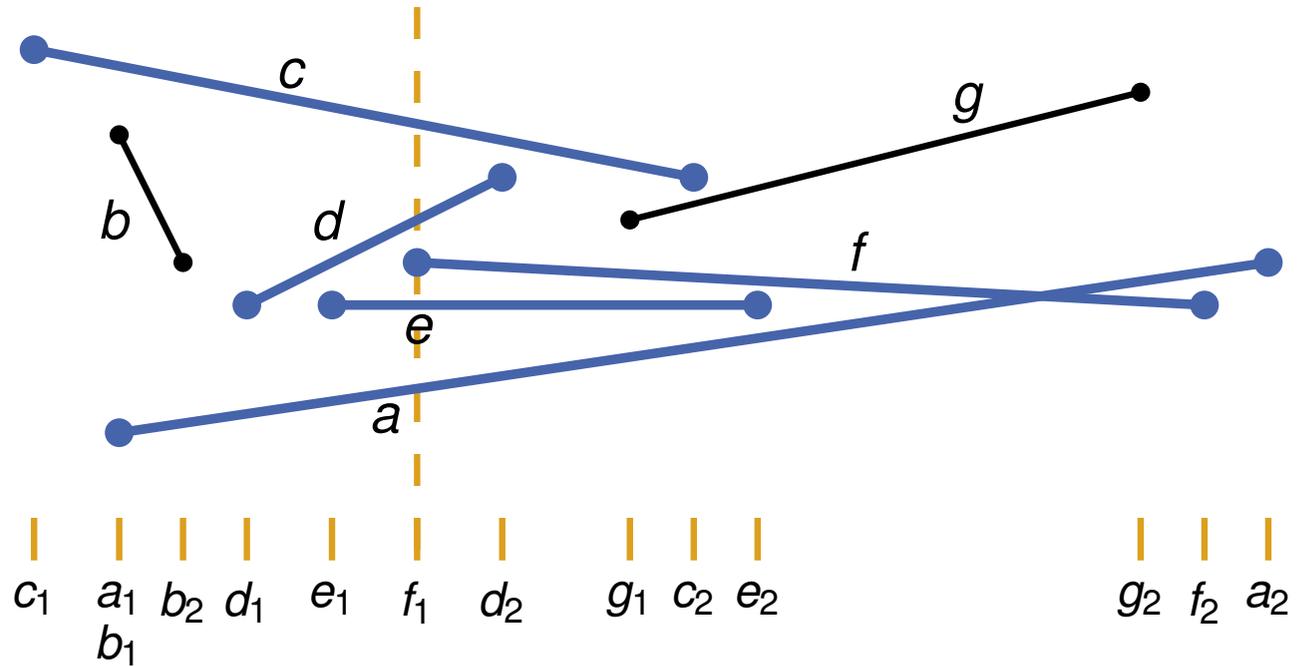
d

+ *f*

e

a

Event-Point Schedule →



f schneidet weder *d* noch *e*

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand



c

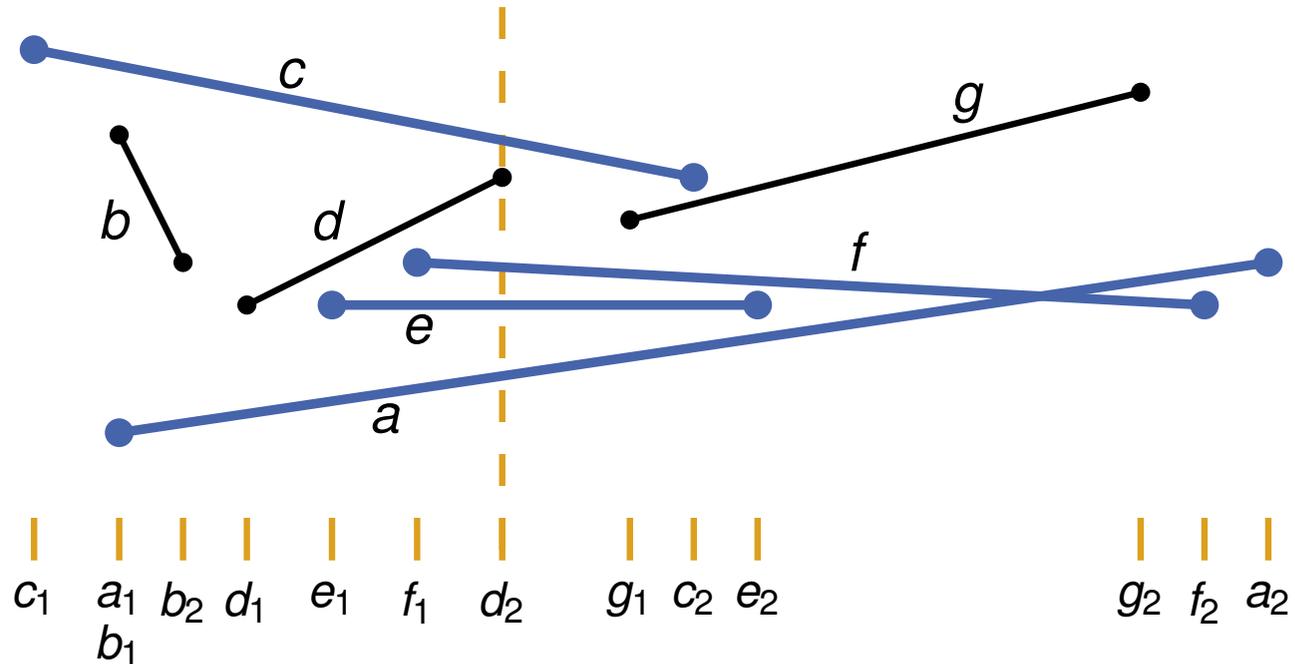
~~*d*~~

f

e

a

Event-Point Schedule →

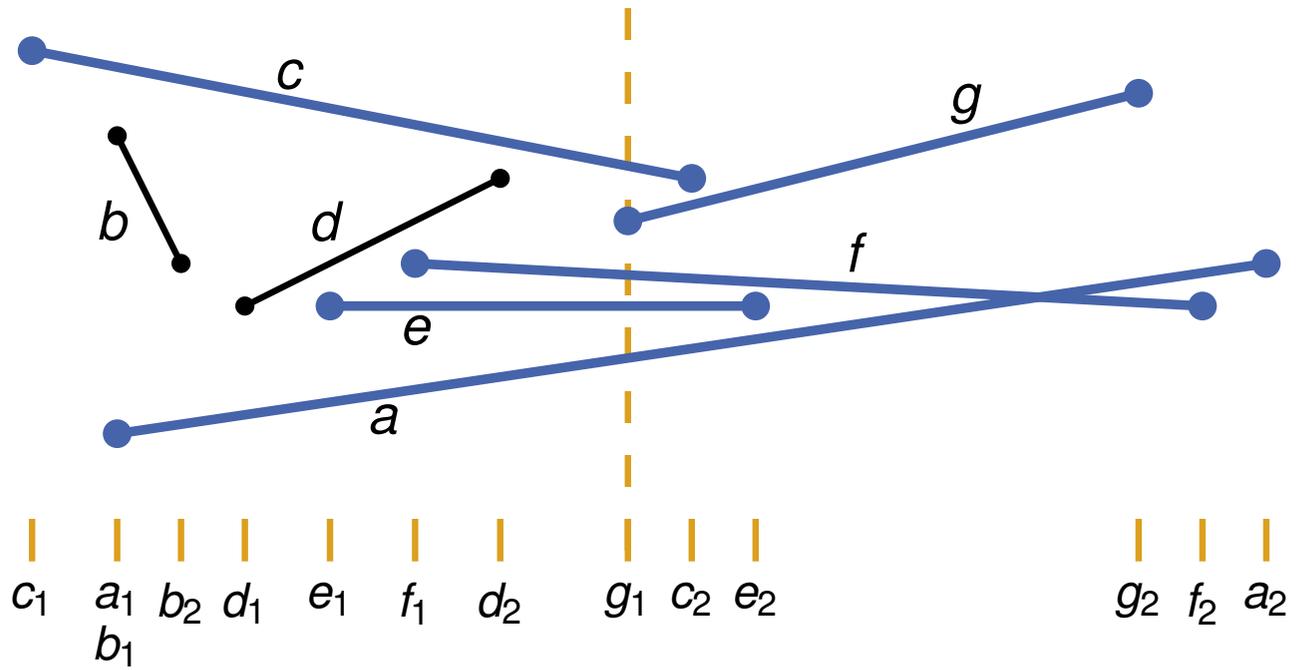


f schneidet *c* nicht

Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

↓
c
 + *g*
f
