Karlsruher Institut für Technologie

Algorithmische Geometrie

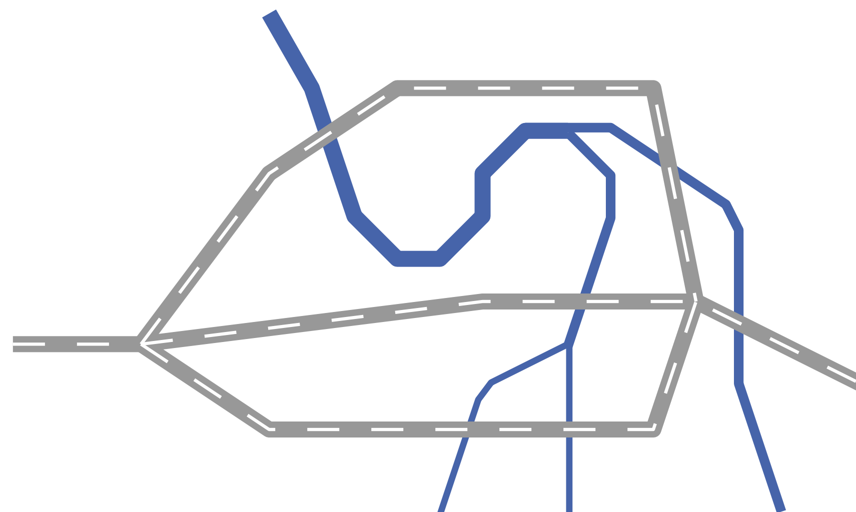
Linienschnitt

Linienschnitt: Motivation

Karlsruher Institut für Technologie

Wo sind Brücken?

- **gegeben:** Straßen und Flüsse
  (jeweils als Menge von Strecken)
- **Ziel:** finde alle Brücken

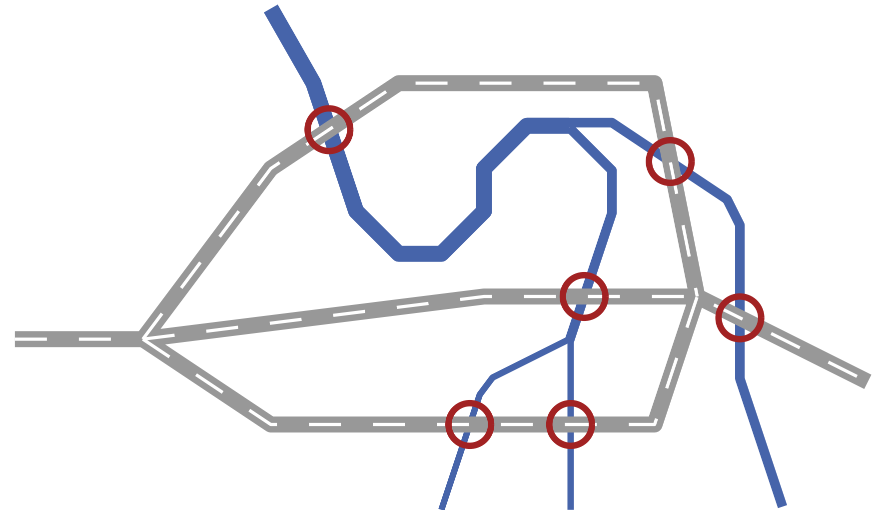


Linienschnitt: Motivation

Karlsruher Institut für Technologie

Wo sind Brücken?

- **gegeben:** Straßen und Flüsse
  (jeweils als Menge von Strecken)
- **Ziel:** finde alle Brücken

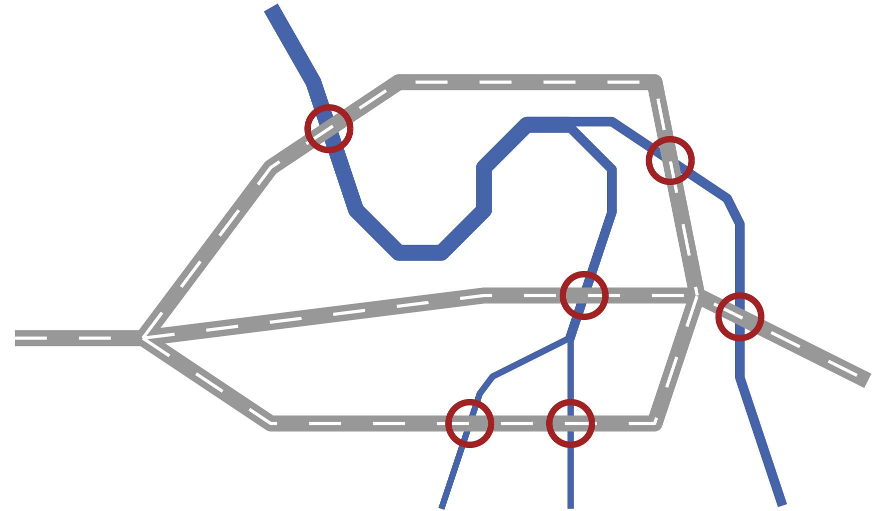


Linienschnitt: Motivation

Karlsruher Institut für Technologie

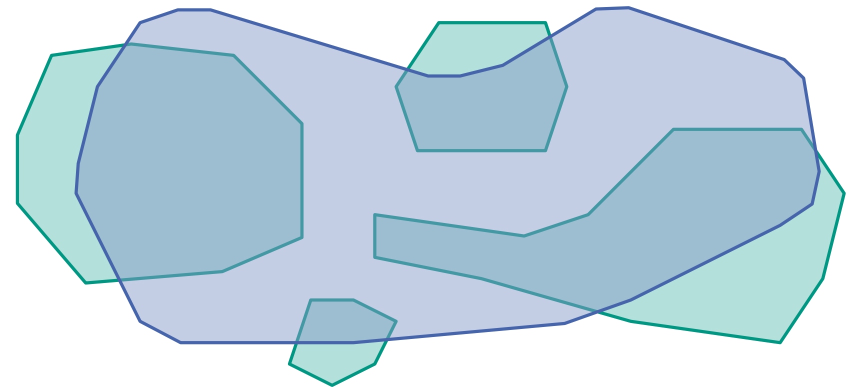
Wo sind Brücken?

- **gegeben:** Straßen und Flüsse
  (jeweils als Menge von Strecken)
- **Ziel:** finde alle Brücken



Tannenwälder mit viel Niederschlag

- **gegeben:** Tannenwälder und Regionen mit mehr als 1500mm Niederschlag
  (jeweils als Polygone)
- **Ziel:** berechne den Schnitt der beiden

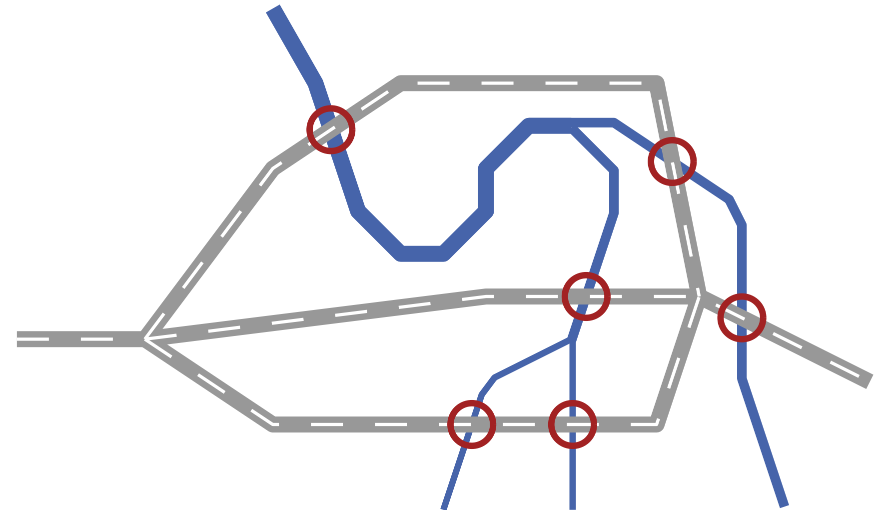


Linienschnitt: Motivation

Karlsruher Institut für Technologie

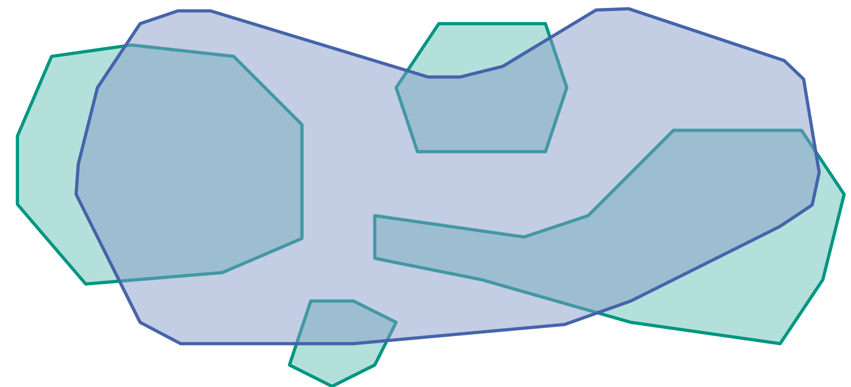
Wo sind Brücken?

- **gegeben:** Straßen und Flüsse
  (jeweils als Menge von Strecken)
- **Ziel:** finde alle Brücken



Tannenwälder mit viel Niederschlag

- **gegeben:** Tannenwälder und Regionen mit mehr als 1500mm Niederschlag
  (jeweils als Polygone)
- **Ziel:** berechne den Schnitt der beiden



Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Erste Beobachtungen

Karlsruher Institut für Technologie

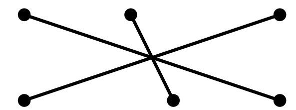
Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Annahme: allgemeine Lage

Erste Beobachtungen

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Annahme: allgemeine Lage



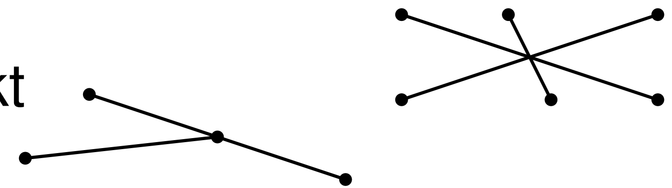
- maximal zwei Strecken schneiden sich pro Punkt

Erste Beobachtungen

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Annahme: allgemeine Lage

- maximal zwei Strecken schneiden sich pro Punkt
- kein Endpunkt auf einer anderen Strecke

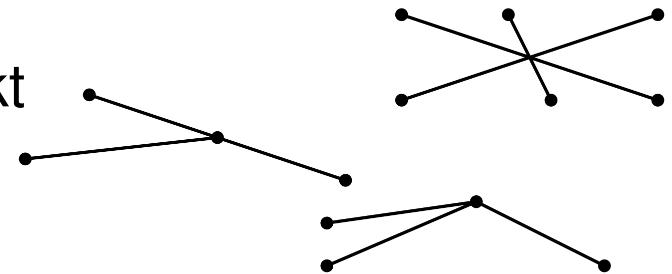


Erste Beobachtungen

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Annahme: allgemeine Lage

- maximal zwei Strecken schneiden sich pro Punkt
- kein Endpunkt auf einer anderen Strecke
- keine übereinstimmenden Endpunkte



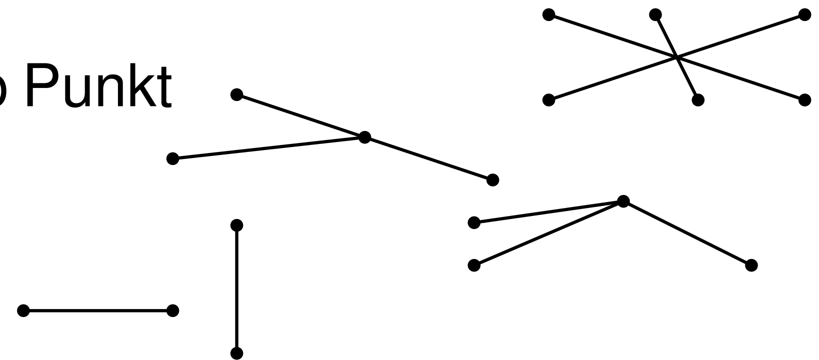
Erste Beobachtungen

Karlsruher Institut für Technologie

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Annahme: allgemeine Lage

- maximal zwei Strecken schneiden sich pro Punkt
- kein Endpunkt auf einer anderen Strecke
- keine übereinstimmenden Endpunkte
- keine horizontalen/vertikalen Strecken



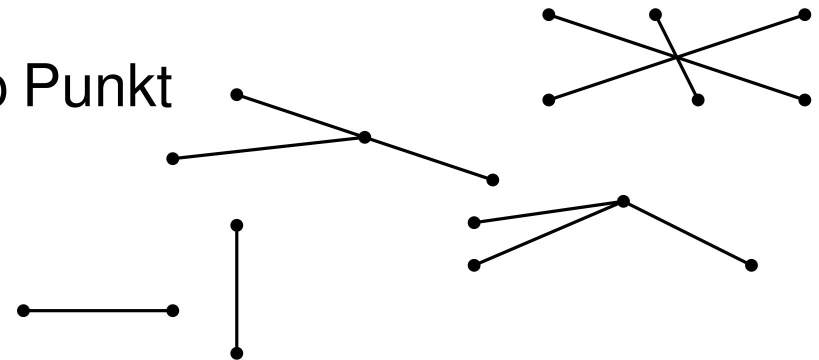
Erste Beobachtungen

Karlsruher Institut für Technologie

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Annahme: allgemeine Lage

- maximal zwei Strecken schneiden sich pro Punkt
- kein Endpunkt auf einer anderen Strecke
- keine übereinstimmenden Endpunkte
- keine horizontalen/vertikalen Strecken



Trivialer Algorithmus

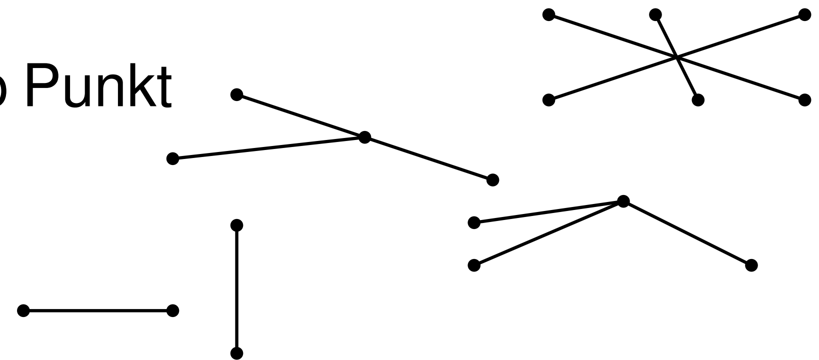
Erste Beobachtungen

Karlsruher Institut für Technologie

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Annahme: allgemeine Lage

- maximal zwei Strecken schneiden sich pro Punkt
- kein Endpunkt auf einer anderen Strecke
- keine übereinstimmenden Endpunkte
- keine horizontalen/vertikalen Strecken

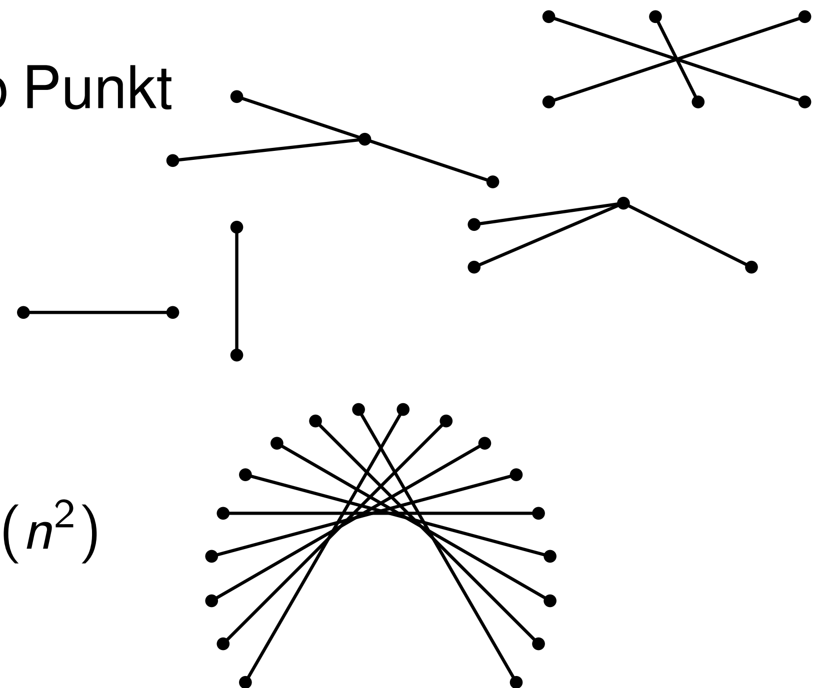


Trivialer Algorithmus

- teste jedes Streckenpaar auf Schnitt $\rightarrow O(n^2)$

Erste Beobachtungen

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.



