Vorlesung Algorithmische Geometrie

Polygone triangulieren

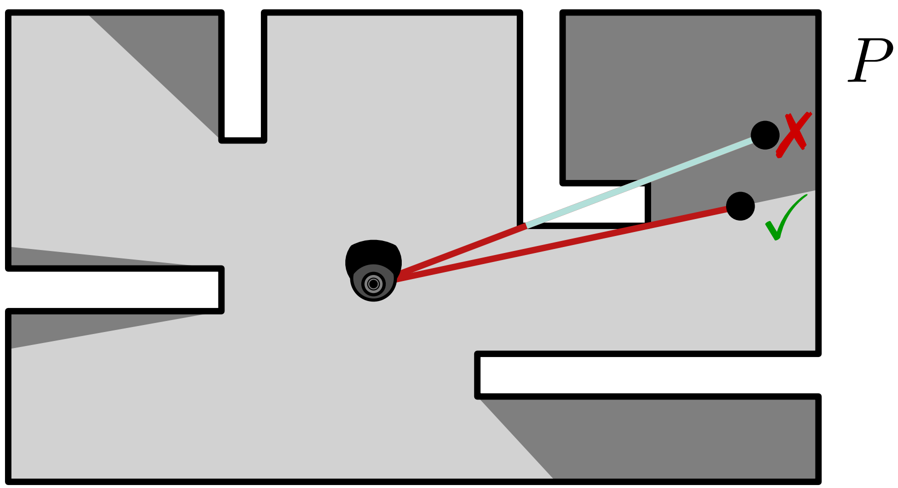
INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Martin Nöllenburg

03.05.2012

Das Kunstgalerie-Problem

Aufgabe: Installiere ein Kamerasystem zur Überwachung einer Kunstgalerie, so dass jede Stelle der Galerie gesehen wird.



Annahme: Galerie ist ein *einfaches* Polygon P mit n Ecken (keine Schnitte, keine Löcher)

Beobachtung: jede Kamera sieht sternförmiges Gebiet

Definition: Punkt $p \in P$ ist *sichtbar* von $c \in P$ wenn $\overline{cp} \in P$

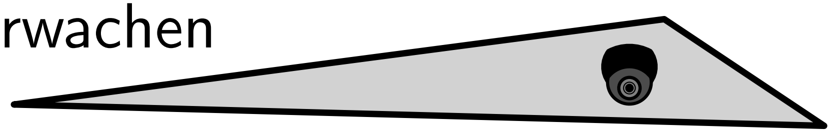
Ziel: Nutze möglichst wenige Kameras!

NP-schwer!

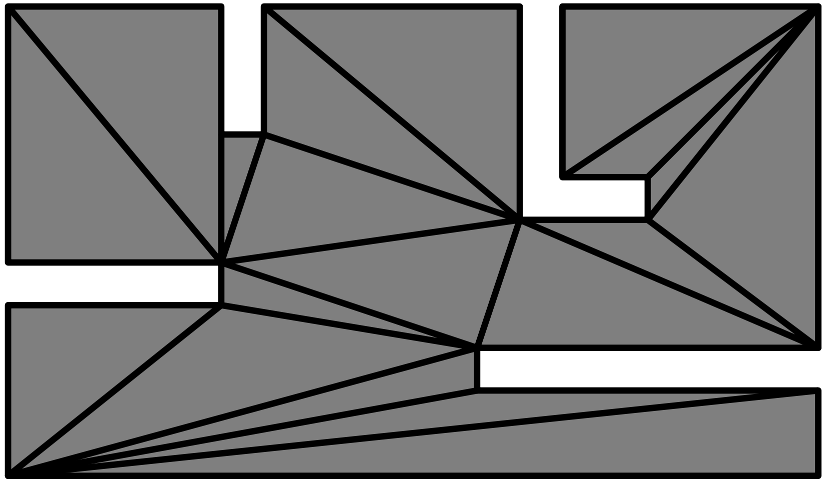
$\rightarrow$ Anzahl hängt von der Komplexität n und der Form von P ab

Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen



Idee: zerlege P in Dreiecke und überwache die Dreiecke

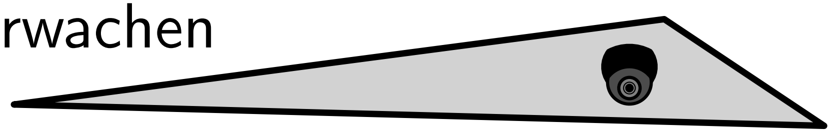


Satz 1: Jedes einfache Polygon mit n Ecken besitzt eine Triangulierung; jede Triangulierung besteht aus $n-2$ Dreiecken.