e
a



Event-Point Schedule →

g schneidet weder *f* noch *c*

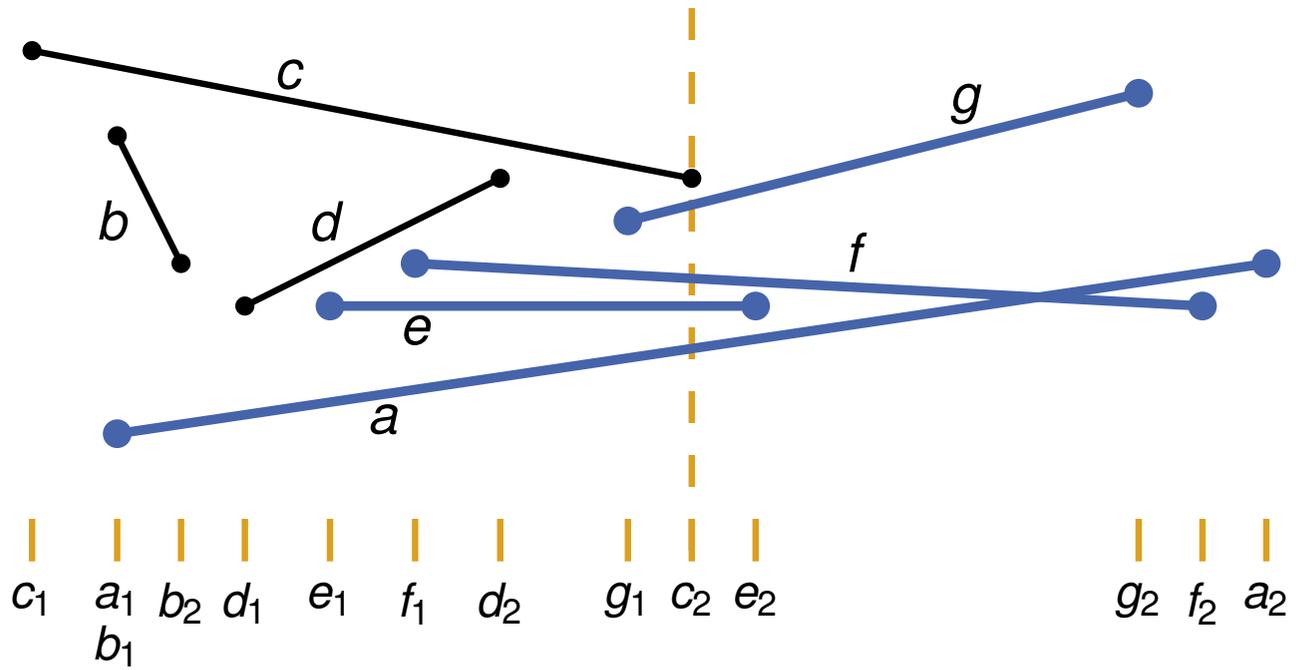
Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand



~~c~~
g
f
e
a

Event-Point Schedule →



Sweep-Line Algorithmus – Beispiel

Sweep-Line Zustand

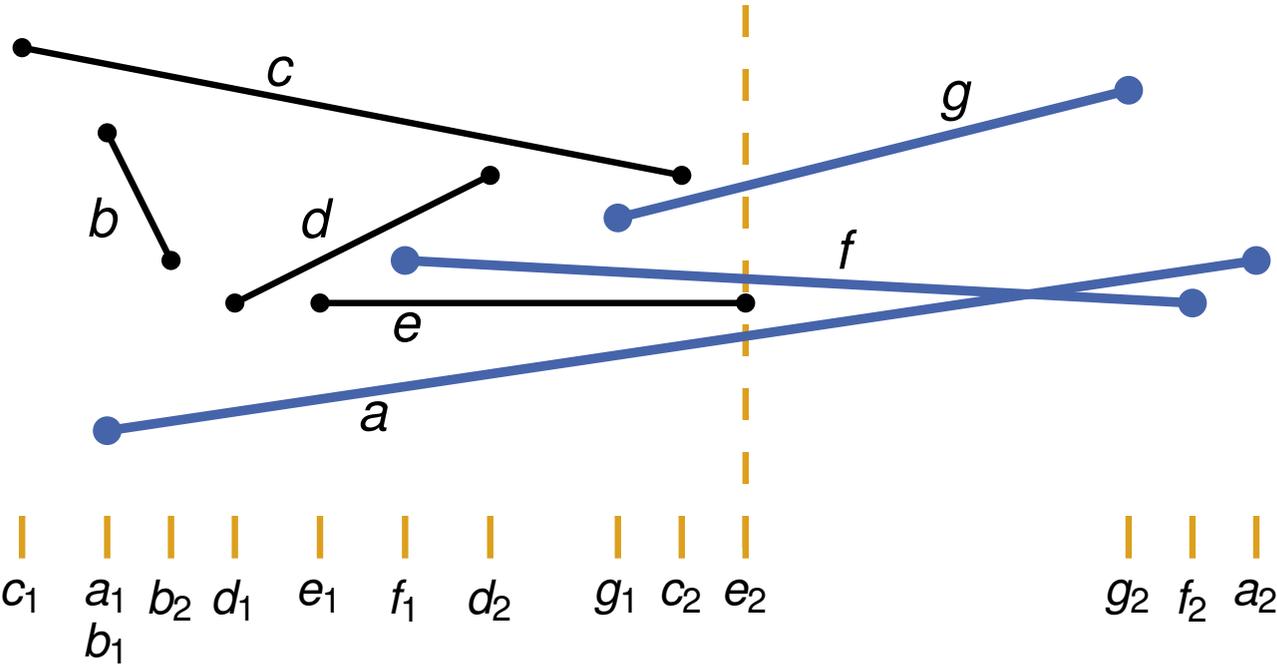


g

f

~~*e*~~

a



Event-Point Schedule →

f schneidet *a*! ⇒ gib TRUE zurück

- Der vorgestellte Algorithmus prüft nur, ob es mindestens eine Kreuzungen gibt und kann die erste gefundene Kreuzung ausgeben.
- Man kann den Algorithmus leicht anpassen um alle Kreuzungen zu finden. Man benötigt dann $O(n \log n + k \log n)$ Zeit, wobei k die Anzahl an Kreuzungen ist.
Beachte: $k \in O(n^2)$.
- Es gibt auch einen Algorithmus mit Laufzeit $O(n \log n + k)$, der alle Kreuzungen findet.