Annahme: allgemeine Lage

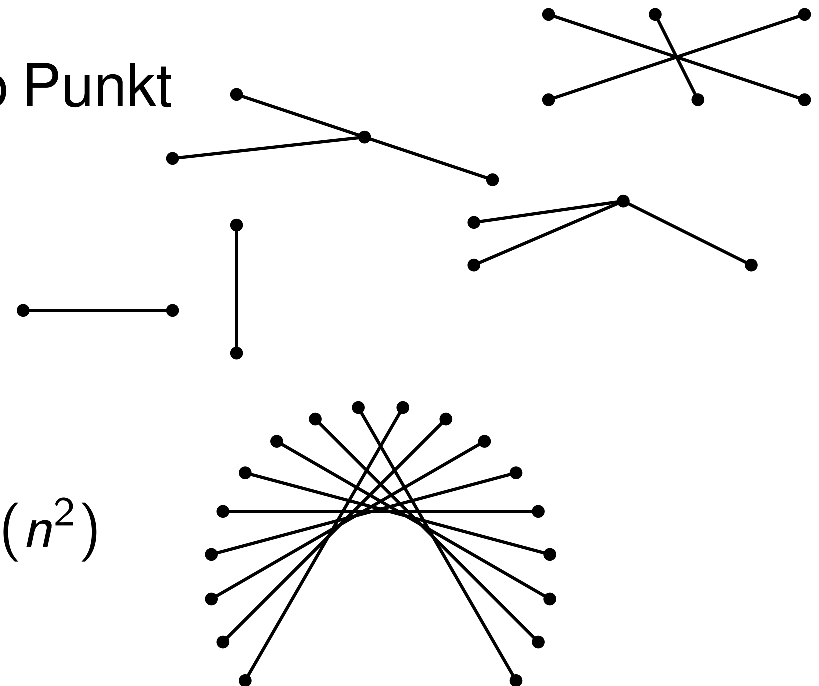
- maximal zwei Strecken schneiden sich pro Punkt
- kein Endpunkt auf einer anderen Strecke
- keine übereinstimmenden Endpunkte
- keine horizontalen/vertikalen Strecken

Trivialer Algorithmus

- teste jedes Streckenpaar auf Schnitt $\rightarrow O(n^2)$
- besser geht es nicht

Erste Beobachtungen

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.



Annahme: allgemeine Lage

- maximal zwei Strecken schneiden sich pro Punkt
- kein Endpunkt auf einer anderen Strecke
- keine übereinstimmenden Endpunkte
- keine horizontalen/vertikalen Strecken

Trivialer Algorithmus

- teste jedes Streckenpaar auf Schnitt $\rightarrow O(n^2)$
- besser geht es nicht

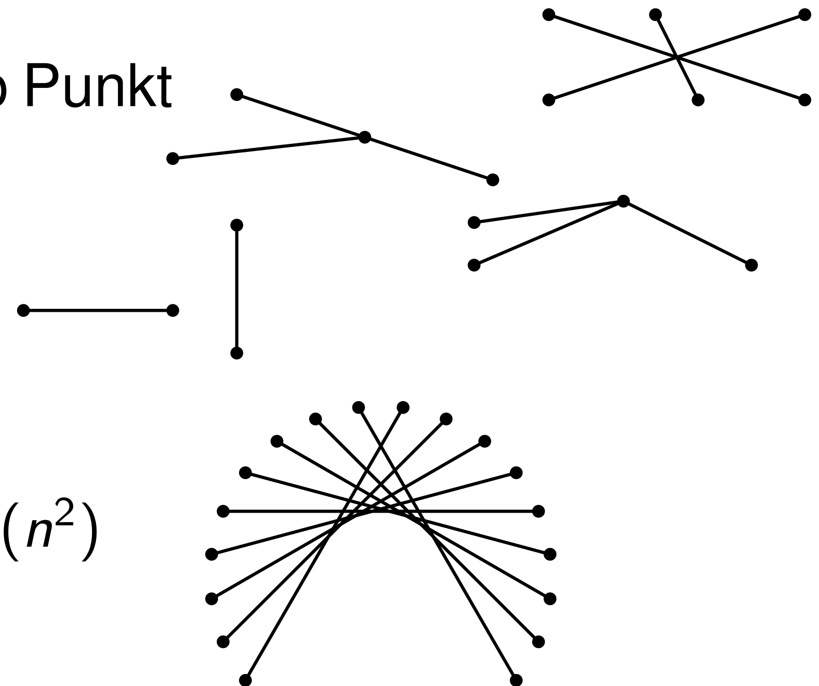
Mögliche Verbesserung (trotz unterer Schranke)

Erste Beobachtungen

Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Annahme: allgemeine Lage

- maximal zwei Strecken schneiden sich pro Punkt
- kein Endpunkt auf einer anderen Strecke
- keine übereinstimmenden Endpunkte
- keine horizontalen/vertikalen Strecken



Trivialer Algorithmus

- teste jedes Streckenpaar auf Schnitt $\rightarrow O(n^2)$
- besser geht es nicht

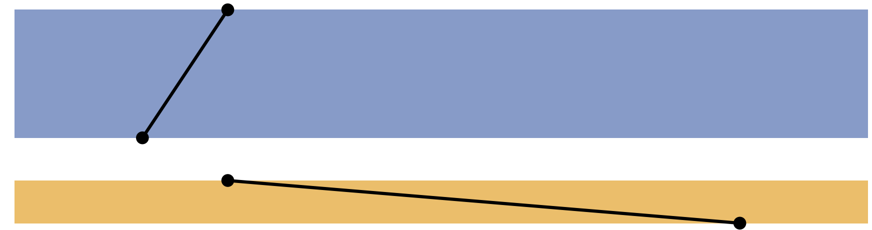
Mögliche Verbesserung (trotz unterer Schranke)

- Ziel: wenige Schnittpunkte $\Rightarrow$ bessere Laufzeit
- Laufzeit abhängig von Ausgabegröße $\rightarrow$ ausgabesensitiver Algorithmus

Ein einfacher Sweep-Line Algorithmus

Idee

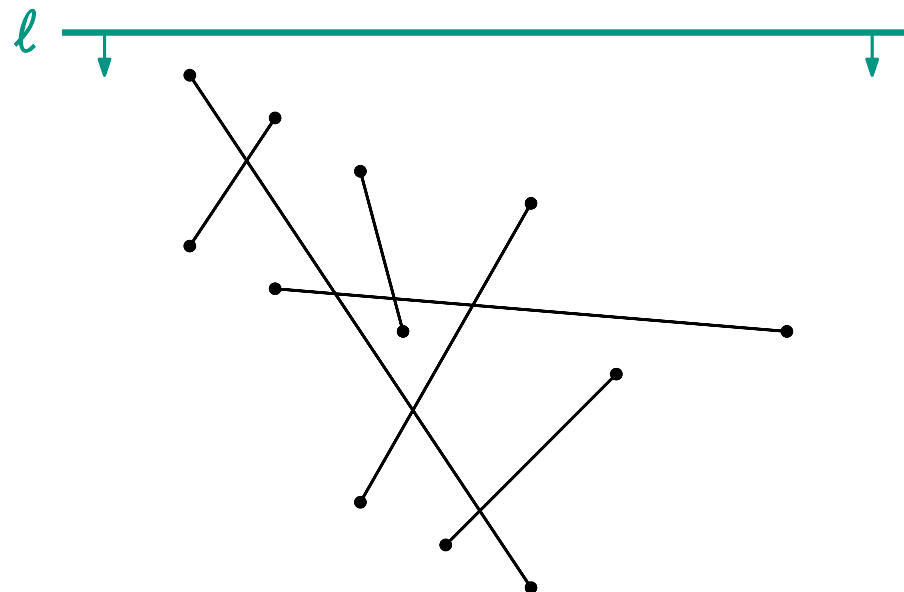
- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt



Ein einfacher Sweep-Line Algorithmus

Idee

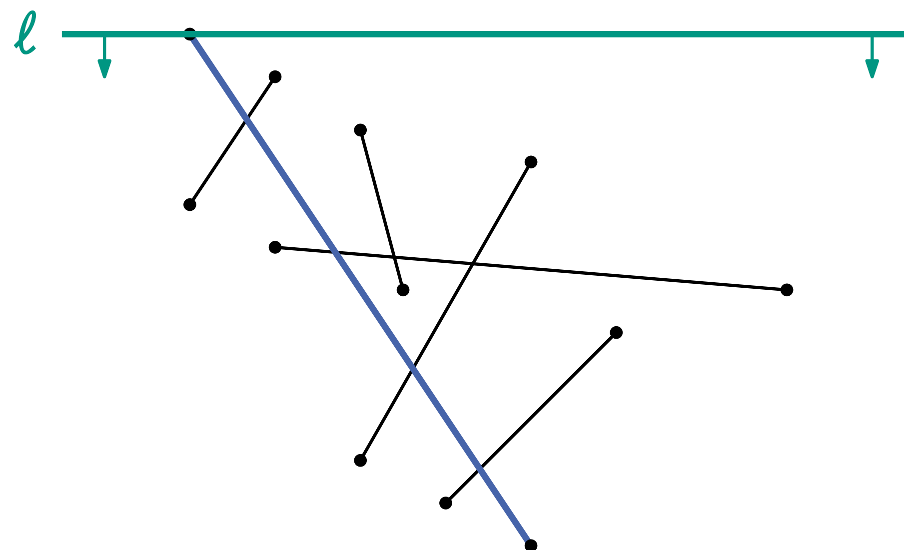
- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Ein einfacher Sweep-Line Algorithmus

Idee

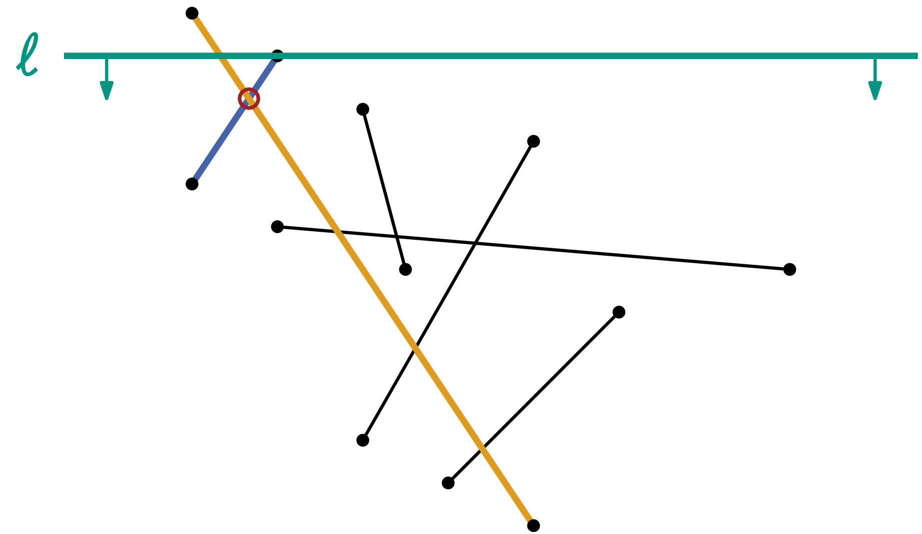
- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Ein einfacher Sweep-Line Algorithmus

Idee

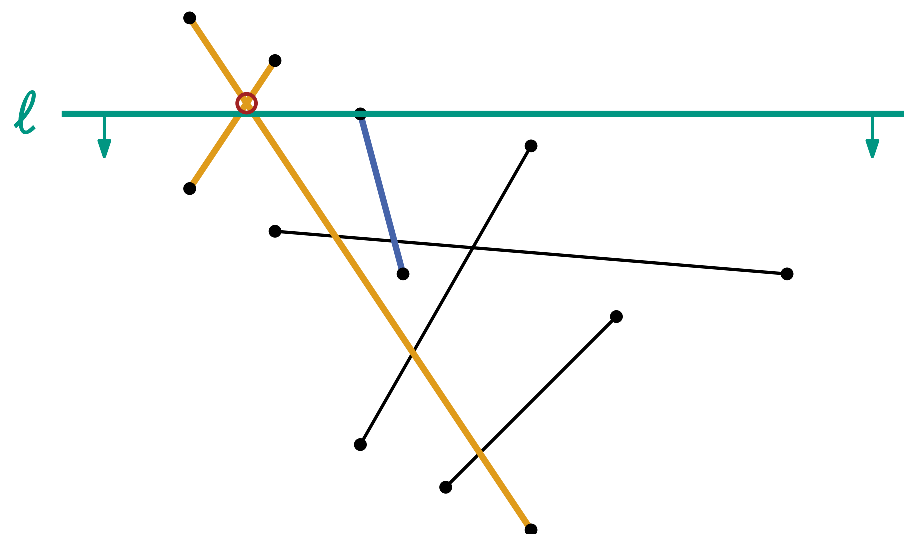
- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Ein einfacher Sweep-Line Algorithmus

Idee

- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken

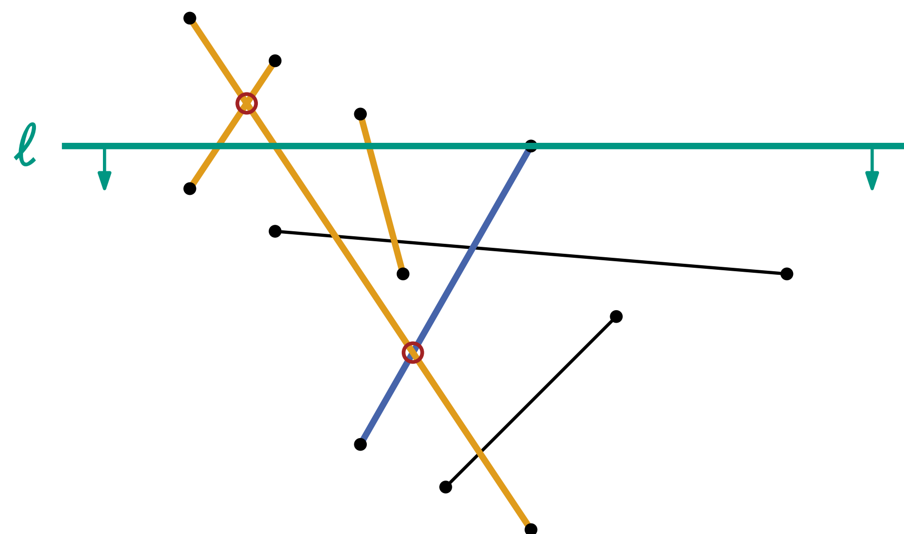


Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Idee

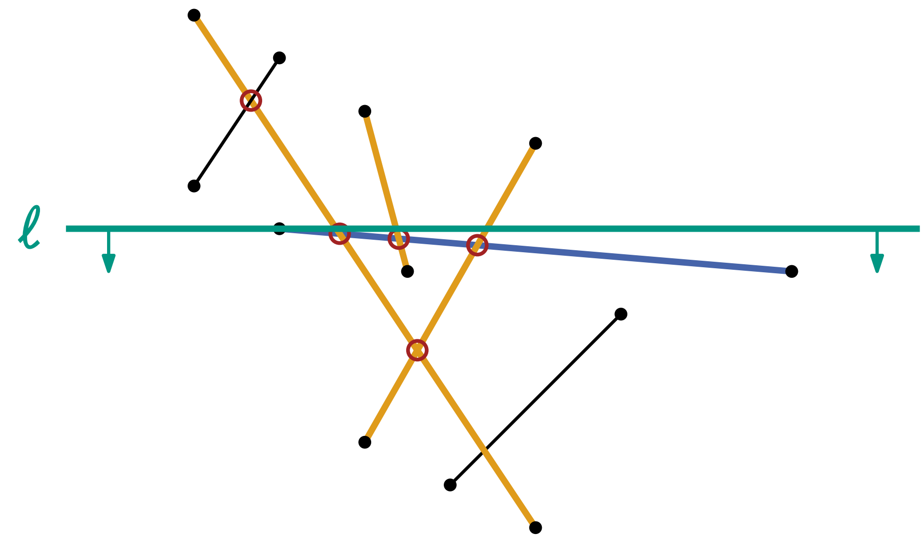
- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Ein einfacher Sweep-Line Algorithmus

Idee

- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken

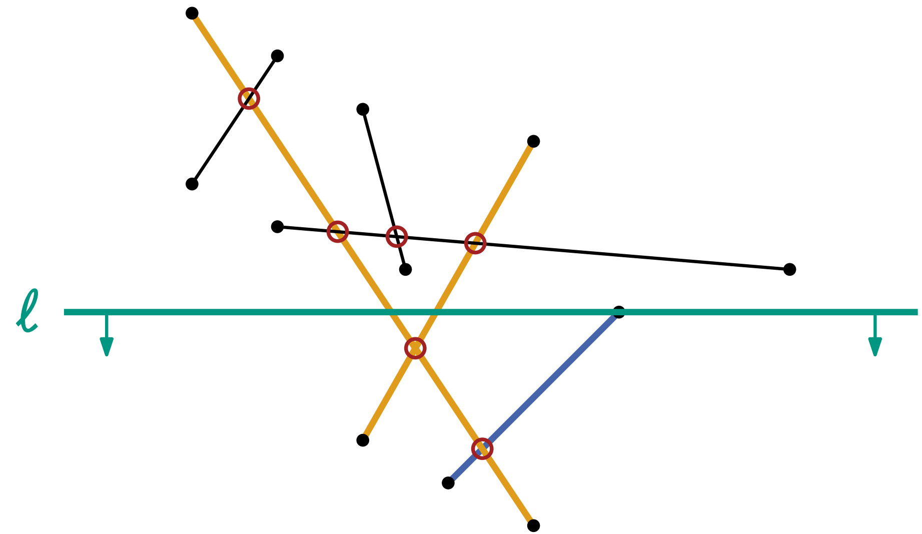


Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Idee

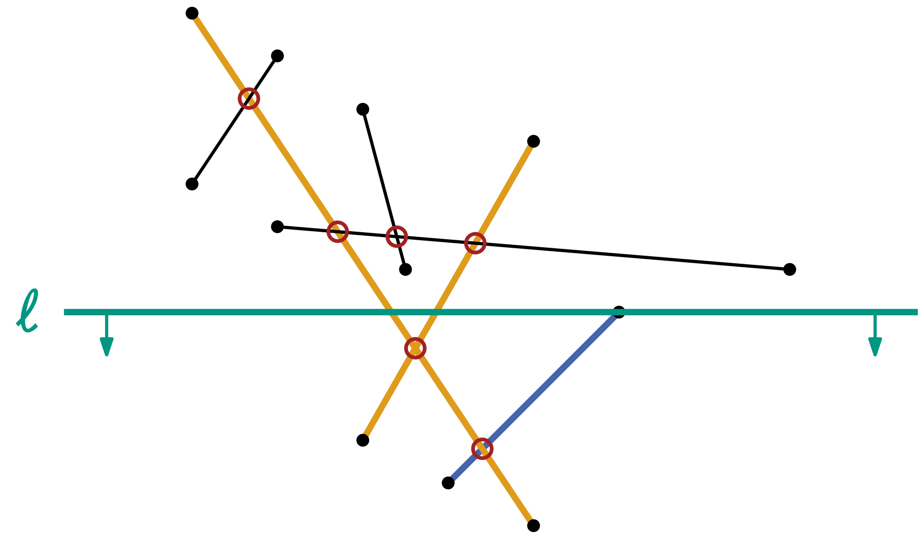
- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Ein einfacher Sweep-Line Algorithmus