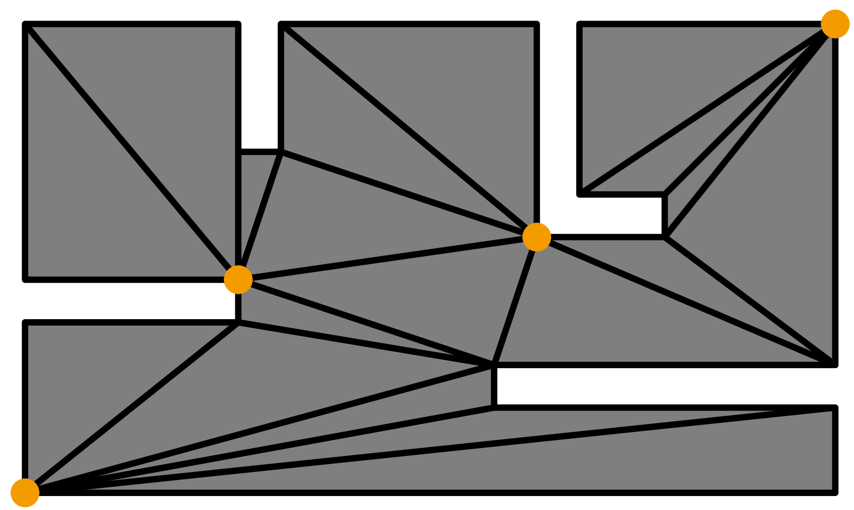
Beweis liefert rekursiven $O(n^2)$-Algorithmus!

Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen



Idee: zerlege P in Dreiecke und überwache die Dreiecke



Satz 1: Jedes einfache Polygon mit n Ecken besitzt eine Triangulierung; jede Triangulierung besteht aus $n-2$ Dreiecken.

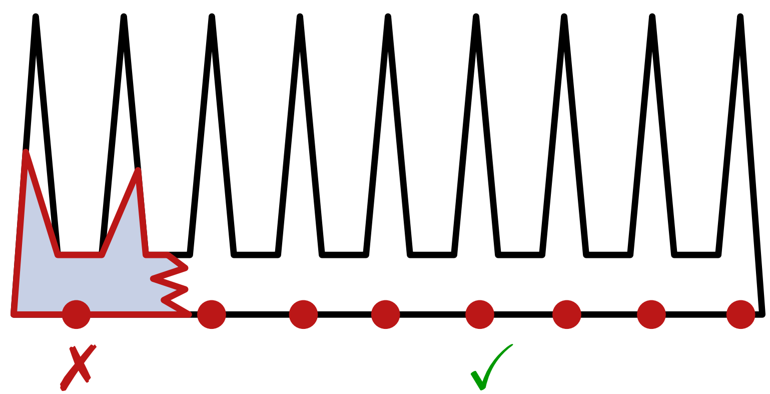
- P lässt sich mit $n-2$ Kameras in den Dreiecken überwachen
- P lässt sich mit $\approx n/2$ Kameras auf den Diagonalen überwachen
- P lässt sich mit noch weniger Kameras auf den Ecken überwachen

Das Art-Gallery-Theorem [Chvátal '75]

Satz 2: Für ein einfaches Polygon P mit n Ecken sind manchmal $\lfloor n/3 \rfloor$ Kameras nötig, aber immer ausreichend um P zu überwachen.

Beweis:

- Finde einfaches Polygon für beliebiges n, das $\approx n/3$ Kameras braucht!



- Teil 2 an der Tafel.

