Karlsruher Institut für Technologie

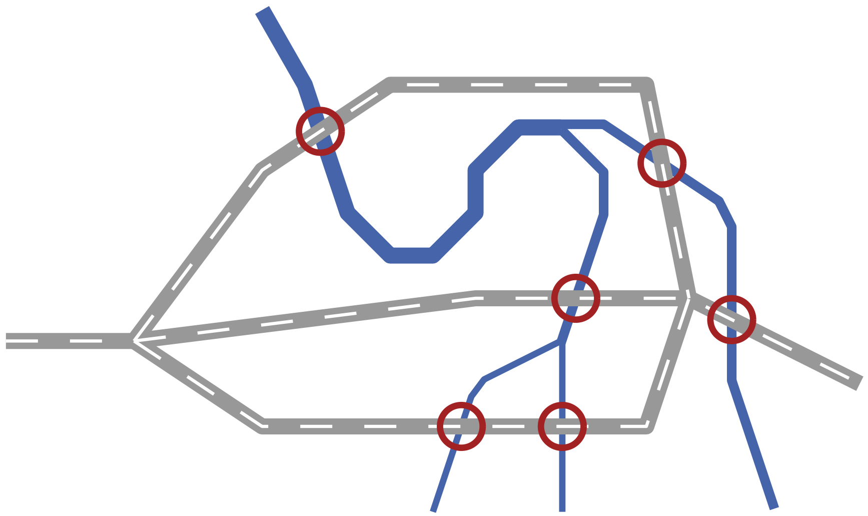
Algorithmische Geometrie

Linienschnitt

Linienschnitt: Motivation

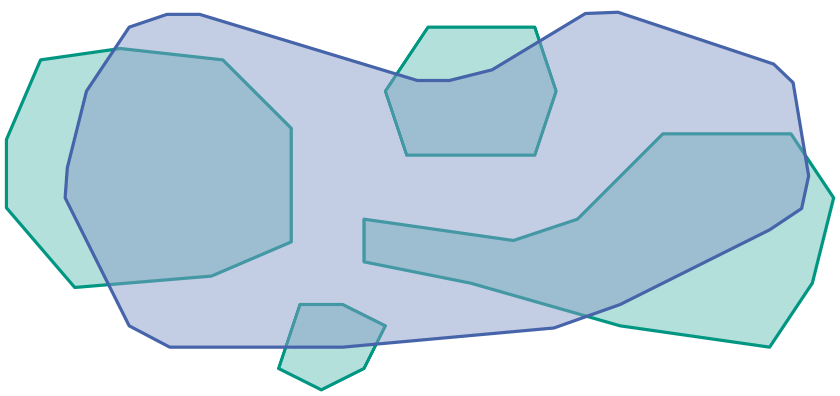
Wo sind Brücken?

- **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



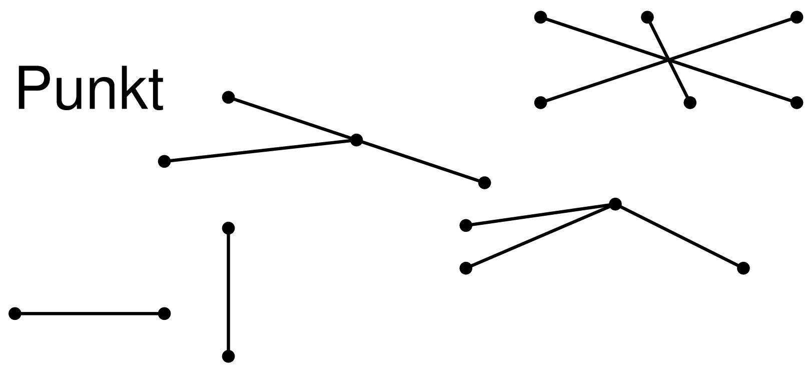
Problem: Linienschnitt
Gegeben n Strecken, berechne alle Schnittpunkte zwischen ihnen.