Idee

- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



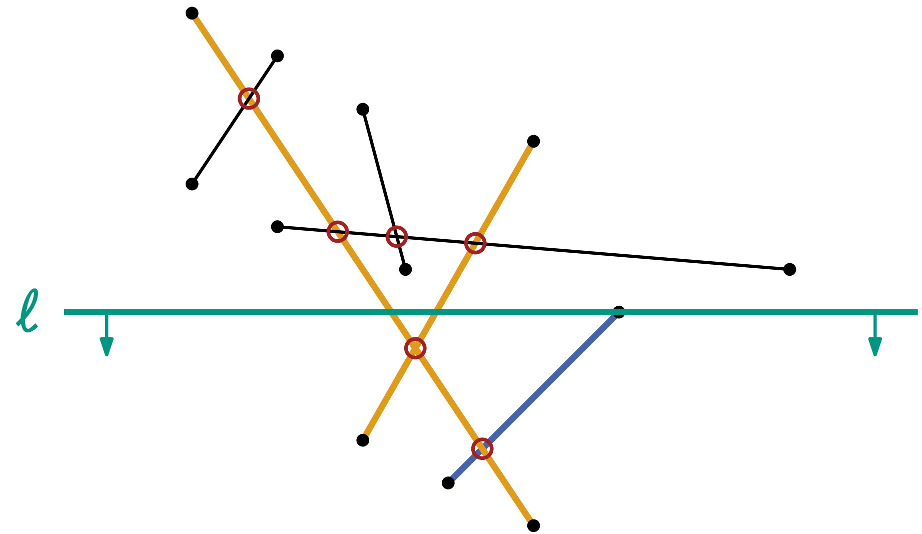
Sweep-Line: etwas formaler

- Sweep-Line Status
  - aktueller Zustand des Sweeps

Ein einfacher Sweep-Line Algorithmus

Idee

- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Sweep-Line: etwas formaler

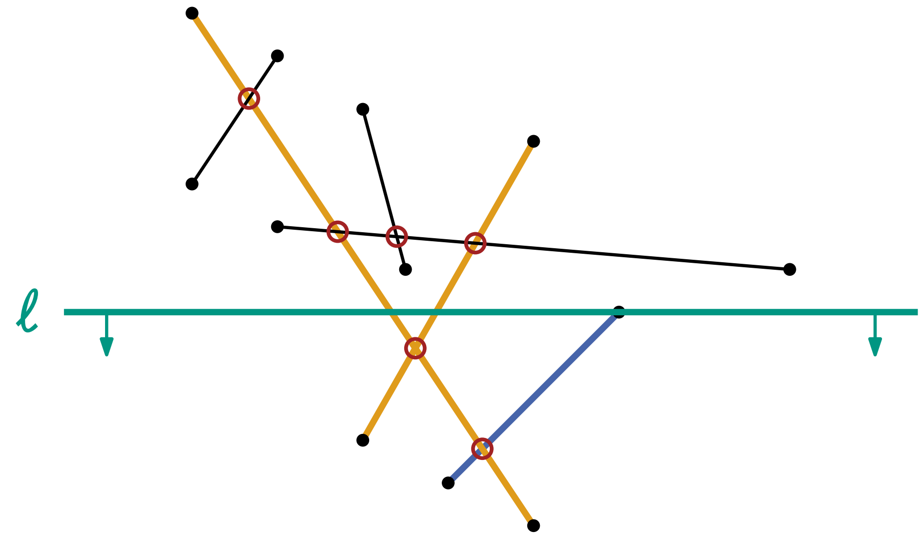
- Sweep-Line Status
  - aktueller Zustand des Sweeps
  - hier: Menge S_ℓ der Strecken, die ℓ schneiden

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Idee

- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Sweep-Line: etwas formaler

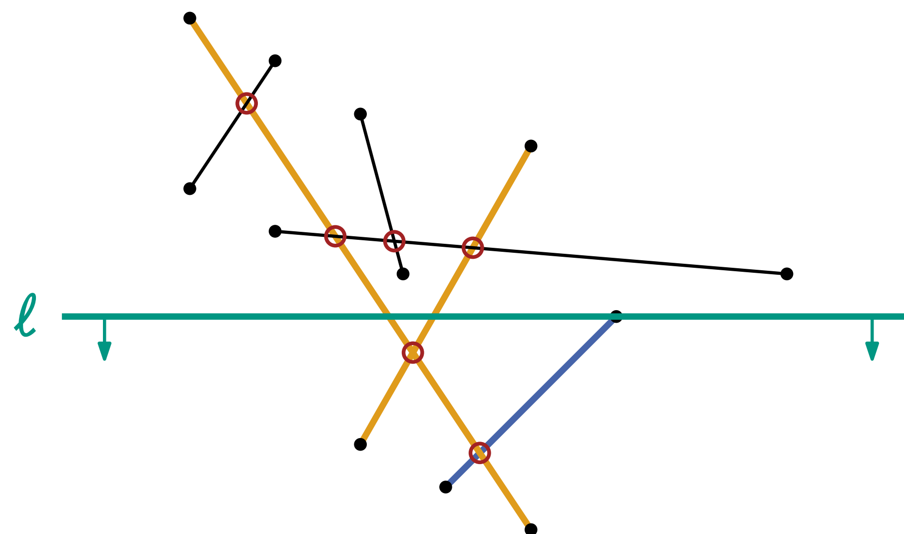
- Sweep-Line Status
  - aktueller Zustand des Sweeps
  - hier: Menge S_ℓ der Strecken, die ℓ schneiden
- Event-Queue
  - zukünftige Positionen der Sweep-Line, an denen spannendes passiert (typischerweise die Positionen, an denen sich der Status ändert)

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Idee

- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Sweep-Line: etwas formaler

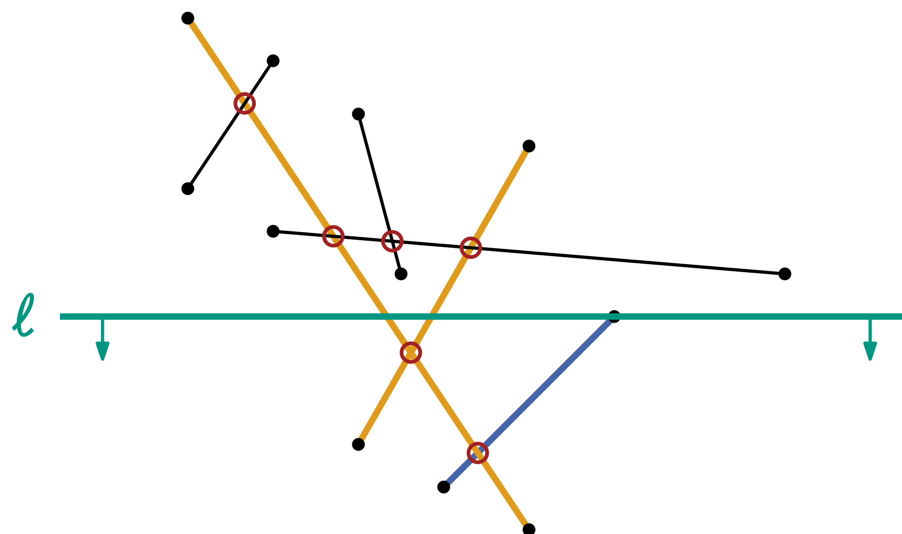
- Sweep-Line Status
  - aktueller Zustand des Sweeps
  - hier: Menge S_ℓ der Strecken, die ℓ schneiden
- Event-Queue
  - zukünftige Positionen der Sweep-Line, an denen spannendes passiert (typischerweise die Positionen, an denen sich der Status ändert)
  - hier: Start- und Endpunkte der Strecken

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Idee

- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Sweep-Line: etwas formaler

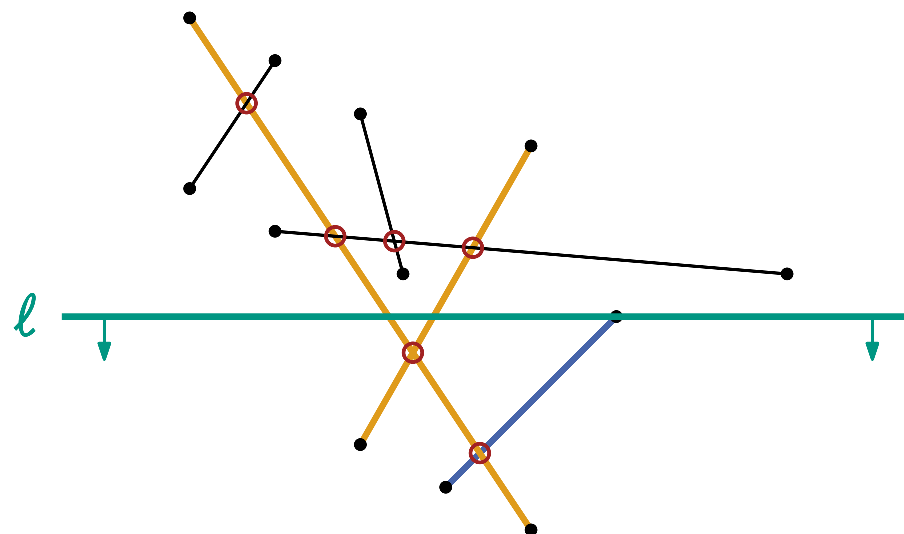
- Sweep-Line Status
  - aktueller Zustand des Sweeps
  - hier: Menge S_ℓ der Strecken, die ℓ schneiden
- Event-Queue
  - zukünftige Positionen der Sweep-Line, an denen spannendes passiert (typischerweise die Positionen, an denen sich der Status ändert)
  - hier: Start- und Endpunkte der Strecken
- Event-Handler; hier:

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Idee

- vergleiche Strecken nicht, wenn eine komplett über der anderen liegt
- schiebe horizontale Gerade ℓ von oben nach unten
- neue Strecke $\rightarrow$ teste auf Schnitt mit allen von ℓ geschnittenen Strecken



Sweep-Line: etwas formaler

- Sweep-Line Status
  - aktueller Zustand des Sweeps
  - hier: Menge S_ℓ der Strecken, die ℓ schneiden
- Event-Queue
  - zukünftige Positionen der Sweep-Line, an denen spannendes passiert (typischerweise die Positionen, an denen sich der Status ändert)
  - hier: Start- und Endpunkte der Strecken
- Event-Handler; hier:
  - Endpunkt der Strecke $s \rightarrow$ setze $S_\ell = S_\ell - s$
  - Startpunkt $\rightarrow$ teste s auf Schnitt mit Strecken in S_ℓ und setze $S_\ell = S_\ell + s$

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

Ein einfacher Sweep-Line Algorithmus

KIT Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

$Q =$ nach y-Koord. sortierte Liste aller Streckenendpunkte // Event-Queue
$S_\ell =$ leere Liste // Sweep-Line Status

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FindeSchnittpunkte(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

```
Q = nach y-Koord. sortierte Liste aller Streckenendpunkte     // Event-Queue
S_ℓ = leere Liste                                              // Sweep-Line Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BehandleEventPunkt(p)                                      // Event-Handler
```

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

```
Q = nach y-Koord. sortierte Liste aller Streckenendpunkte      // Event-Queue
S_ℓ = leere Liste                                               // Sweep-Line Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENTPUNKT(p)                                       // Event-Handler
```

BEHANDLEEVENTPUNKT(p)

```
s = Strecke mit Endpunkt p
wenn s ∈ S_ℓ                                                    // Strecke endet
    S_ℓ = S_ℓ − s
```

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FindeSchnittpunkte(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

```
Q = nach y-Koord. sortierte Liste aller Streckenendpunkte      // Event-Queue
S_ℓ = leere Liste                                              // Sweep-Line Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BehandleEventPunkt(p)                                      // Event-Handler
```

BehandleEventPunkt(p)

```
s = Strecke mit Endpunkt p
wenn s ∈ S_ℓ                                                   // Strecke endet
    S_ℓ = S_ℓ − s
sonst                                                          // Strecke startet
    für alle s' ∈ S_ℓ
        wenn s ∩ s' ≠ ∅ gib (s ∩ s', s, s') aus
    S_ℓ = S_ℓ + s
```

Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

```
Q = nach y-Koord. sortierte Liste aller Streckenendpunkte     // Event-Queue
S_ℓ = leere Liste                                            // Sweep-Line Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENTPUNKT(p)                                    // Event-Handler
```

BEHANDLEEVENTPUNKT(p)

```
s = Strecke mit Endpunkt p
wenn s ∈ S_ℓ                                                 // Strecke endet
    S_ℓ = S_ℓ − s
sonst                                                        // Strecke startet
    für alle s' ∈ S_ℓ
        wenn s ∩ s' ≠ ∅ gib (s ∩ s', s, s') aus
    S_ℓ = S_ℓ + s
```

Problem: langsam ($O(n^2)$), wenn $|S_\ell|$ meistens groß ist

Ein einfacher Sweep-Line Algorithmus

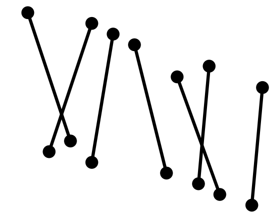
Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

```
Q = nach y-Koord. sortierte Liste aller Streckenendpunkte     // Event-Queue
S_ℓ = leere Liste                                             // Sweep-Line Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENTPUNKT(p)                                     // Event-Handler
```

BEHANDLEEVENTPUNKT(p)

```
s = Strecke mit Endpunkt p
wenn s ∈ S_ℓ                                                  // Strecke endet
    S_ℓ = S_ℓ − s
sonst                                                         // Strecke startet
    für alle s' ∈ S_ℓ
        wenn s ∩ s' ≠ ∅ gib (s ∩ s', s, s') aus
    S_ℓ = S_ℓ + s
```

Problem: langsam ($O(n^2)$), wenn $|S_\ell|$ meistens groß ist



Ein einfacher Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

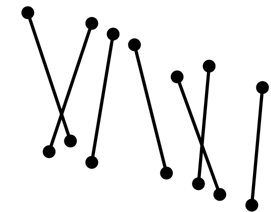
```
Q = nach y-Koord. sortierte Liste aller Streckenendpunkte    // Event-Queue
S_ℓ = leere Liste                                           // Sweep-Line Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENTPUNKT(p)                                   // Event-Handler
```

BEHANDLEEVENTPUNKT(p)

```
s = Strecke mit Endpunkt p
wenn s ∈ S_ℓ                                                // Strecke endet
    S_ℓ = S_ℓ − s
sonst                                                       // Strecke startet
    für alle s' ∈ S_ℓ
        wenn s ∩ s' ≠ ∅ gib (s ∩ s', s, s') aus
    S_ℓ = S_ℓ + s
```

Problem: langsam ($O(n^2)$), wenn $|S_\ell|$ meistens groß ist

Beobachtung: schneidende Strecken sind irgendwann benachbart (auf der Sweep-Line)



Ein einfacher Sweep-Line Algorithmus

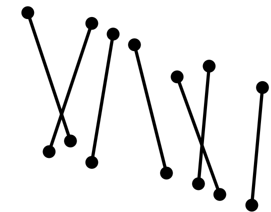
Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten; jeweils mit den dazugehörigen Strecken

```
Q = nach y-Koord. sortierte Liste aller Streckenendpunkte      // Event-Queue
S_ℓ = leere Liste                                              // Sweep-Line Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENTPUNKT(p)                                      // Event-Handler
```

BEHANDLEEVENTPUNKT(p)

```
s = Strecke mit Endpunkt p
wenn s ∈ S_ℓ                                                   // Strecke endet
    S_ℓ = S_ℓ − s
sonst                                                          // Strecke startet
    für alle s' ∈ S_ℓ
        wenn s ∩ s' ≠ ∅ gib (s ∩ s', s, s') aus
    S_ℓ = S_ℓ + s
```

Problem: langsam ($O(n^2)$), wenn $|S_\ell|$ meistens groß ist
Beobachtung: schneidende Strecken sind irgendwann benachbart (auf der Sweep-Line)
Lösung: vergleiche nur benachbarte Strecken



Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

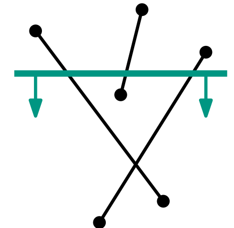
- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten

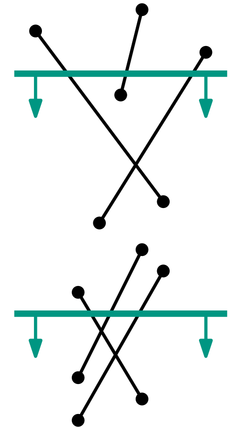


Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

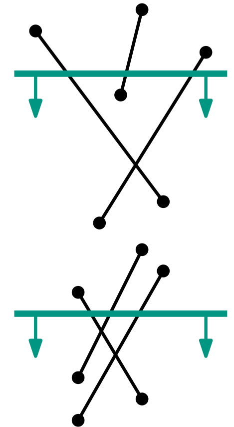
- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten
  - Schnittpunkt: Reihenfolge der schneidenden Strecken ändern und Schnitttest für neu benachbarte



Vergleich benachbarter Strecken

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten
  - Schnittpunkt: Reihenfolge der schneidenden Strecken ändern und Schnitttest für neu benachbarte
  - Schnittpunkt gefunden: Schnittpunkt in Event-Queue einfügen

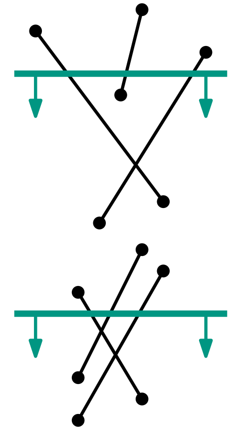


Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten
  - Schnittpunkt: Reihenfolge der schneidenden Strecken ändern und Schnitttest für neu benachbarte
  - Schnittpunkt gefunden: Schnittpunkt in Event-Queue einfügen



Welche Datenstrukturen?