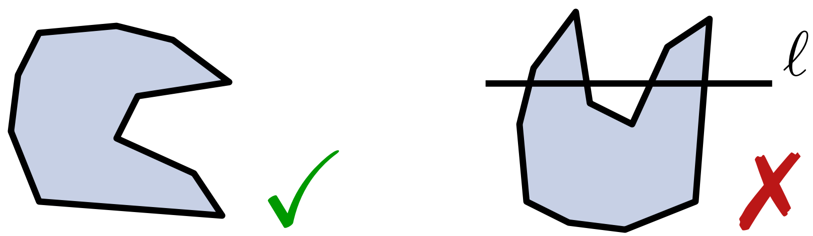
Fazit: Hat man eine Triangulierung, lassen sich $\lfloor n/3 \rfloor$ Kameras in $O(n)$ Zeit platzieren.

Triangulierung: Überblick

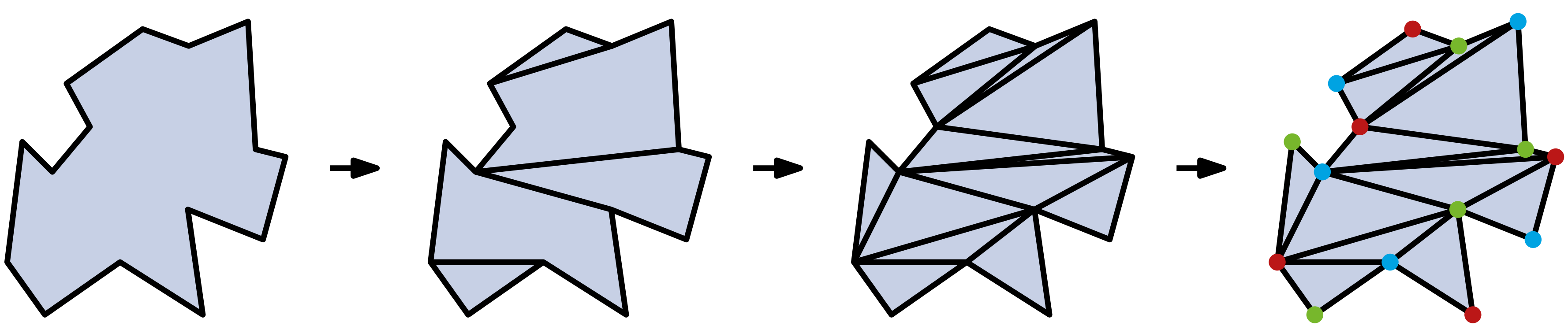
Dreistufiges Verfahren:

- Schritt 1: Zerlege P in y-monotone Teilpolygone

  Definition: Ein Polygon P ist y-monoton, falls der Schnitt $\ell \cap P$ für jede horizontale Gerade ℓ zusammenhängend ist.

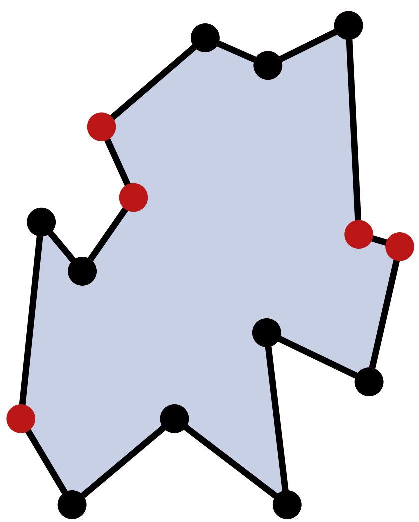


- Schritt 2: Trianguliere y-monotone Teilpolygone
- Schritt 3: benutze DFS um Triangulierung zu färben



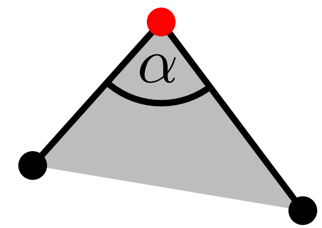
Zerlegen in y-monotone Teile

Idee: Unterscheide fünf verschiedene Knotenarten



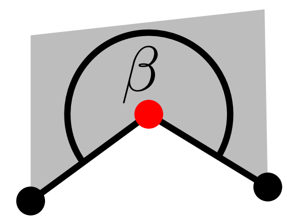
– *Wendeknoten*:
vertikale Laufrichtung wechselt