Erste Beobachtungen

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

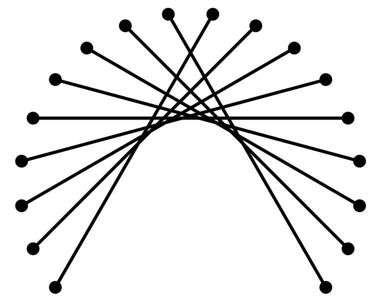
Annahme: allgemeine Lage

- 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



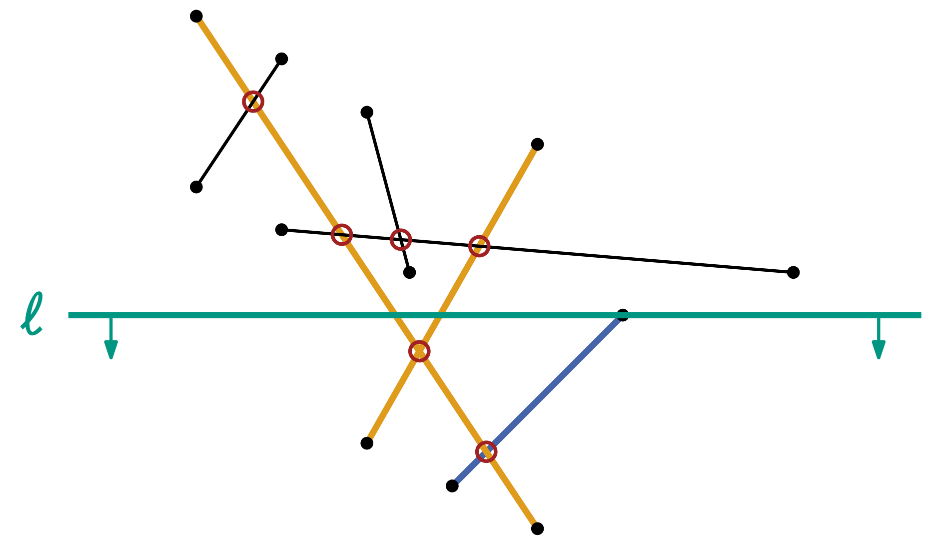
Mögliche Verbesserung (trotz unterer Schranke)

- 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
- 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

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
- **solange** $Q \neq \emptyset$
  - $p = \min\{Q\}$ und $Q = Q - p$
  - BehandleEventPunkt(p) // Event-Handler

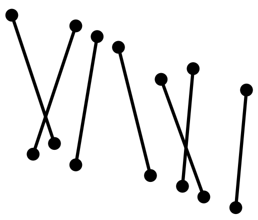
BehandleEventPunkt(p)

- $s =$ Strecke mit Endpunkt p
- **wenn** $s \in S_\ell$ // Strecke endet
  - $S_\ell = S_\ell - s$
- **sonst** // Strecke startet
  - **für alle** $s' \in S_\ell$
    - **wenn** $s \cap s' \neq \emptyset$ **gib** $(s \cap s', s, s')$ **aus**
  - $S_\ell = S_\ell + s$

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

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

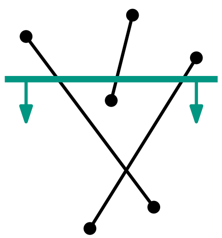
Lösung: vergleiche nur benachbarte Strecken

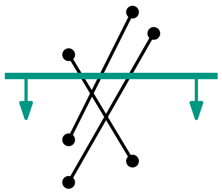


Vergleich benachbarter Strecken

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
  - Suchbaum: $O(\log n)$ (z.B. (a, b)-Baum, rot-schwarz Baum)
- Event-Queue: einfügen, Minimum finden/entfernen, suchen
  - Suchbaum: $O(\log n)$

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

Warum brauchen wir „suchen“ für die Queue?

Verbesserter Sweep-Line Algorithmus

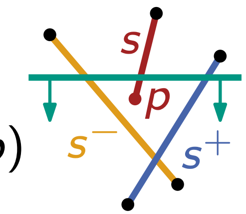
Algorithmus FINDESCHNITTPUNKTE(S)
Eingabe. Streckenmenge S
Ausgabe. alle Schnittpunkten
(mit den dazugehörigen Strecken)

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

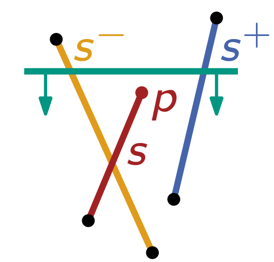
FINDENEUESEVENT(s^-, s^+, p)
  $q = s^- \cap s^+$ // Schnittpunkt
  wenn $q \neq \emptyset$ **und** $p_y < q_y$ // nach der // aktuellen // Position der // Sweep-Line
    wenn $q \notin Q$
      $Q = Q + q$

Laufzeit?