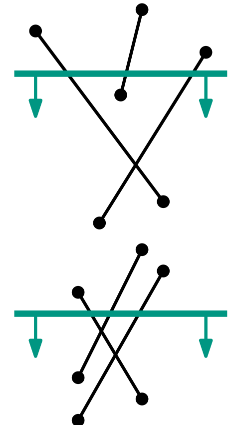
- Sweep-Line Status: einfügen, löschen, Vorgänger, Nachfolger

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten
  - Schnittpunkt: Reihenfolge der schneidenden Strecken ändern und Schnitttest für neu benachbarte
  - Schnittpunkt gefunden: Schnittpunkt in Event-Queue einfügen



Welche Datenstrukturen?

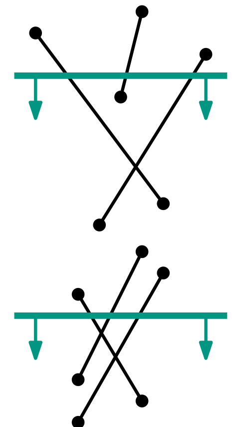
- Sweep-Line Status: einfügen, löschen, Vorgänger, Nachfolger
  - binärer Suchbaum: $O(\log n)$ (z.B. AVL-Baum, rot-schwarz Baum)

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten
  - Schnittpunkt: Reihenfolge der schneidenden Strecken ändern und Schnitttest für neu benachbarte
  - Schnittpunkt gefunden: Schnittpunkt in Event-Queue einfügen



Welche Datenstrukturen?

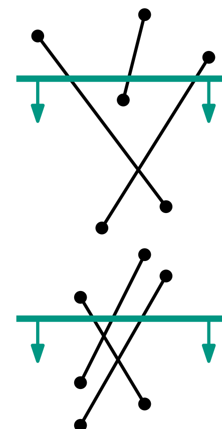
- Sweep-Line Status: einfügen, löschen, Vorgänger, Nachfolger
  - binärer Suchbaum: $O(\log n)$ (z.B. AVL-Baum, rot-schwarz Baum)
- Event-Queue: einfügen, Minimum finden/entfernen, suchen

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten
  - Schnittpunkt: Reihenfolge der schneidenden Strecken ändern und Schnitttest für neu benachbarte
  - Schnittpunkt gefunden: Schnittpunkt in Event-Queue einfügen



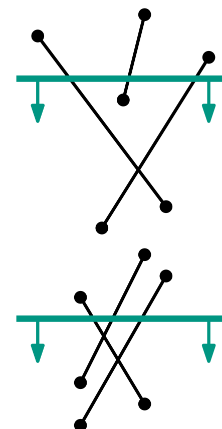
Welche Datenstrukturen?

- Sweep-Line Status: einfügen, löschen, Vorgänger, Nachfolger
  - binärer Suchbaum: $O(\log n)$ (z.B. AVL-Baum, rot-schwarz Baum)
- Event-Queue: einfügen, Minimum finden/entfernen, suchen
  - binärer Suchbaum: $O(\log n)$

Vergleich benachbarter Strecken

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten
  - Schnittpunkt: Reihenfolge der schneidenden Strecken ändern und Schnitttest für neu benachbarte
  - Schnittpunkt gefunden: Schnittpunkt in Event-Queue einfügen



Welche Datenstrukturen?

- Sweep-Line Status: einfügen, löschen, Vorgänger, Nachfolger
  - binärer Suchbaum: $O(\log n)$ (z.B. AVL-Baum, rot-schwarz Baum)
- Event-Queue: einfügen, Minimum finden/entfernen, suchen
  - binärer Suchbaum: $O(\log n)$

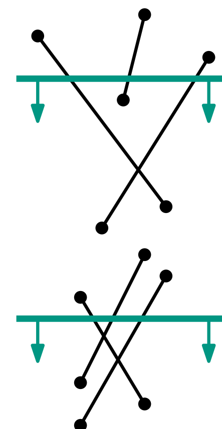
Warum kein Heap?
(Fibonacci-/Hollow-Heap)

Vergleich benachbarter Strecken

Karlsruher Institut für Technologie

Was ändert sich?

- Sweep-Line Status: von ℓ geschnittene Strecken in sortierter Form
- Event-Handler
  - Streckenstart: Schnitttest nur mit benachbarten
  - Streckenstart: Sortierung erhalten beim Einfügen
  - Streckenende: Schnitttest für neu benachbarten
  - Schnittpunkt: Reihenfolge der schneidenden Strecken ändern und Schnitttest für neu benachbarte
  - Schnittpunkt gefunden: Schnittpunkt in Event-Queue einfügen



Welche Datenstrukturen?

- Sweep-Line Status: einfügen, löschen, Vorgänger, Nachfolger
  - binärer Suchbaum: $O(\log n)$ (z.B. AVL-Baum, rot-schwarz Baum)
- Event-Queue: einfügen, Minimum finden/entfernen, suchen
  - binärer Suchbaum: $O(\log n)$

Warum kein Heap?
(Fibonacci-/Hollow-Heap)

Warum brauchen wir „suchen“ für die Queue?

Verbesserter Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

```
Q = leere Queue              // Event-Queue
für pq ∈ S
    Q = Q + p + q
T = binärer Suchbaum         // Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENT(p)         // Event-Handler
```

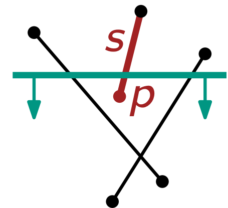
Verbesserter Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

$Q =$ leere Queue // Event-Queue
für $pq \in S$
  $Q = Q + p + q$
$T =$ binärer Suchbaum // Status
solange $Q \neq \emptyset$
  $p = \min\{Q\}$ und $Q = Q - p$
  BEHANDLEEVENT(p) // Event-Handler

BEHANDLEEVENT(p)
wenn p ist Endpunkt einer Strecke
  $s =$ Strecke mit Endpunkt p
  wenn $s \in T$ // Streckenende



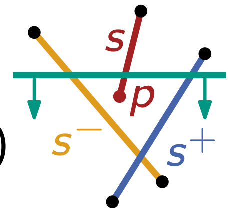
Verbesserter Sweep-Line Algorithmus

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

- $Q =$ leere Queue // Event-Queue
- **für** $pq \in S$
  - $Q = Q + p + q$
- $T =$ binärer Suchbaum // Status
- **solange** $Q \neq \emptyset$
  - $p = \min\{Q\}$ und $Q = Q - p$
  - BEHANDLEEVENT(p) // Event-Handler

BEHANDLEEVENT(p)

- **wenn** p ist Endpunkt einer Strecke
  - $s =$ Strecke mit Endpunkt p
  - **wenn** $s \in T$ // Streckenende
    - $s^- =$ Vorgänger von s in T
    - s^+ = Nachfolger von s in T
    - FINDENEUESEVENT(s^-, s^+, p)
    - $T = T - s$



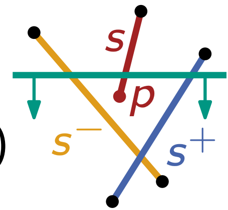
Verbesserter Sweep-Line Algorithmus

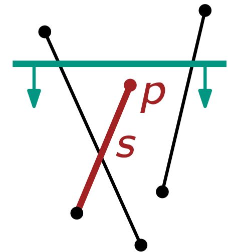
Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

- Q = leere Queue // Event-Queue
- **für** $pq \in S$
  - $Q = Q + p + q$
- T = binärer Suchbaum // Status
- **solange** $Q \neq \emptyset$
  - $p = \min\{Q\}$ und $Q = Q - p$
  - BEHANDLEEVENT(p) // Event-Handler

BEHANDLEEVENT(p)

- **wenn** p ist Endpunkt einer Strecke
  - s = Strecke mit Endpunkt p
  - **wenn** $s \in T$ // Streckenende
    - s^- = Vorgänger von s in T
    - s^+ = Nachfolger von s in T
    - FINDENEUESEVENT(s^-, s^+, p)
    - $T = T - s$
  - **sonst** // Streckenstart





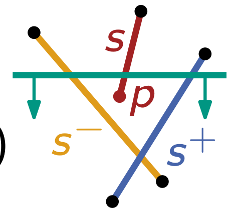
Verbesserter Sweep-Line Algorithmus

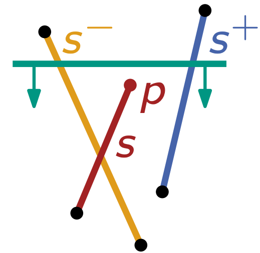
Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

```
Q = leere Queue                       // Event-Queue
für pq ∈ S
    Q = Q + p + q
T = binärer Suchbaum                  // Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENT(p)                  // Event-Handler
```

BEHANDLEEVENT(p)

```
wenn p ist Endpunkt einer Strecke
    s = Strecke mit Endpunkt p
    wenn s ∈ T                                // Streckenende
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s⁺, p)
        T = T − s
    sonst                                     // Streckenstart
        T = T + s
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s, p)
        FINDENEUESEVENT(s⁺, s, p)
```





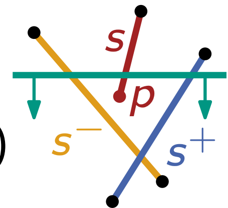
Verbesserter Sweep-Line Algorithmus

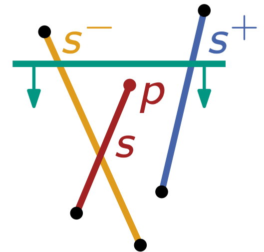
Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

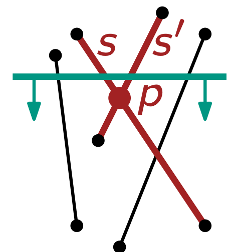
```
Q = leere Queue                         // Event-Queue
für pq ∈ S
    Q = Q + p + q
T = binärer Suchbaum                    // Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENT(p)                    // Event-Handler
```

BEHANDLEEVENT(p)

```
wenn p ist Endpunkt einer Strecke
    s = Strecke mit Endpunkt p
    wenn s ∈ T                          // Streckenende
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s⁺, p)
        T = T − s
    sonst                               // Streckenstart
        T = T + s
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s, p)
        FINDENEUESEVENT(s⁺, s, p)
sonst                                   // Schnittpunkt
    s, s′ = Strecken mit Schnitt p (s < s′ in T)
    gib (p, s, s′) aus
```







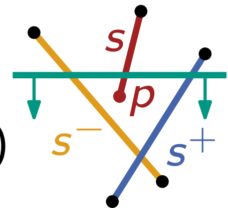
Verbesserter Sweep-Line Algorithmus

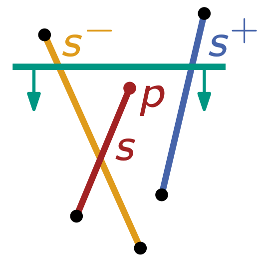
Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

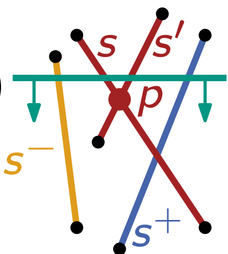
```
Q = leere Queue                  // Event-Queue
für pq ∈ S
    Q = Q + p + q
T = binärer Suchbaum             // Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENT(p)             // Event-Handler
```

BEHANDLEEVENT(p)

```
wenn p ist Endpunkt einer Strecke
    s = Strecke mit Endpunkt p
    wenn s ∈ T                                   // Streckenende
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s⁺, p)
        T = T − s
    sonst                                        // Streckenstart
        T = T + s
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s, p)
        FINDENEUESEVENT(s⁺, s, p)
sonst                                            // Schnittpunkt
    s, s′ = Strecken mit Schnitt p (s < s′ in T)
    gib (p, s, s′) aus
    vertausche s und s′ in T (s′ < s)
    s⁻ = Vorgänger von s′ in T
    s⁺ = Nachfolger von s in T
    FINDENEUESEVENT(s⁻, s′, p)
    FINDENEUESEVENT(s⁺, s, p)
```







Verbesserter Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

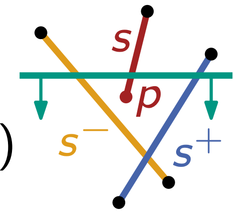
```
Q = leere Queue                    // Event-Queue
für pq ∈ S
    Q = Q + p + q
T = binärer Suchbaum               // Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENT(p)               // Event-Handler
```

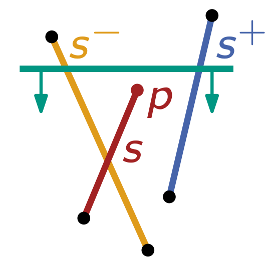
FINDENEUESEVENT(s^-, s^+, p)

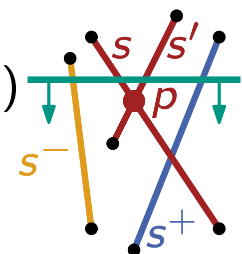
```
q = s⁻ ∩ s⁺                        // Schnittpunkt
```

BEHANDLEEVENT(p)

```
wenn p ist Endpunkt einer Strecke
    s = Strecke mit Endpunkt p
    wenn s ∈ T                                 // Streckenende
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s⁺, p)
        T = T − s
    sonst                                      // Streckenstart
        T = T + s
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s, p)
        FINDENEUESEVENT(s⁺, s, p)
sonst                                          // Schnittpunkt
    s, s′ = Strecken mit Schnitt p (s < s′ in T)
    gib (p, s, s′) aus
    vertausche s und s′ in T (s′ < s)
    s⁻ = Vorgänger von s′ in T
    s⁺ = Nachfolger von s in T
    FINDENEUESEVENT(s⁻, s′, p)
    FINDENEUESEVENT(s⁺, s, p)
```







Verbesserter Sweep-Line Algorithmus

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

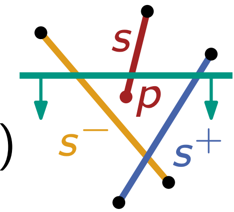
```
Q = leere Queue                       // Event-Queue
für pq ∈ S
    Q = Q + p + q
T = binärer Suchbaum                  // Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENT(p)                  // Event-Handler
```

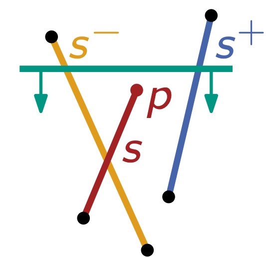
FINDENEUESEVENT(s^-, s^+, p)

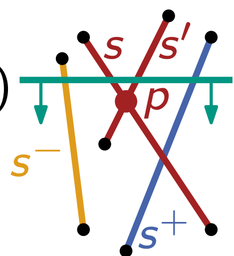
```
q = s⁻ ∩ s⁺                           // Schnittpunkt
wenn q ≠ ∅ und p_y < q_y              // nach der
    wenn q ∉ Q                        // aktuellen
        Q = Q + q                     // Position der
                                      // Sweep-Line
```

BEHANDLEEVENT(p)

```
wenn p ist Endpunkt einer Strecke
    s = Strecke mit Endpunkt p
    wenn s ∈ T                                    // Streckenende
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s⁺, p)
        T = T − s
    sonst                                         // Streckenstart
        T = T + s
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s, p)
        FINDENEUESEVENT(s⁺, s, p)
sonst                                             // Schnittpunkt
    s, s' = Strecken mit Schnitt p (s < s' in T)
    gib (p, s, s') aus
    vertausche s und s' in T (s' < s)
    s⁻ = Vorgänger von s' in T
    s⁺ = Nachfolger von s in T
    FINDENEUESEVENT(s⁻, s', p)
    FINDENEUESEVENT(s⁺, s, p)
```







Verbesserter Sweep-Line Algorithmus

Karlsruher Institut für Technologie

Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

```
Q = leere Queue                      // Event-Queue
für pq ∈ S
    Q = Q + p + q
T = binärer Suchbaum                 // Status
solange Q ≠ ∅
    p = min{Q} und Q = Q − p
    BEHANDLEEVENT(p)                 // Event-Handler
```

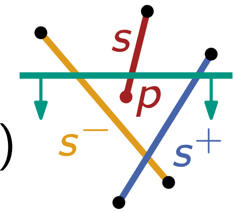
FINDENEUESEVENT(s^-, s^+, p)

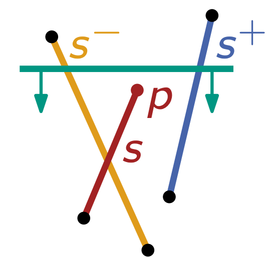
```
q = s⁻ ∩ s⁺                          // Schnittpunkt
wenn q ≠ ∅ und p_y < q_y             // nach der
    wenn q ∉ Q                       // aktuellen
        Q = Q + q                    // Position der
                                     // Sweep-Line
```

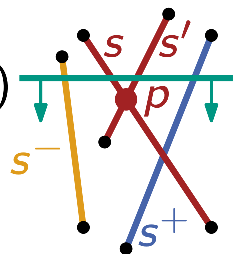
Laufzeit?