- *Startknoten*



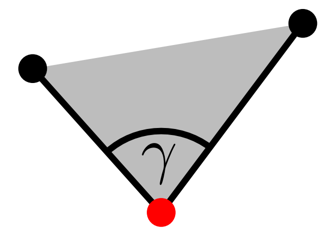
falls $\alpha < 180°$

- *Splitknoten*



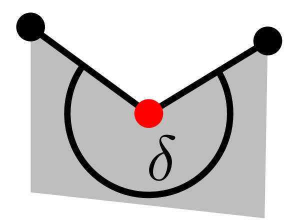
falls $\beta > 180°$

- *Endknoten*



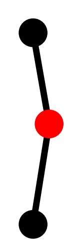
falls $\gamma < 180°$

- *Mergeknoten*



falls $\delta > 180°$

– *reguläre* Knoten

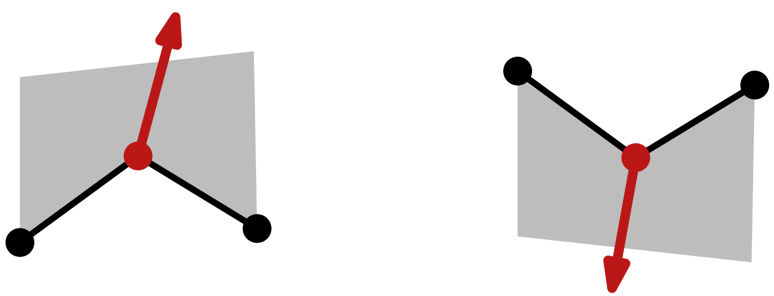


Charakterisierung

Lemma 1: Ein Polygon ist y-monoton, wenn es keine Split- oder Mergeknoten besitzt.

Beweis: an der Tafel

$\Rightarrow$ Wir müssen alle Split- und Mergeknoten durch Einfügen von Diagonalen entfernen

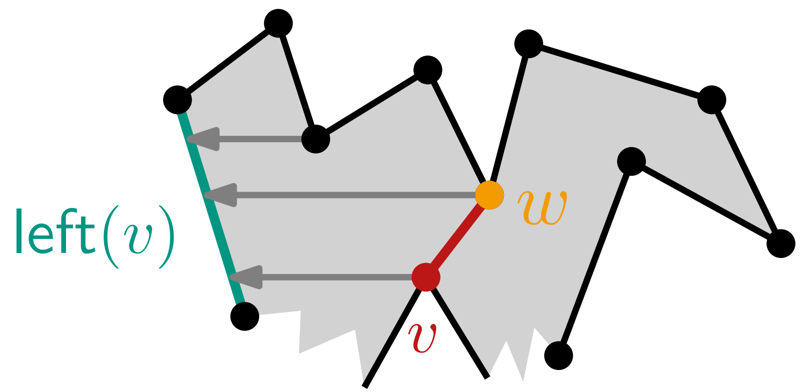


Vorsicht: Diagonalen dürfen weder Kanten von P noch andere Diagonalen schneiden

In Richtung Sweep-Line-Algorithmus

1) Diagonalen für Splitknoten

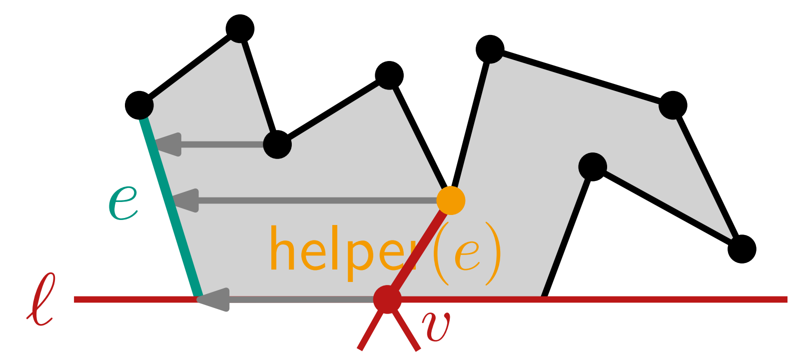
- betrachte für jeden Knoten v linke Nachbarkante left(v) bzgl. horizontaler sweep line ℓ



- verbinde Splitknoten v zu niedrigstem Knoten w oberhalb v mit left(w) = left(v)

- speichere für jede Kante e den untersten Knoten w mit left$(w) = e$ als helper(e)