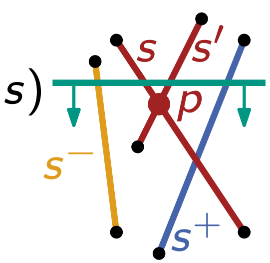
BEHANDLEEVENT(p)
wenn p ist Endpunkt einer Strecke
  $s =$ Strecke mit Endpunkt p
  wenn $s \in T$ // Streckenende
    $s^- =$ Vorgänger von s in T
    $s^+ =$ Nachfolger von s in T
    FINDENEUESEVENT(s^-, s^+, p)
    $T = T - s$



  sonst // Streckenstart
    $T = T + s$
    $s^- =$ 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)

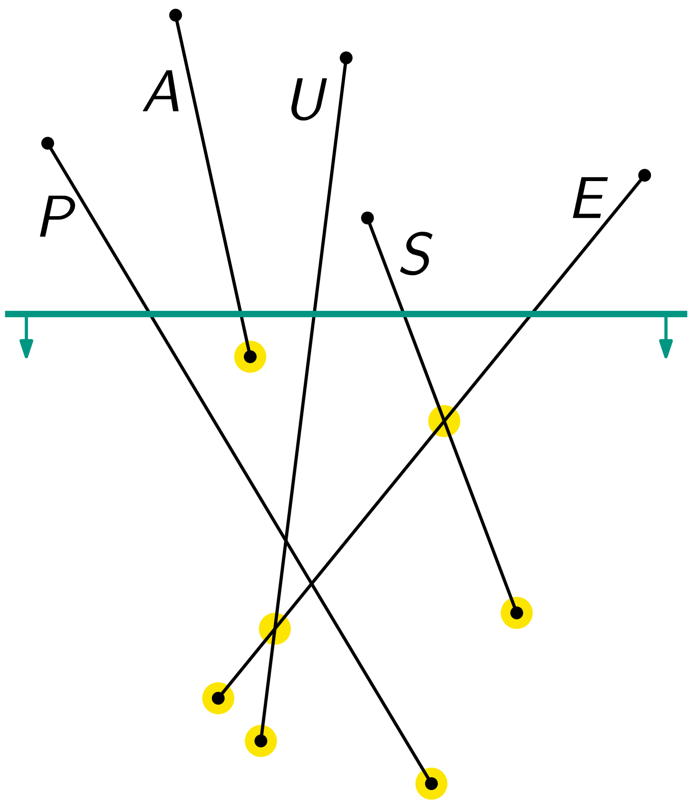


Verbesserter Sweep-Line Algorithmus

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?

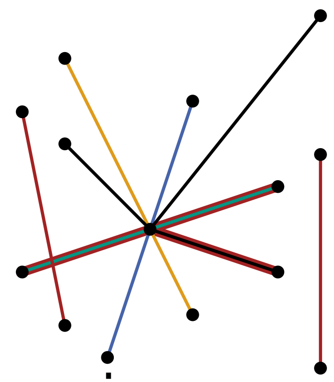


7

Robustheit

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
  - 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

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 end(p) $\cup$ schnitt(p) aus T

füge schnitt(p) $\cup$ 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 start(p) $\cup$ schnitt(p)

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

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

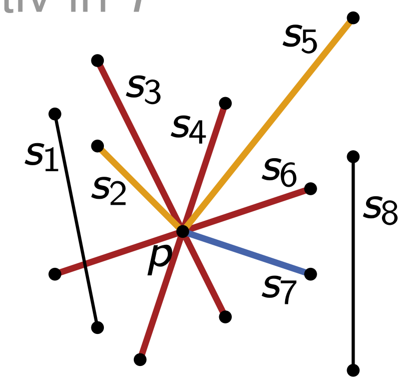
  $\hat{s}^+ =$ Nachfolger von s^+ in T

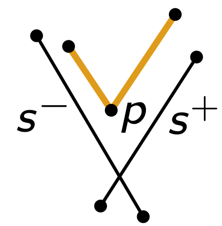
  FINDENEUESEVENT($s^-, \hat{s}^-$

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

T: s_1 s_2 s_3 s_4 s_5 s_6 s_8 $\rightarrow$ s_1 s_8

T: s_1 s_8 $\rightarrow$ 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

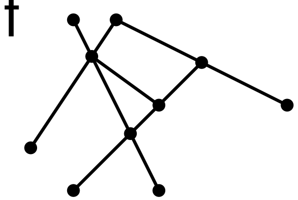
Theorem
Die k Schnittpunkte von n Strecken sowie die sich schneidenden Streckenpaaren können in $O((n+k)\log n)$ berechnet werden.

Beweis (Laufzeit)

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

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

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?