BEHANDLEEVENT(p)

```
wenn p ist Endpunkt einer Strecke
    s = Strecke mit Endpunkt p
    wenn s ∈ T                                   // Streckenende
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s⁺, p)
        T = T − s
    sonst                                        // Streckenstart
        T = T + s
        s⁻ = Vorgänger von s in T
        s⁺ = Nachfolger von s in T
        FINDENEUESEVENT(s⁻, s, p)
        FINDENEUESEVENT(s⁺, s, p)
sonst                                            // Schnittpunkt
    s, s′ = Strecken mit Schnitt p (s < s′ in T)
    gib (p, s, s′) aus
    vertausche s und s′ in T (s′ < s)
    s⁻ = Vorgänger von s′ in T
    s⁺ = Nachfolger von s in T
    FINDENEUESEVENT(s⁻, s′, p)
    FINDENEUESEVENT(s⁺, s, p)
```







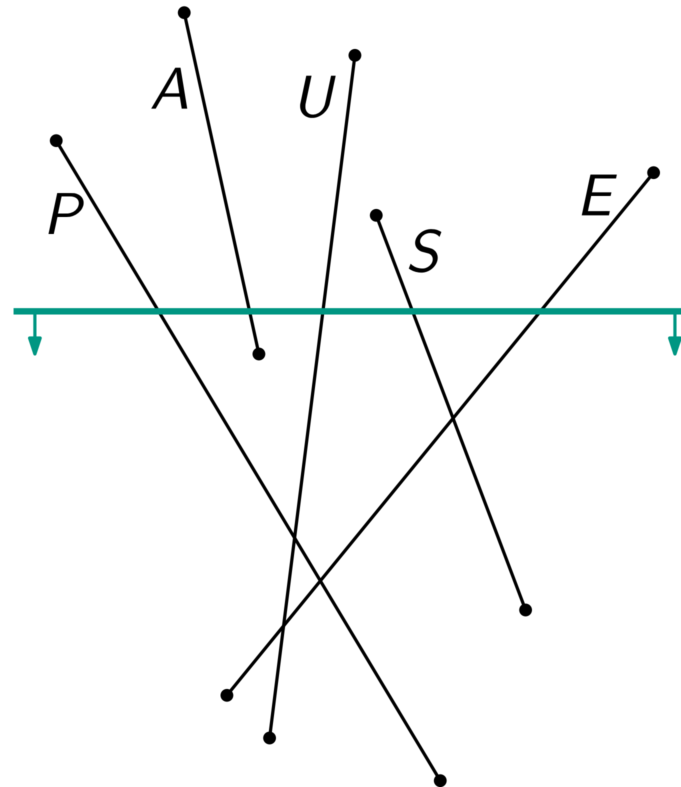
Verbesserter Sweep-Line Algorithmus

Karlsruher Institut für Technologie

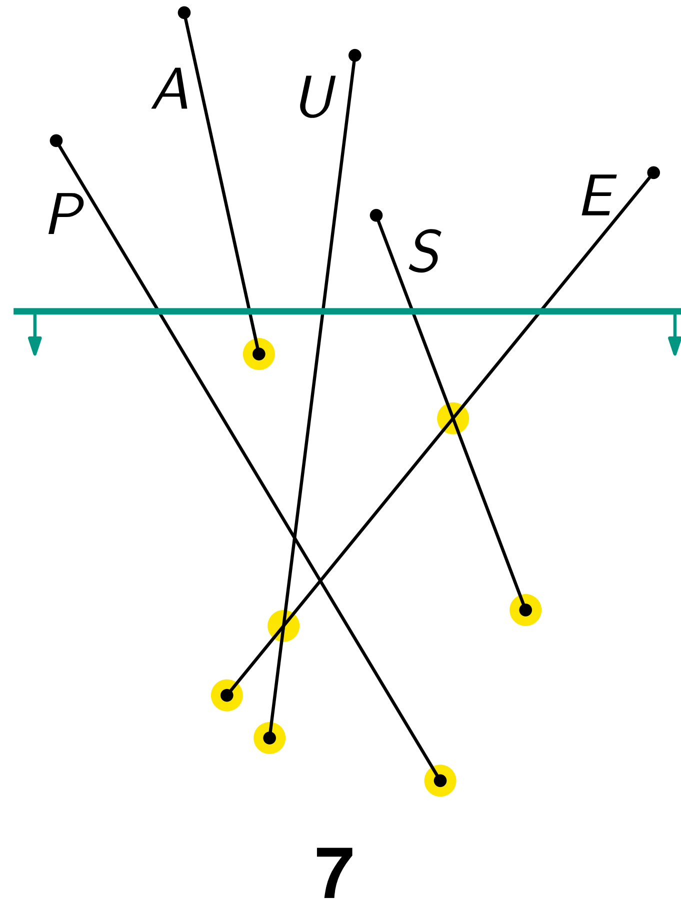
Theorem **Annahme:** allgemeine Lage

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n + k) \log n)$ berechnet werden.

Wie viele Events enthält die Event-Queue aktuell?



Wie viele Events enthält die Event-Queue aktuell?



7

Robustheit

Karlsruher Institut für Technologie

Theorem **Annahme:** allgemeine Lage

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n + k) \log n)$ berechnet werden.

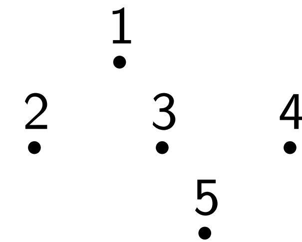
Robustheit

Karlsruher Institut für Technologie

Theorem **Annahme:** allgemeine Lage
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Problem: Eventpunkte mit gleicher y-Koordinate

- sortiere Eventpunkte lexikografisch nach y, x
- äquivalent zu einer leichten Rotation (im Uhrzeigersinn)



Robustheit

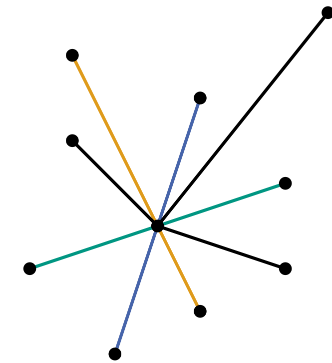
Karlsruher Institut für Technologie

Theorem **Annahme:** allgemeine Lage

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Problem: mehrere Ereignisse an einem Eventpunkt

- drei mögliche Ereignisse pro Eventpunkt
  - Startpunkt einer (oder mehrerer) Strecke
  - Endpunkt einer (oder mehrerer) Strecke
  - Schnittpunkt von Strecken



Robustheit

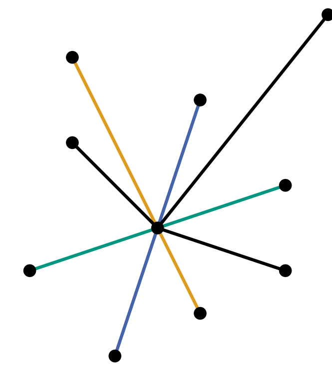
Karlsruher Institut für Technologie

Theorem **Annahme:** allgemeine Lage

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Strecken-paaren können in $O((n+k)\log n)$ berechnet werden.

Problem: mehrere Ereignisse an einem Eventpunkt

- drei mögliche Ereignisse pro Eventpunkt
  - Startpunkt einer (oder mehrerer) Strecke
  - Endpunkt einer (oder mehrerer) Strecke
  - Schnittpunkt von Strecken
- Plan: behandle alle Ereignisse an einem Eventpunkt gemeinsam

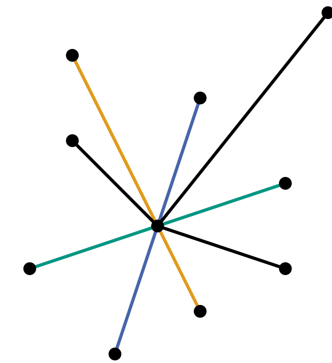


Robustheit

Theorem **Annahme:** allgemeine Lage
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Problem: mehrere Ereignisse an einem Eventpunkt

- drei mögliche Ereignisse pro Eventpunkt
  - Startpunkt einer (oder mehrerer) Strecke
  - Endpunkt einer (oder mehrerer) Strecke
  - Schnittpunkt von Strecken
- Plan: behandle alle Ereignisse an einem Eventpunkt gemeinsam
- Strecken, die bei p starten/enden/schneiden: $\mathsf{start}(p)/\mathsf{end}(p)/\mathsf{schnitt}(p)$

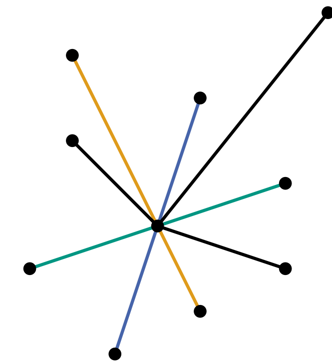


Robustheit

Theorem **Annahme:** allgemeine Lage

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Problem: mehrere Ereignisse an einem Eventpunkt



- drei mögliche Ereignisse pro Eventpunkt
  - Startpunkt einer (oder mehrerer) Strecke
  - Endpunkt einer (oder mehrerer) Strecke
  - Schnittpunkt von Strecken
- Plan: behandle alle Ereignisse an einem Eventpunkt gemeinsam
- Strecken, die bei p starten/enden/schneiden: $\mathsf{start}(p)$/$\mathsf{end}(p)$/$\mathsf{schnitt}(p)$
- $\mathsf{start}(p)$ muss explizit bei p gespeichert sein
- $\mathsf{end}(p)$ und $\mathsf{schnitt}(p)$ ergeben sich aus dem Sweep-Line Status T Wie?

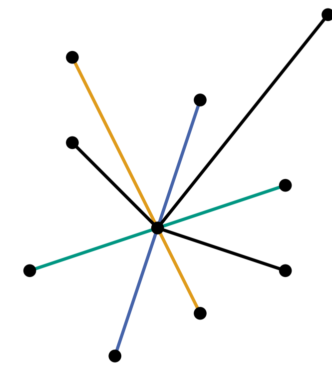
Robustheit

Karlsruher Institut für Technologie

Theorem **Annahme:** allgemeine Lage

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Strecken-paaren können in $O((n + k) \log n)$ berechnet werden.

Problem: mehrere Ereignisse an einem Eventpunkt



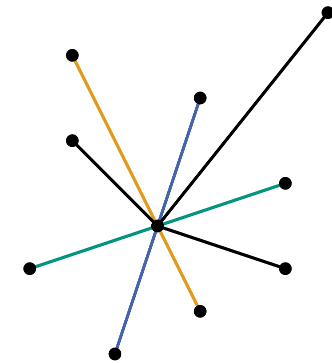
- drei mögliche Ereignisse pro Eventpunkt
  - Startpunkt einer (oder mehrerer) Strecke
  - Endpunkt einer (oder mehrerer) Strecke
  - Schnittpunkt von Strecken
- Plan: behandle alle Ereignisse an einem Eventpunkt gemeinsam
- Strecken, die bei p starten/enden/schneiden: $\mathsf{start}(p)$/$\mathsf{end}(p)$/$\mathsf{schnitt}(p)$
- $\mathsf{start}(p)$ muss explizit bei p gespeichert sein
- $\mathsf{end}(p)$ und $\mathsf{schnitt}(p)$ ergeben sich aus dem Sweep-Line Status T Wie?
- Aktualisierung des Sweep-Line Zustands T

Robustheit

Karlsruher Institut für Technologie

Theorem **Annahme:** allgemeine Lage
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Problem: mehrere Ereignisse an einem Eventpunkt



- drei mögliche Ereignisse pro Eventpunkt
  - Startpunkt einer (oder mehrerer) Strecke
  - Endpunkt einer (oder mehrerer) Strecke
  - Schnittpunkt von Strecken
- Plan: behandle alle Ereignisse an einem Eventpunkt gemeinsam
- Strecken, die bei p starten/enden/schneiden: $\mathsf{start}(p)$/$\mathsf{end}(p)$/$\mathsf{schnitt}(p)$
- $\mathsf{start}(p)$ muss explizit bei p gespeichert sein
- $\mathsf{end}(p)$ und $\mathsf{schnitt}(p)$ ergeben sich aus dem Sweep-Line Status T Wie?
- Aktualisierung des Sweep-Line Zustands T
  - lösche $\mathsf{end}(p) \cup \mathsf{schnitt}(p)$
  - füge $\mathsf{schnitt}(p) \cup \mathsf{start}(p)$ ein (entsprechend Sortierung kurz nach p)

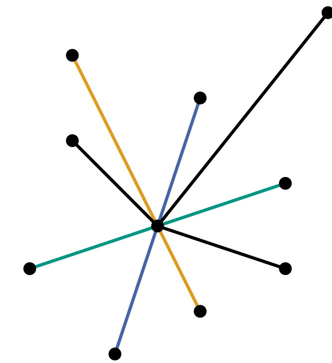
Robustheit

Karlsruher Institut für Technologie

Theorem **Annahme:** allgemeine Lage

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Problem: mehrere Ereignisse an einem Eventpunkt



- drei mögliche Ereignisse pro Eventpunkt
  - Startpunkt einer (oder mehrerer) Strecke
  - Endpunkt einer (oder mehrerer) Strecke
  - Schnittpunkt von Strecken
- Plan: behandle alle Ereignisse an einem Eventpunkt gemeinsam
- Strecken, die bei p starten/enden/schneiden: $\text{start}(p)/\text{end}(p)/\text{schnitt}(p)$
- $\text{start}(p)$ muss explizit bei p gespeichert sein
- $\text{end}(p)$ und $\text{schnitt}(p)$ ergeben sich aus dem Sweep-Line Status T Wie?
- Aktualisierung des Sweep-Line Zustands T
  - lösche $\text{end}(p) \cup \text{schnitt}(p)$ Horizontale Strecken?
  - füge $\text{schnitt}(p) \cup \text{start}(p)$ ein (entsprechend Sortierung kurz nach p)

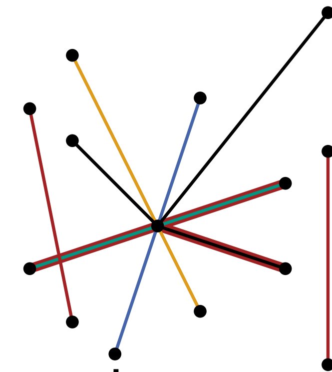
Robustheit

Karlsruher Institut für Technologie

Theorem **Annahme:** allgemeine Lage

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Problem: mehrere Ereignisse an einem Eventpunkt



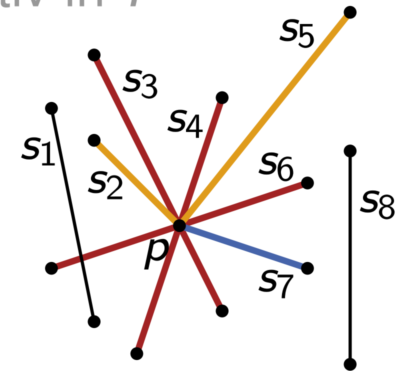
- drei mögliche Ereignisse pro Eventpunkt
  - Startpunkt einer (oder mehrerer) Strecke
  - Endpunkt einer (oder mehrerer) Strecke
  - Schnittpunkt von Strecken
- Plan: behandle alle Ereignisse an einem Eventpunkt gemeinsam
- Strecken, die bei p starten/enden/schneiden: $\mathsf{start}(p)$/$\mathsf{end}(p)$/$\mathsf{schnitt}(p)$
- $\mathsf{start}(p)$ muss explizit bei p gespeichert sein
- $\mathsf{end}(p)$ und $\mathsf{schnitt}(p)$ ergeben sich aus dem Sweep-Line Status T **Wie?**
- Aktualisierung des Sweep-Line Zustands T
  - lösche $\mathsf{end}(p) \cup \mathsf{schnitt}(p)$ **Horizontale Strecken?**
  - füge $\mathsf{schnitt}(p) \cup \mathsf{start}(p)$ ein (entsprechend Sortierung kurz nach p)
- behandle hinterher neu benachbarte Kanten

Robuster Sweep-Line Algorithmus

Karlsruher Institut für Technologie

BEHANDLEEVENT(p)

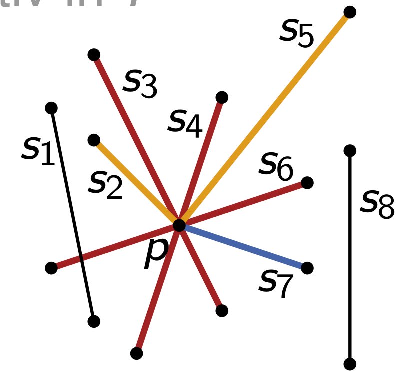
- $\text{start}(p)$ = Strecken die bei p starten // mit p in Q gespeichert
- $\text{end}(p)$, $\text{schnitt}(p)$ = Strecken, die bei p enden/sich schneiden // konsekutiv in T
- **wenn** $|\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)| > 1$
  - **gib** p (mit $\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)$) **aus**



Robuster Sweep-Line Algorithmus

BEHANDLEEVENT(p)

- start(p) = Strecken die bei p starten // mit p in Q gespeichert
- end(p), schnitt(p) = Strecken, die bei p enden/sich schneiden // konsekutiv in T
- **wenn** $|\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)| > 1$
  - **gib** p (mit $\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)$) **aus**
- lösche $\text{end}(p) \cup \text{schnitt}(p)$ aus T — T: s_1 s_2 s_3 s_4 s_5 s_6 s_8 → s_1 s_8
- füge $\text{schnitt}(p) \cup \text{start}(p)$ in T ein — T: s_1 s_8 → s_1 s_6 s_4 s_3 s_7 s_8