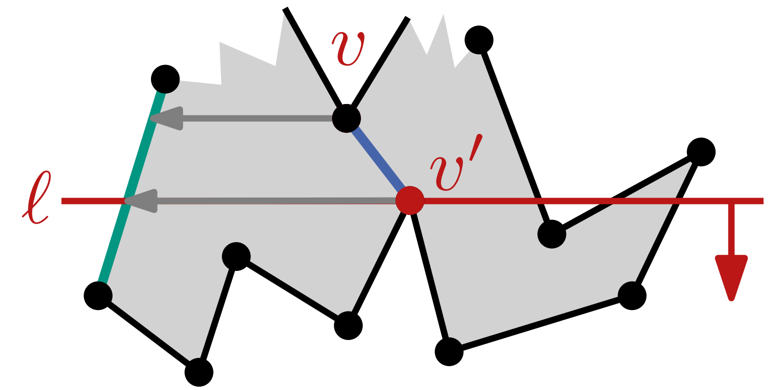
- trifft ℓ auf Splitknoten v: verbinde v mit helper(left(v))



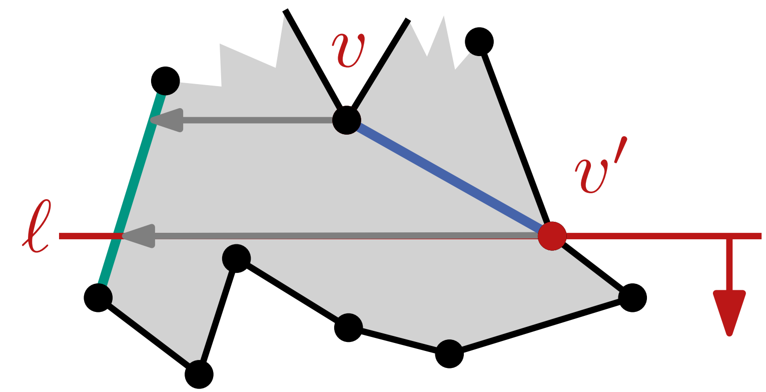
In Richtung Sweep-Line-Algorithmus

2) Diagonalen für Mergeknoten

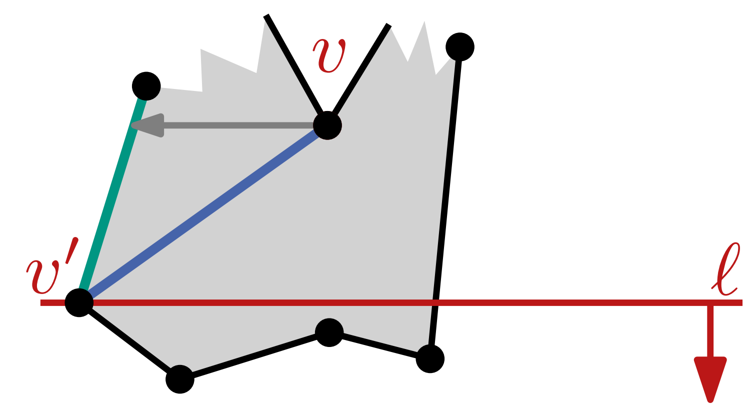
- erreicht man Mergeknoten v wird helper(left(v)) $= v$
- erreicht man Splitknoten v' mit left(v') $=$ left(v) wird Diagonale (v, v') eingefügt



- ersetzt man helper(left(v)) durch v' wird Diagonale (v, v') eingefügt



- erreicht man das Ende v' von left(v) wird Diagonale (v, v') eingefügt

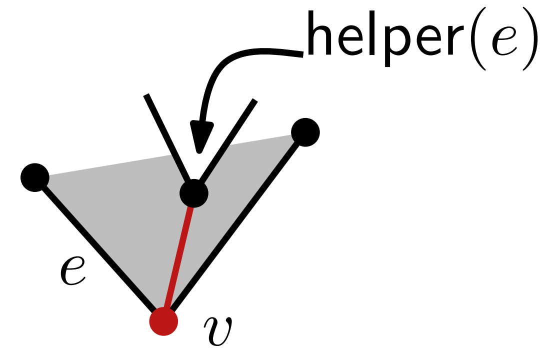


Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

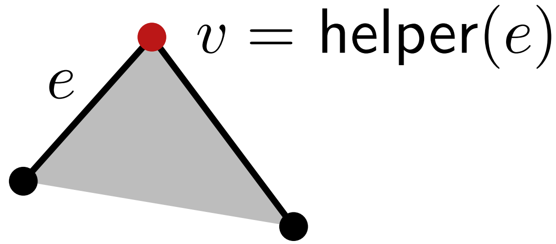
$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
  $v \leftarrow \mathcal{Q}$.nextVertex()
  $\mathcal{Q}$.deleteVertex(v)
  handleVertex(v)

return $\mathcal{D}$



handleStartVertex(vertex v)