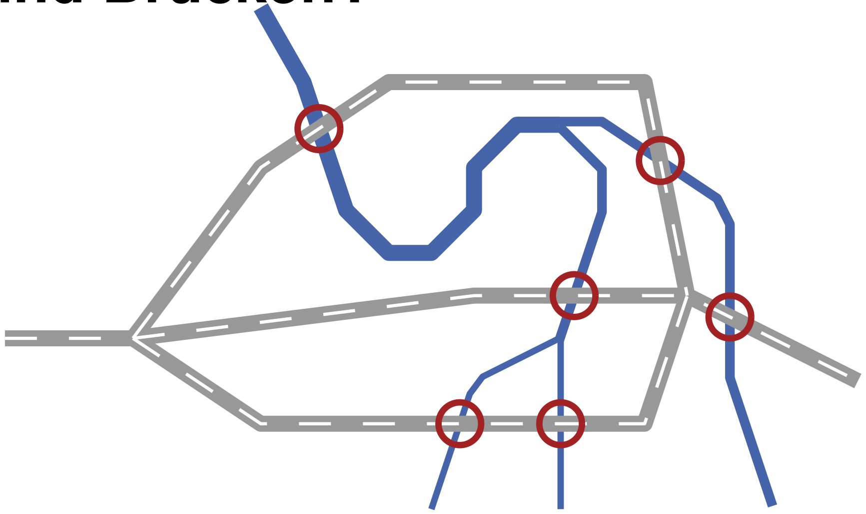
- offensichtliche Schranke: $n + k$
- Schnittpunkte können vor Abarbeitung lange in der Queue sein
- mögliche Verbesserung: halte nur Schnittpunkte in der Queue, die zu benachbarten Strecken im Sweep-Line Status gehören $\to O(n)$

Zurück zum Anfang

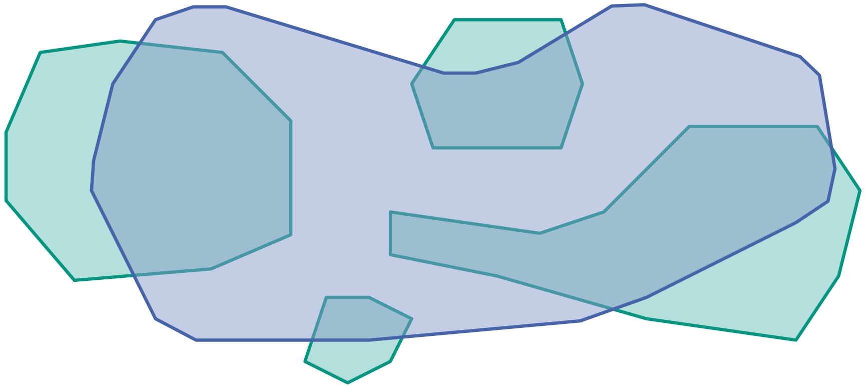
Haben wir unser Ziel erreicht?
- wir können die Brücken finden
- den Schnitt von Polygonen kön-nen wir noch nicht berechnen

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

Wo sind Brücken?



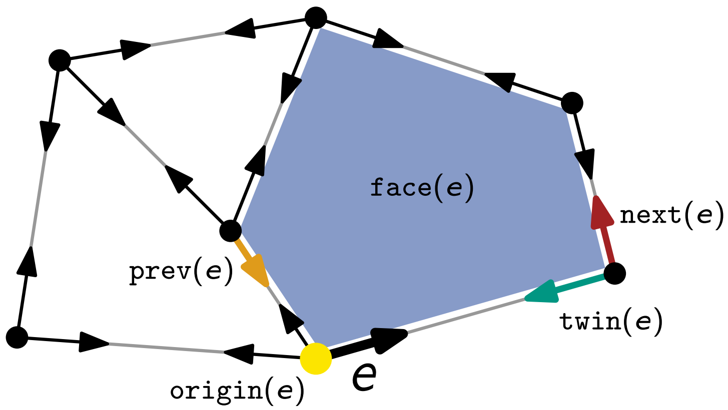
Tannenwälder mit viel Niederschlag



Doppelt-verkettete Kantenliste

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: $\texttt{next}(e)$, $\texttt{prev}(e)$

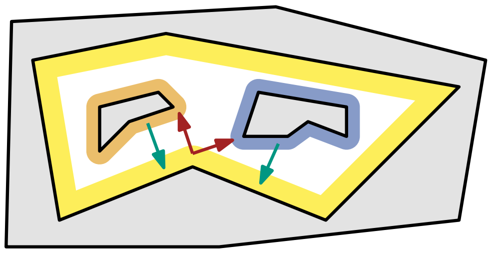
- für Knoten v / Facette f eine angrenzende Kante $\texttt{edge}(v)$ / $\texttt{edge}(f)$

Abgeleitete Operationen

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

Mehrere Zusammenhangskomponenten

- zunächst: Struktur für jede Komponenten getrennt
- innere Facette f: Liste $\texttt{children}(f)$ von Kindfacetten
- äußere Facette f: Elternfacette $\texttt{parent}(f)$



Zusammenfassung

Heute gesehen

- Ausgabesensitiver Algorithmus Schnitt von n Strecken:
  Laufzeit $O((n+k)\log n)$ bei k Schnittpunkten
- Sweep-Line als allgemeine Technik: diskretisierung des Kontinuierlichen durch endlich viele Eventpunkte
- 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