Robuster Sweep-Line Algorithmus

BEHANDLEEVENT(p)

start(p) $=$ Strecken die bei p starten // mit p in Q gespeichert

end(p), schnitt(p) $=$ Strecken, die bei p enden/sich schneiden // konsekutiv in T

wenn $|\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)| > 1$

  gib p (mit $\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)$) **aus**

lösche $\text{end}(p) \cup \text{schnitt}(p)$ aus T

füge $\text{schnitt}(p) \cup \text{start}(p)$ in T ein

wenn $\text{start}(p) \cup \text{schnitt}(p) = \emptyset$

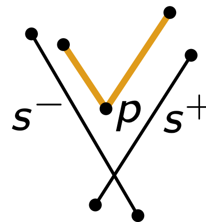
  $s^- =$ Vorgänger von p in T

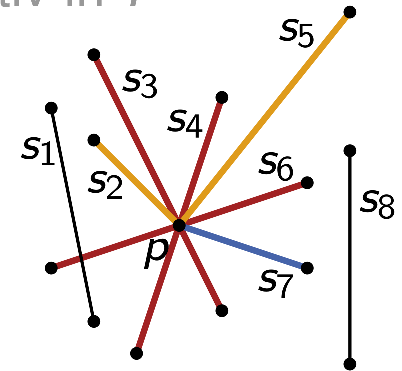
  $s^+ =$ Nachfolger von p in T

  FINDENEUESEVENT(s^-, s^+, p)

T: s_1 s_2 s_3 s_4 s_5 s_6 s_8 → s_1 s_8

T: s_1 s_8 → s_1 s_6 s_4 s_3 s_7 s_8





Robuster Sweep-Line Algorithmus

BEHANDLEEVENT(p)

start(p) = Strecken die bei p starten // mit p in Q gespeichert

end(p), schnitt(p) = Strecken, die bei p enden/sich schneiden // konsekutiv in T

wenn $|\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)| > 1$

  gib p (mit $\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)$) **aus**

lösche $\text{end}(p) \cup \text{schnitt}(p)$ aus T

füge $\text{schnitt}(p) \cup \text{start}(p)$ in T ein

wenn $\text{start}(p) \cup \text{schnitt}(p) = \emptyset$

  s^- = Vorgänger von p in T

  s^+ = Nachfolger von p in T

  FINDENEUESEVENT(s^-, s^+, p)

else

  s^- = linkeste Strecke in $\text{start}(p) \cup \text{schnitt}(p)$

  s^+ = rechteste Strecke in $\text{start}(p) \cup \text{schnitt}(p)$

  $\hat{s}^-$ = Vorgänger von s^- in T

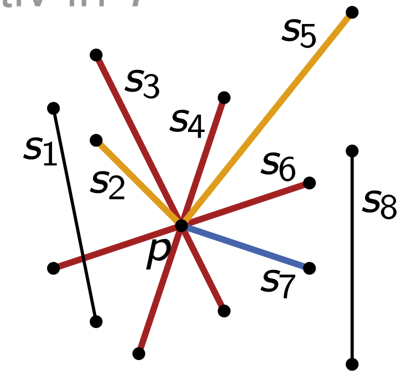
  $\hat{s}^+$ = Nachfolger von s^+ in T

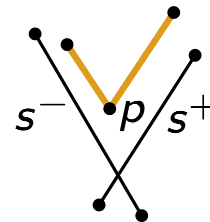
  FINDENEUESEVENT($s^-, \hat{s}^-, p$)

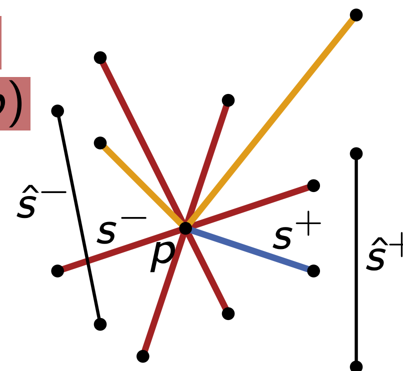
  FINDENEUESEVENT($s^+, \hat{s}^+, p$)

T: s_1 s_2 s_3 s_4 s_5 s_6 s_8 → s_1 s_8

T: s_1 s_8 → s_1 s_6 s_4 s_3 s_7 s_8







Robuster Sweep-Line Algorithmus

BEHANDLEEVENT(p)

start(p) = Strecken die bei p starten // mit p in Q gespeichert

end(p), schnitt(p) = Strecken, die bei p enden/sich schneiden // konsekutiv in T

wenn $|\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)| > 1$

  gib p (mit $\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)$) **aus**

lösche $\text{end}(p) \cup \text{schnitt}(p)$ aus T

füge $\text{schnitt}(p) \cup \text{start}(p)$ in T ein

wenn $\text{start}(p) \cup \text{schnitt}(p) = \emptyset$

  s^- = Vorgänger von p in T

  s^+ = Nachfolger von p in T

  FINDENEUESEVENT(s^-, s^+, p)

else

  s^- = linkeste Strecke in $\text{start}(p) \cup \text{schnitt}(p)$

  s^+ = rechteste Strecke in $\text{start}(p) \cup \text{schnitt}(p)$

  $\hat{s}^-$ = Vorgänger von s^- in T

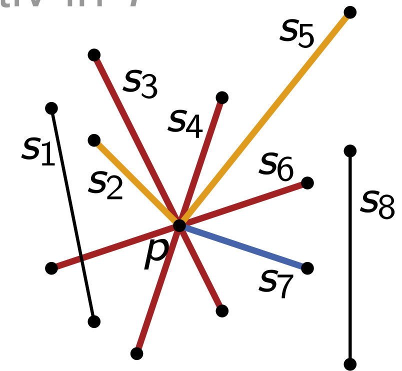
  $\hat{s}^+$ = Nachfolger von s^+ in T

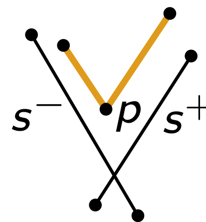
  FINDENEUESEVENT($s^-, \hat{s}^-$

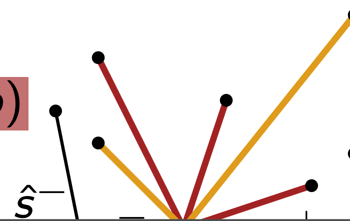
  FINDENEUESEVENT($s^+, \hat{s}^+$,

T: s_1 s_2 s_3 s_4 s_5 s_6 s_8 → s_1 s_8

T: s_1 s_8 → s_1 s_6 s_4 s_3 s_7 s_8







Theorem

Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

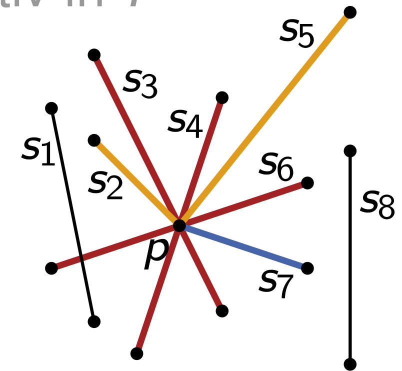
Robuster Sweep-Line Algorithmus

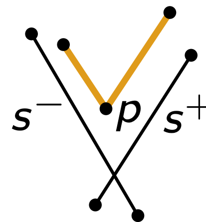
Karlsruher Institut für Technologie

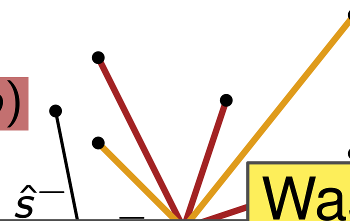
BEHANDLEEVENT(p)

start(p) = Strecken die bei p starten // mit p in Q gespeichert
end(p), schnitt(p) = Strecken, die bei p enden/sich schneiden // konsekutiv in T
wenn $|\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)| > 1$
  gib p (mit $\text{start}(p) \cup \text{end}(p) \cup \text{schnitt}(p)$) **aus**
lösche $\text{end}(p) \cup \text{schnitt}(p)$ aus T
füge $\text{schnitt}(p) \cup \text{start}(p)$ in T ein
wenn $\text{start}(p) \cup \text{schnitt}(p) = \emptyset$
  s^- = Vorgänger von p in T
  s^+ = Nachfolger von p in T
  FINDENEUESEVENT(s^-, s^+, p)
else
  s^- = linkeste Strecke in $\text{start}(p) \cup \text{schnitt}(p)$
  s^+ = rechteste Strecke in $\text{start}(p) \cup \text{schnitt}(p)$
  $\hat{s}^-$ = Vorgänger von s^- in T
  $\hat{s}^+$ = Nachfolger von s^+ in T
  FINDENEUESEVENT($s^-, \hat{s}^-$
  FINDENEUESEVENT($s^+, \hat{s}^+$,

T: s_1 s_2 s_3 s_4 s_5 s_6 s_8 → s_1 s_8
T: s_1 s_8 → s_1 s_6 s_4 s_3 s_7 s_8







Was ist noch zu zeigen?

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Laufzeitanalyse

Karlsruher Institut für Technologie

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$

Laufzeitanalyse

Karlsruher Institut für Technologie

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt

Laufzeitanalyse

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt
  - Minimum in der Queue finden und entfernen $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FINDENEUESEVENT) $O(\log n)$

Laufzeitanalyse

Karlsruher Institut für Technologie

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein — $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt — $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen — $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FINDENEUESEVENT) — $O(\log n)$

Laufzeitanalyse

Karlsruher Institut für Technologie

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FINDENEUESEVENT) $O(\log n)$
- Operationen auf dem Sweep-Line Status bei einem Eventpunkt p

Laufzeitanalyse

Karlsruher Institut für Technologie

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FindeNeuesEvent) $O(\log n)$
- Operationen auf dem Sweep-Line Status bei einem Eventpunkt p
  - hängt von der Anzahl schneidender Strecken $m(p)$ ab

Laufzeitanalyse

Karlsruher Institut für Technologie

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FINDENEUESEVENT) $O(\log n)$
- Operationen auf dem Sweep-Line Status bei einem Eventpunkt p
  - hängt von der Anzahl schneidender Strecken $m(p)$ ab
  - insgesamt $O(m(p))$ Operationen $O(m(p)\log n)$

Laufzeitanalyse

Karlsruher Institut für Technologie

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FINDENEUESEVENT) $O(\log n)$
- Operationen auf dem Sweep-Line Status bei einem Eventpunkt p
  - hängt von der Anzahl schneidender Strecken $m(p)$ ab
  - insgesamt $O(m(p))$ Operationen $O(m(p)\log n)$

Gesamtlaufzeit: $(n+k)\log n + m \log n$ mit $m = \sum_p m(p)$

Laufzeitanalyse

Karlsruher Institut für Technologie

Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Strecken-paaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein — $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt — $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen — $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FINDENEUESEVENT) — $O(\log n)$
- Operationen auf dem Sweep-Line Status bei einem Eventpunkt p
  - hängt von der Anzahl schneidender Strecken $m(p)$ ab
  - insgesamt $O(m(p))$ Operationen — $O(m(p)\log n)$

Gesamtlaufzeit: $(n+k)\log n + m \log n$ mit $m = \sum_p m(p)$

Gilt $m \in O(n+k)$?

Laufzeitanalyse

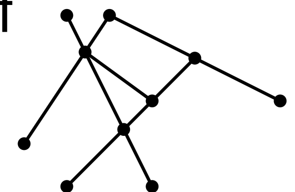
Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FindeNeuesEvent) $O(\log n)$
- Operationen auf dem Sweep-Line Status bei einem Eventpunkt p
  - hängt von der Anzahl schneidender Strecken $m(p)$ ab
  - insgesamt $O(m(p))$ Operationen $O(m(p)\log n)$

Gesamtlaufzeit: $(n+k)\log n + m \log n$ mit $m = \sum_p m(p)$ Gilt $m \in O(n+k)$?

- fasse Strecken-Arrangement als planarer Graph $G = (V, E)$ auf



Laufzeitanalyse

Karlsruher Institut für Technologie

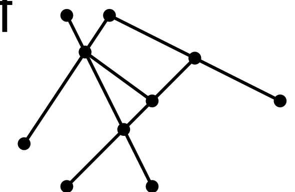
Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FindeNeuesEvent) $O(\log n)$
- Operationen auf dem Sweep-Line Status bei einem Eventpunkt p
  - hängt von der Anzahl schneidender Strecken $m(p)$ ab
  - insgesamt $O(m(p))$ Operationen $O(m(p)\log n)$

Gesamtlaufzeit: $(n+k)\log n + m \log n$ mit $m = \sum_p m(p)$ Gilt $m \in O(n+k)$?

- fasse Strecken-Arrangement als planarer Graph $G = (V, E)$ auf
- $|V| \leq 2n + k$ und $2|E| = m$



Laufzeitanalyse

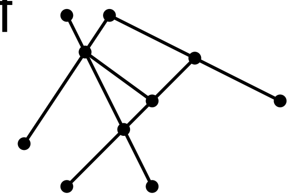
Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Strecken-paaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

- Initialisierung: füge $2n$ Eventpunkte in die Queue ein — $O(n \log n)$
- Queue-Operationen bei einem Eventpunkt — $O((n+k)\log n)$
  - Minimum in der Queue finden und entfernen — $O(\log n)$
  - bis zu zwei Einfügeoperationen (in FINDENEUESEVENT) — $O(\log n)$
- Operationen auf dem Sweep-Line Status bei einem Eventpunkt p
  - hängt von der Anzahl schneidender Strecken $m(p)$ ab
  - insgesamt $O(m(p))$ Operationen — $O(m(p)\log n)$

Gesamtlaufzeit: $(n+k)\log n + m \log n$ mit $m = \sum_p m(p)$ — Gilt $m \in O(n+k)$?

- fasse Strecken-Arrangement als planarer Graph $G = (V, E)$ auf
- $|V| \leq 2n + k$ und $2|E| = m$
- in planaren Graphen gilt: $|E| \leq 3|V| - 6 \Rightarrow m \in O(n+k)$



Speicherverbrauch

Karlsruher Institut für Technologie

Warum interessiert uns das?

Speicherverbrauch

Karlsruher Institut für Technologie

Warum interessiert uns das?

- der Speicherverbrauch ist oft deutlich kritischer als die Laufzeit
- auf einen Algorithmus mit Laufzeit $O(n^2)$ kann man warten
- $O(n^2)$ Speicherverbrauch ist oft nicht praktikabel

Speicherverbrauch

Warum interessiert uns das?

- der Speicherverbrauch ist oft deutlich kritischer als die Laufzeit
- auf einen Algorithmus mit Laufzeit $O(n^2)$ kann man warten
- $O(n^2)$ Speicherverbrauch ist oft nicht praktikabel

Wie groß ist der Sweep-Line Status?

Speicherverbrauch

Karlsruher Institut für Technologie

Warum interessiert uns das?

- der Speicherverbrauch ist oft deutlich kritischer als die Laufzeit
- auf einen Algorithmus mit Laufzeit $O(n^2)$ kann man warten
- $O(n^2)$ Speicherverbrauch ist oft nicht praktikabel

Wie groß ist der Sweep-Line Status?

- enthält maximal n Strecken $\rightarrow O(n)$

Speicherverbrauch

Warum interessiert uns das?

- der Speicherverbrauch ist oft deutlich kritischer als die Laufzeit
- auf einen Algorithmus mit Laufzeit $O(n^2)$ kann man warten
- $O(n^2)$ Speicherverbrauch ist oft nicht praktikabel

Wie groß ist der Sweep-Line Status?

- enthält maximal n Strecken $\rightarrow O(n)$

Wie groß ist die Event-Queue?

Speicherverbrauch

Karlsruher Institut für Technologie

Warum interessiert uns das?

- der Speicherverbrauch ist oft deutlich kritischer als die Laufzeit
- auf einen Algorithmus mit Laufzeit $O(n^2)$ kann man warten
- $O(n^2)$ Speicherverbrauch ist oft nicht praktikabel

Wie groß ist der Sweep-Line Status?

- enthält maximal n Strecken $\to O(n)$

Wie groß ist die Event-Queue?

- offensichtliche Schranke: $n + k$
- Schnittpunkte können vor Abarbeitung lange in der Queue sein

Speicherverbrauch

Karlsruher Institut für Technologie

Warum interessiert uns das?

- der Speicherverbrauch ist oft deutlich kritischer als die Laufzeit
- auf einen Algorithmus mit Laufzeit $O(n^2)$ kann man warten
- $O(n^2)$ Speicherverbrauch ist oft nicht praktikabel

Wie groß ist der Sweep-Line Status?

- enthält maximal n Strecken $\rightarrow O(n)$

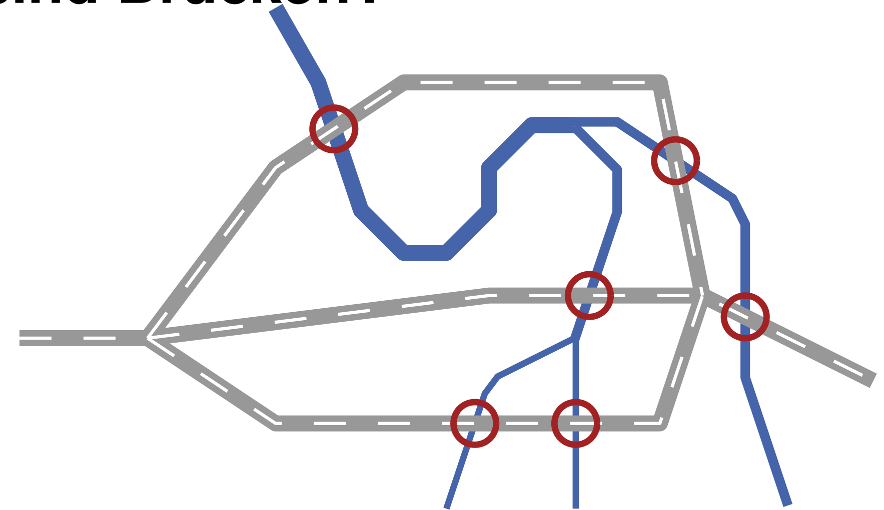
Wie groß ist die Event-Queue?