$\mathcal{T} \leftarrow$ füge linke Kante e ein
helper(e) $\leftarrow v$



handleEndVertex(vertex v)

$e \leftarrow$ linke Kante
if isMergeVertex(helper(e)) **then**
  $\mathcal{D} \leftarrow$ füge (helper(e), v) ein
lösche e aus $\mathcal{T}$

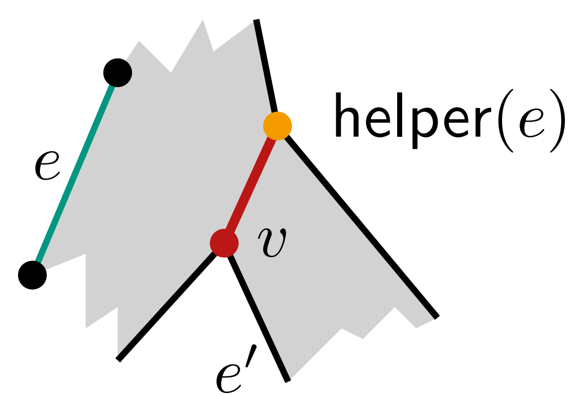
Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
  $v \leftarrow \mathcal{Q}$.nextVertex()
  $\mathcal{Q}$.deleteVertex(v)
  handleVertex(v)
return $\mathcal{D}$

handleSplitVertex(vertex v)

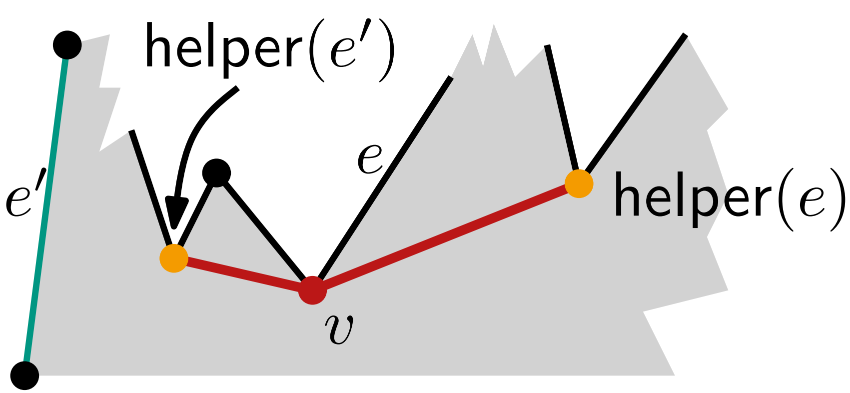
$e \leftarrow$ Kante links von v in $\mathcal{T}$
$\mathcal{D} \leftarrow$ füge (helper(e), v) ein
helper(e) $\leftarrow v$
$\mathcal{T} \leftarrow$ füge rechte Kante e' von v ein
helper(e') $\leftarrow v$



Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
while $\mathcal{Q} \neq \emptyset$ **do**
    $v \leftarrow \mathcal{Q}$.nextVertex()
    $\mathcal{Q}$.deleteVertex(v)
    handleVertex(v)
return $\mathcal{D}$



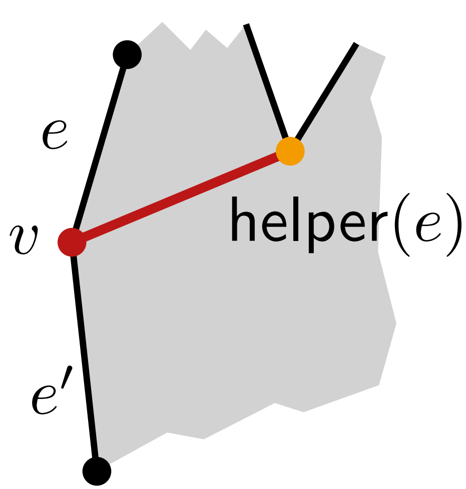
handleMergeVertex(vertex v)