- offensichtliche Schranke: $n + k$
- Schnittpunkte können vor Abarbeitung lange in der Queue sein
- mögliche Verbesserung: halte nur Schnittpunkte in der Queue, die zu benachbarten Strecken im Sweep-Line Status gehören $\rightarrow O(n)$

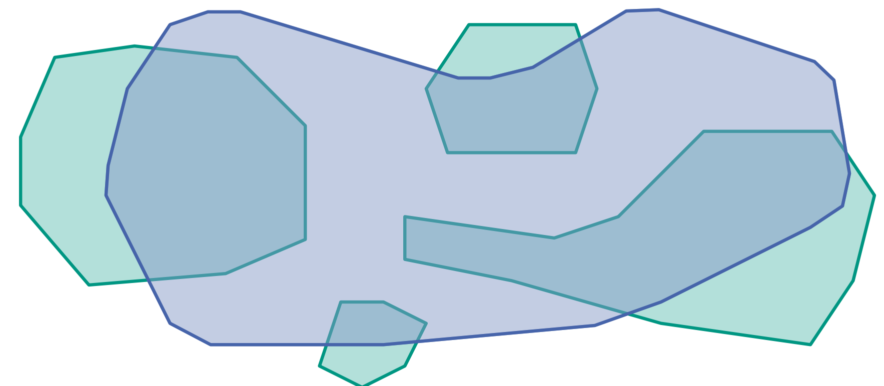
Zurück zum Anfang

Haben wir unser Ziel erreicht?

Wo sind Brücken?



Tannenwälder mit viel Niederschlag

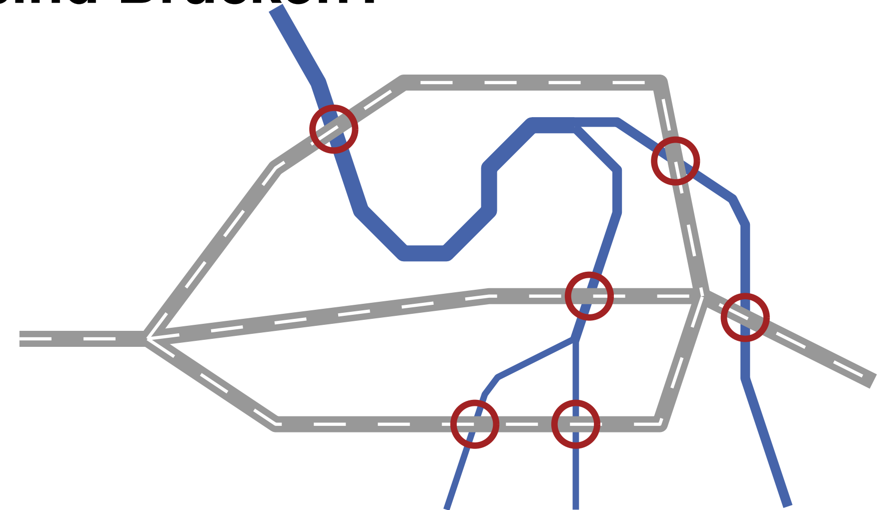


Zurück zum Anfang

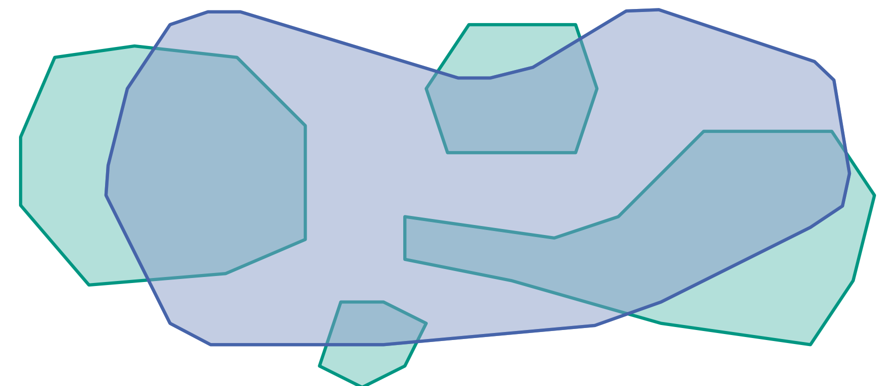
Haben wir unser Ziel erreicht?

- wir können die Brücken finden

Wo sind Brücken?



Tannenwälder mit viel Niederschlag



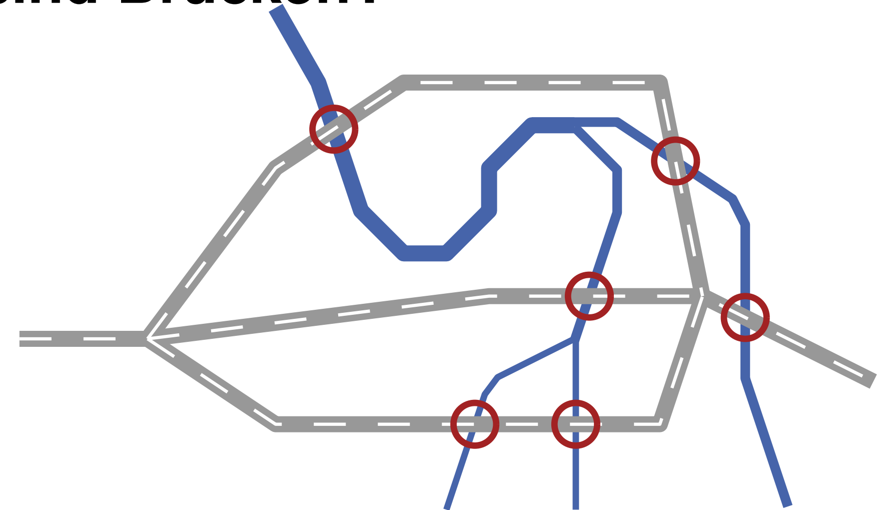
Zurück zum Anfang

Karlsruher Institut für Technologie

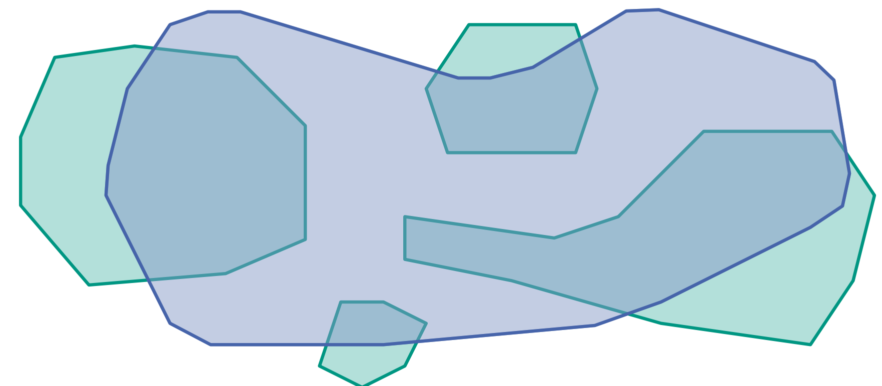
Haben wir unser Ziel erreicht?

- wir können die Brücken finden
- den Schnitt von Polygonen können wir noch nicht berechnen

Wo sind Brücken?



Tannenwälder mit viel Niederschlag



Zurück zum Anfang

Karlsruher Institut für Technologie

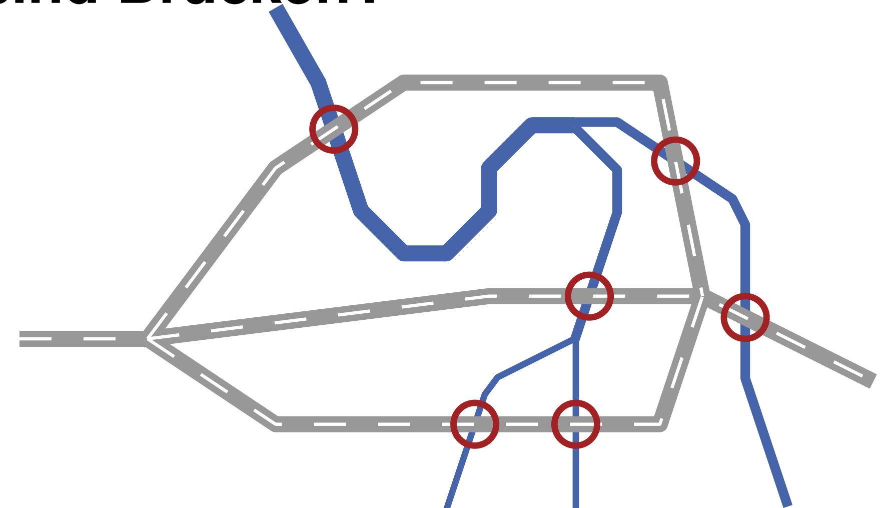
Haben wir unser Ziel erreicht?

- wir können die Brücken finden
- den Schnitt von Polygonen können wir noch nicht berechnen

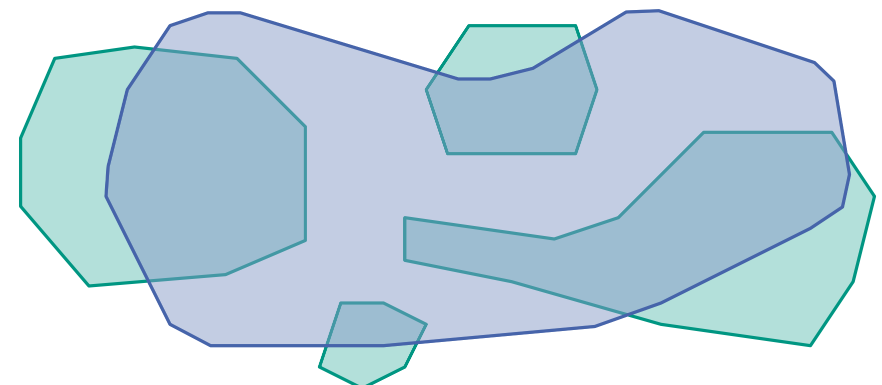
Im Folgenden

- Datenstruktur, die hilft den Schnitt von Polygonen zu berechnen
- Durchführung des Schnitts: Übung

Wo sind Brücken?



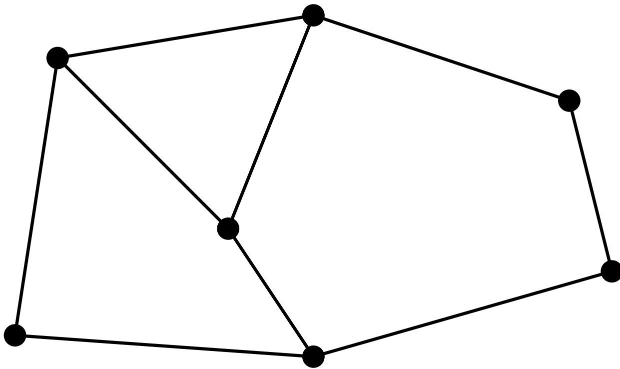
Tannenwälder mit viel Niederschlag



Doppelt-verkettete Kantenliste

Karlsruher Institut für Technologie

Bestandteile eines geometrischen Graphen

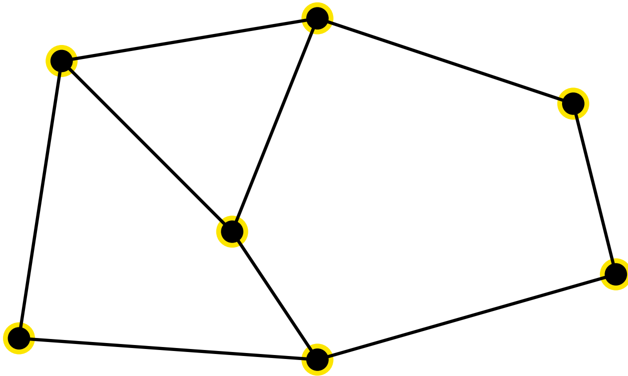


Doppelt-verkettete Kantenliste

Karlsruher Institut für Technologie

Bestandteile eines geometrischen Graphen

- Knoten mit Koordinaten

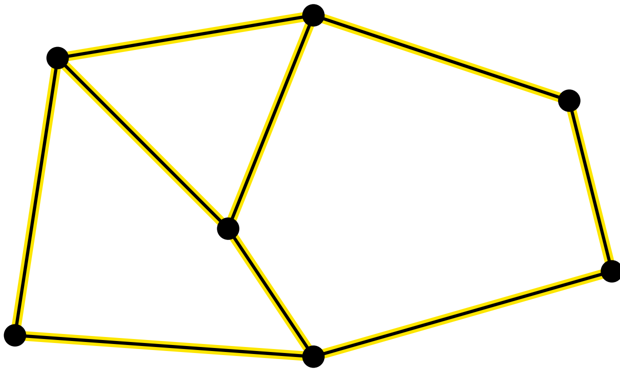


Doppelt-verkettete Kantenliste

Karlsruher Institut für Technologie

Bestandteile eines geometrischen Graphen

- Knoten mit Koordinaten
- Kanten

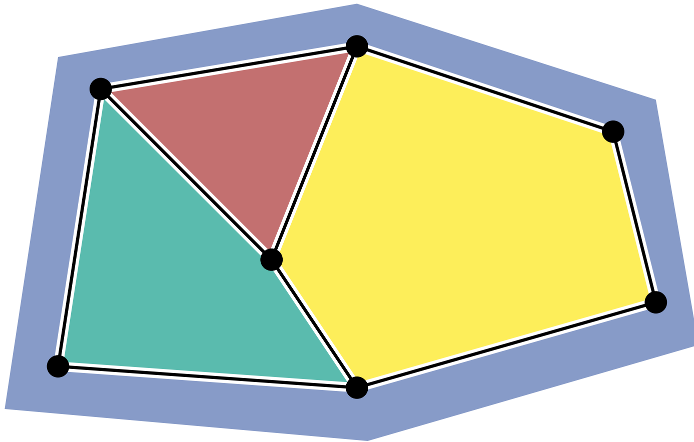


Doppelt-verkettete Kantenliste

Karlsruher Institut für Technologie

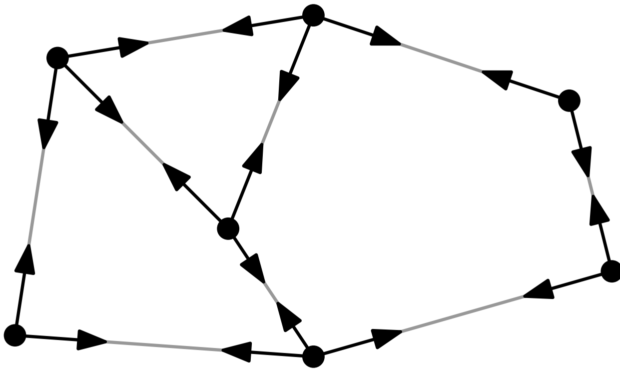
Bestandteile eines geometrischen Graphen

- Knoten mit Koordinaten
- Kanten
- Facetten



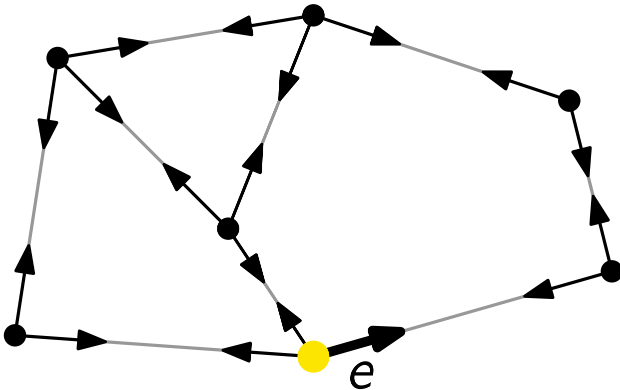
Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



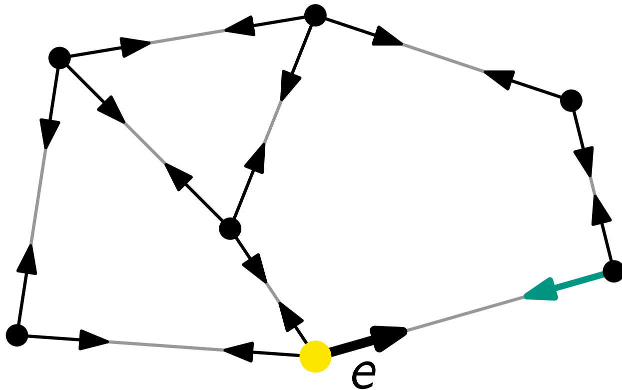
Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$

Karlsruher Institut für Technologie

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ *e*

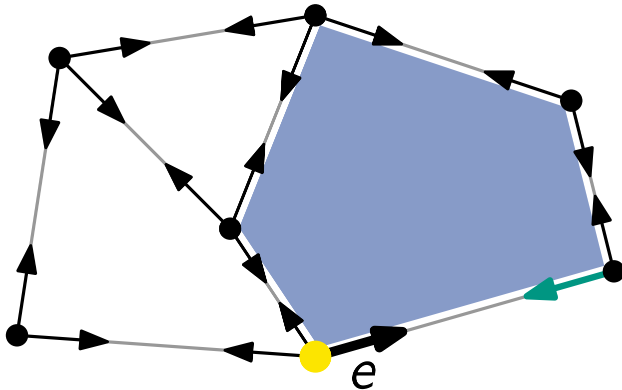
- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$

Doppelt-verkettete Kantenliste

Karlsruher Institut für Technologie

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



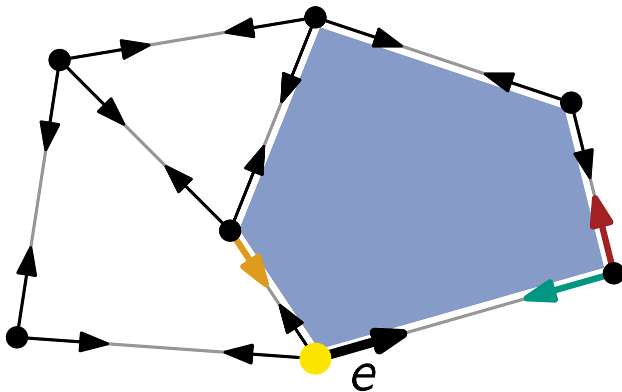
Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$

Doppelt-verkettete Kantenliste

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



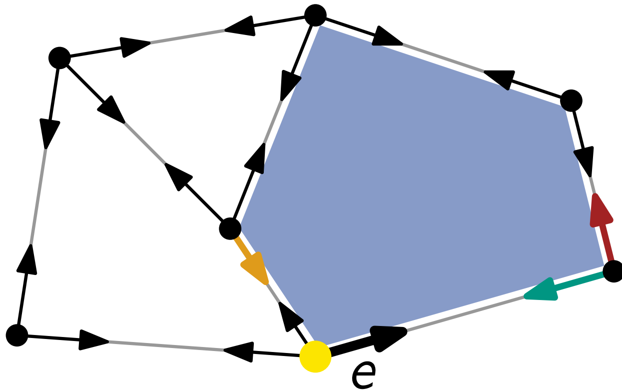
Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\texttt{next}(e)$, $\texttt{prev}(e)$

Doppelt-verkettete Kantenliste

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\texttt{next}(e)$, $\texttt{prev}(e)$

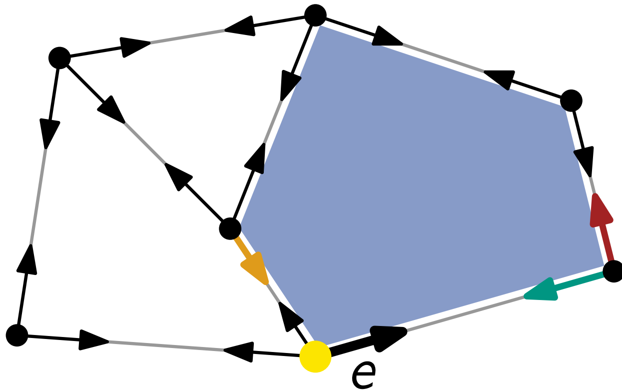
- für Knoten v / Facette f eine angrenzende Kante $\texttt{edge}(v)$ / $\texttt{edge}(f)$

Doppelt-verkettete Kantenliste

Karlsruher Institut für Technologie

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\texttt{next}(e)$, $\texttt{prev}(e)$

- für Knoten v / Facette f eine angrenzende Kante $\texttt{edge}(v)$ / $\texttt{edge}(f)$

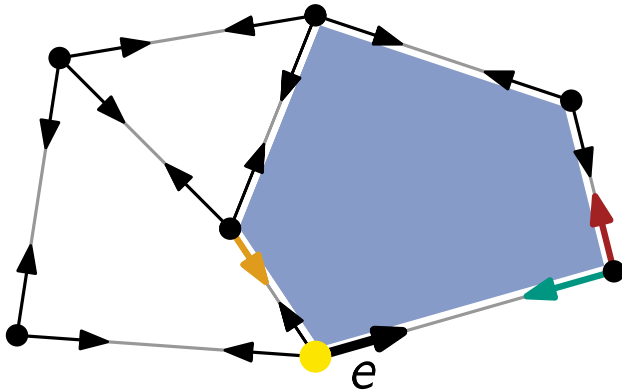
Abgeleitete Operationen

- im Uhrzeigersinn nächste Kante nach e um v:

Doppelt-verkettete Kantenliste

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\texttt{next}(e)$, $\texttt{prev}(e)$

- für Knoten v / Facette f eine angrenzende Kante $\texttt{edge}(v)$ / $\texttt{edge}(f)$

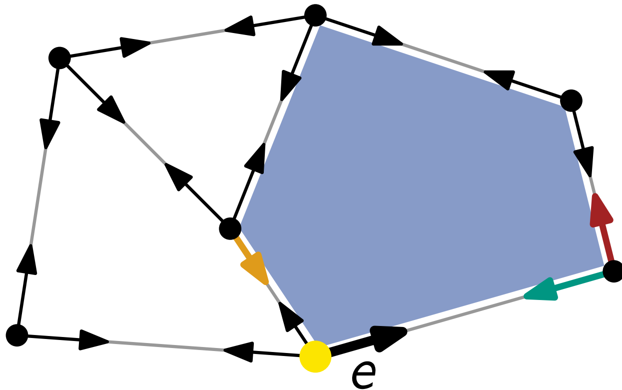
Abgeleitete Operationen

- im Uhrzeigersinn nächste Kante nach e um v: $\texttt{next}(\texttt{twin}(e))$

Karlsruher Institut für Technologie

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\texttt{next}(e)$, $\texttt{prev}(e)$

- für Knoten v / Facette f eine angrenzende Kante $\texttt{edge}(v)$ / $\texttt{edge}(f)$

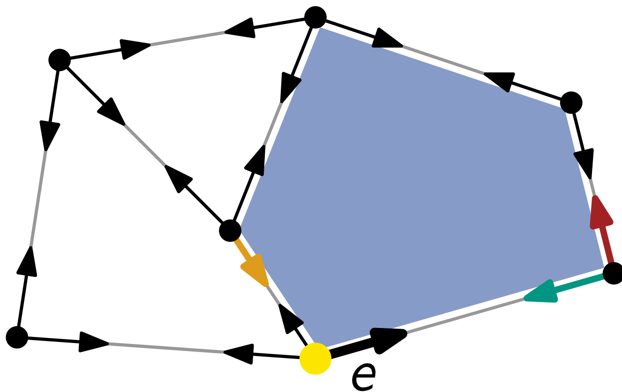
Abgeleitete Operationen

- im Uhrzeigersinn nächste Kante nach e um v: $\texttt{next}(\texttt{twin}(e))$
- gegen den Uhrzeigersinn nächste Kante nach e um v:

Doppelt-verkettete Kantenliste

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ *e*

- zugehöriger Knoten: $\mathtt{origin}(e)$
- Kante am anderen Endpunkt: $\mathtt{twin}(e)$
- inzidente Facette (links): $\mathtt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\mathtt{next}(e)$, $\mathtt{prev}(e)$

- für Knoten v / Facette f eine angrenzende Kante $\mathtt{edge}(v)$ / $\mathtt{edge}(f)$

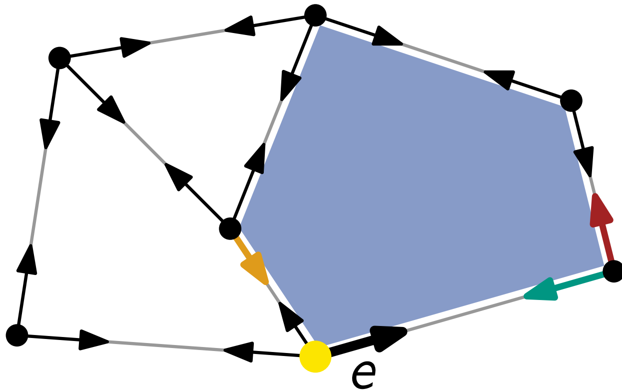
Abgeleitete Operationen

- im Uhrzeigersinn nächste Kante nach e um v: $\mathtt{next}(\mathtt{twin}(e))$
- gegen den Uhrzeigersinn nächste Kante nach e um v: $\mathtt{twin}(\mathtt{prev}(e))$

Karlsruher Institut für Technologie

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten → speichere jede Kante doppelt



Für jede „Halbkante“ e

- zugehöriger Knoten: origin(e)
- Kante am anderen Endpunkt: twin(e)
- inzidente Facette (links): face(e)
- nächste/vorherige Kante dieser Facette: next(e), prev(e)

- für Knoten v / Facette f eine angrenzende Kante edge(v) / edge(f)

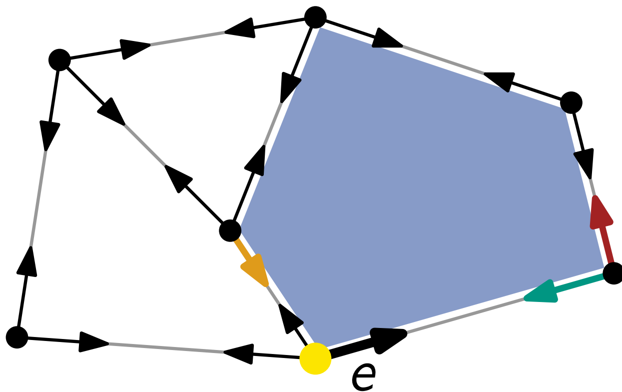
Abgeleitete Operationen

- im Uhrzeigersinn nächste Kante nach e um v: next(twin(e))
- gegen den Uhrzeigersinn nächste Kante nach e um v: twin(prev(e))
- Facette rechts von e:

Doppelt-verkettete Kantenliste

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\texttt{next}(e)$, $\texttt{prev}(e)$

- für Knoten v / Facette f eine angrenzende Kante $\texttt{edge}(v)$ / $\texttt{edge}(f)$

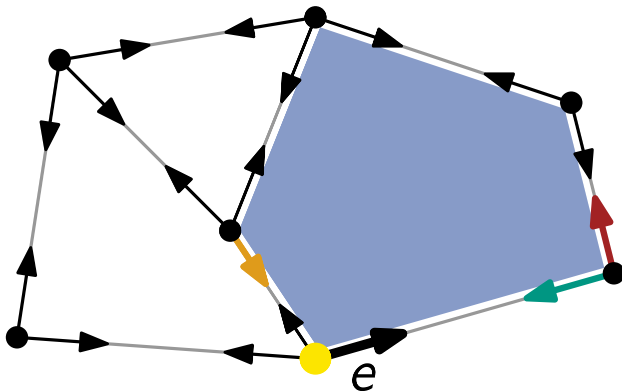
Abgeleitete Operationen

- im Uhrzeigersinn nächste Kante nach e um v: $\texttt{next}(\texttt{twin}(e))$
- gegen den Uhrzeigersinn nächste Kante nach e um v: $\texttt{twin}(\texttt{prev}(e))$
- Facette rechts von e: $\texttt{face}(\texttt{twin}(e))$

Doppelt-verkettete Kantenliste

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ *e*

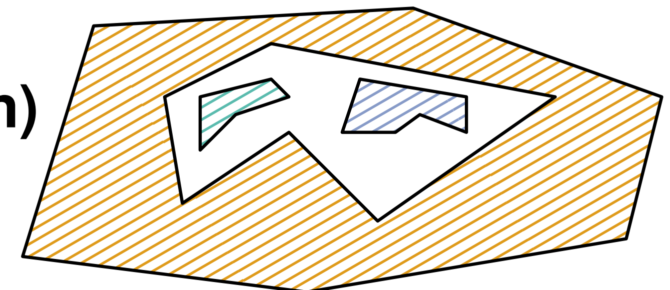
- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\texttt{next}(e)$, $\texttt{prev}(e)$

- für Knoten v / Facette f eine angrenzende Kante $\texttt{edge}(v)$ / $\texttt{edge}(f)$

Abgeleitete Operationen

- im Uhrzeigersinn nächste Kante nach e um v: $\texttt{next}(\texttt{twin}(e))$
- gegen den Uhrzeigersinn nächste Kante nach e um v: $\texttt{twin}(\texttt{prev}(e))$
- Facette rechts von e: $\texttt{face}(\texttt{twin}(e))$

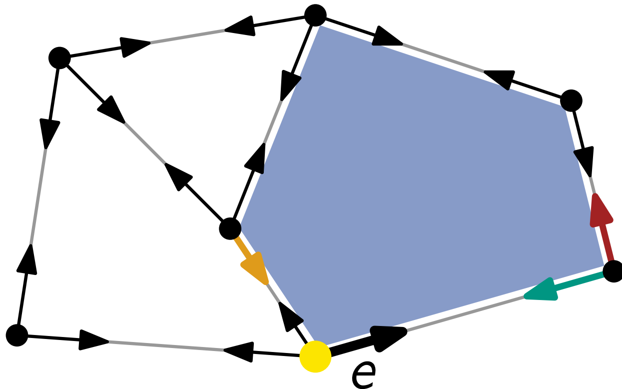
Für jede Facette *f* (bei mehreren Komponenten)



Doppelt-verkettete Kantenliste

Doppelt-verkettete Kantenliste

- jede Kante hat 2 inzidente Knoten $\rightarrow$ speichere jede Kante doppelt



Für jede „Halbkante“ e

- zugehöriger Knoten: $\texttt{origin}(e)$
- Kante am anderen Endpunkt: $\texttt{twin}(e)$
- inzidente Facette (links): $\texttt{face}(e)$
- nächste/vorherige Kante dieser Facette: $\texttt{next}(e)$, $\texttt{prev}(e)$

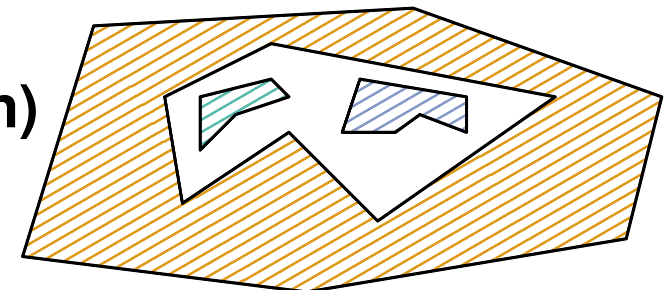
- für Knoten v / Facette f eine angrenzende Kante $\texttt{edge}(v)$ / $\texttt{edge}(f)$

Abgeleitete Operationen

- im Uhrzeigersinn nächste Kante nach e um v: $\texttt{next}(\texttt{twin}(e))$
- gegen den Uhrzeigersinn nächste Kante nach e um v: $\texttt{twin}(\texttt{prev}(e))$
- Facette rechts von e: $\texttt{face}(\texttt{twin}(e))$

Für jede Facette f (bei mehreren Komponenten)

- Liste $\texttt{children}(f)$ von Kindfacetten
- Elternfacette $\texttt{parent}(f)$



Zusamenfassung

Karlsruher Institut für Technologie

Heute gesehen

- Ausgabesensitiver Algorithmus Schnitt von n Strecken: Laufzeit $O((n + k) \log n)$ bei k Schnittpunkten

Zusamenfassung

Karlsruher Institut für Technologie

Heute gesehen

- Ausgabesensitiver Algorithmus Schnitt von n Strecken: Laufzeit $O((n + k) \log n)$ bei k Schnittpunkten
- Sweep-Line als allgemeine Technik

Zusamenfassung

Karlsruher Institut für Technologie

Heute gesehen

- Ausgabesensitiver Algorithmus Schnitt von n Strecken: Laufzeit $O((n + k) \log n)$ bei k Schnittpunkten
- Sweep-Line als allgemeine Technik
- initiale Vereinfachung durch Ausschließen von Sonderfällen lohnt sich

Zusamenfassung

Karlsruher Institut für Technologie

Heute gesehen

- Ausgabesensitiver Algorithmus Schnitt von n Strecken: Laufzeit $O((n + k) \log n)$ bei k Schnittpunkten
- Sweep-Line als allgemeine Technik
- initiale Vereinfachung durch Ausschließen von Sonderfällen lohnt sich
- doppelt-verkettete Kantenliste

Zusamenfassung

Karlsruher Institut für Technologie

Heute gesehen

- Ausgabesensitiver Algorithmus Schnitt von n Strecken: Laufzeit $O((n + k) \log n)$ bei k Schnittpunkten
- Sweep-Line als allgemeine Technik
- initiale Vereinfachung durch Ausschließen von Sonderfällen lohnt sich
- doppelt-verkettete Kantenliste

Was gibt es sonst noch?

- Erweiterung auf „Map-Overlay“: Überlagerung geometrischer Graphen
- Boolesche Operationen auf Polygonen

Zusammenfassung

Karlsruher Institut für Technologie

Heute gesehen

- Ausgabesensitiver Algorithmus Schnitt von n Strecken: Laufzeit $O((n + k) \log n)$ bei k Schnittpunkten
- Sweep-Line als allgemeine Technik
- initiale Vereinfachung durch Ausschließen von Sonderfällen lohnt sich
- doppelt-verkettete Kantenliste

Was gibt es sonst noch?

- Erweiterung auf „Map-Overlay“: Überlagerung geometrischer Graphen
- Boolesche Operationen auf Polygonen
- untere Schranke: $\Omega(n \log n + k)$
- kann tatsächlich in $O(n \log n + k)$ Zeit mit $O(n)$ Platzbedarf gelöst werden

Zusamenfassung

Karlsruher Institut für Technologie

Heute gesehen

- Ausgabesensitiver Algorithmus Schnitt von n Strecken: Laufzeit $O((n + k) \log n)$ bei k Schnittpunkten
- Sweep-Line als allgemeine Technik
- initiale Vereinfachung durch Ausschließen von Sonderfällen lohnt sich
- doppelt-verkettete Kantenliste

Was gibt es sonst noch?

- Erweiterung auf „Map-Overlay“: Überlagerung geometrischer Graphen
- Boolesche Operationen auf Polygonen
- untere Schranke: $\Omega(n \log n + k)$
- kann tatsächlich in $O(n \log n + k)$ Zeit mit $O(n)$ Platzbedarf gelöst werden
- Erweiterungen für den Sweep-Line-Ansatz
  - die Sweep-Line kann sich auch anders bewegen (z.B. rotieren)
  - die Sweep-Line muss keine Gerade sein