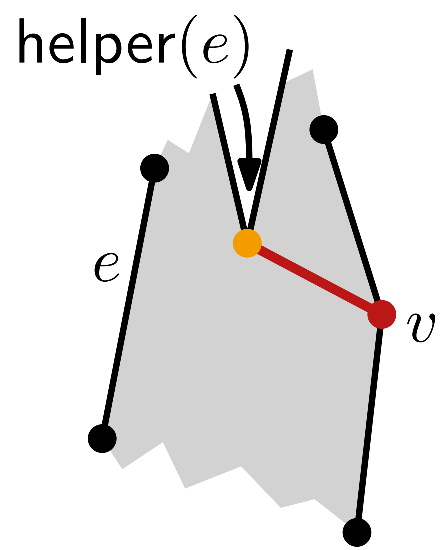
$e \leftarrow$ rechte Kante
if isMergeVertex(helper(e)) **then**
    $\mathcal{D} \leftarrow$ füge (helper(e), v) ein
lösche e aus $\mathcal{T}$
$e' \leftarrow$ Kante links von v in $\mathcal{T}$
if isMergeVertex(helper(e')) **then**
    $\mathcal{D} \leftarrow$ füge (helper(e'), v) ein
helper(e') $\leftarrow v$

Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

- $\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$
- $\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert
- $\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)
- **while** $\mathcal{Q} \neq \emptyset$ **do**
  - $v \leftarrow \mathcal{Q}$.nextVertex()
  - $\mathcal{Q}$.deleteVertex(v)
  - handleVertex(v)
- **return** $\mathcal{D}$





handleRegularVertex(vertex v)

- **if** P liegt lokal rechts von v **then**
  - $e, e' \leftarrow$ obere, untere Kante
  - **if** isMergeVertex(helper(e)) **then**
    - $\mathcal{D} \leftarrow$ füge (helper$(e), v$) ein
  - lösche e aus $\mathcal{T}$
  - $\mathcal{T} \leftarrow$ füge e' ein; helper$(e') \leftarrow v$
- **else**
  - $e \leftarrow$ Kante links von v in $\mathcal{T}$
  - **if** isMergeVertex(helper(e)) **then**
    - $\mathcal{D} \leftarrow$ füge (helper$(e), v$) ein
  - helper$(e) \leftarrow v$

Analyse

Lemma 2: Algorithmus MakeMonotone fügt eine Menge von kreuzungsfreien Diagonalen in P ein, die P in y-monotone Teilpolygone zerlegen.

Satz 3: Ein einfaches Polygon mit n Knoten kann in $O(n \log n)$ Zeit und $O(n)$ Platz in y-monotone Teilpolygone zerlegt werden.

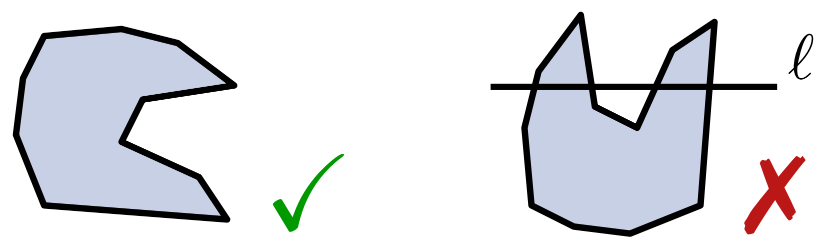
- priority queue $\mathcal{Q}$ erzeugen: $O(n)$ Zeit
- Sweep-Line Status $\mathcal{T}$ initialisieren: $O(1)$ Zeit
- Eventbehandlung pro Event: $O(\log n)$ Zeit
  - $\mathcal{Q}$.deleteMax: $O(\log n)$ Zeit
  - Element aus $\mathcal{T}$ suchen, löschen, einfügen: $O(\log n)$ Zeit
  - ≤ 2 Diagonalen in $\mathcal{D}$ einfügen: $O(1)$ Zeit
- Platz: offensichtlich $O(n)$

Triangulierung: Überblick

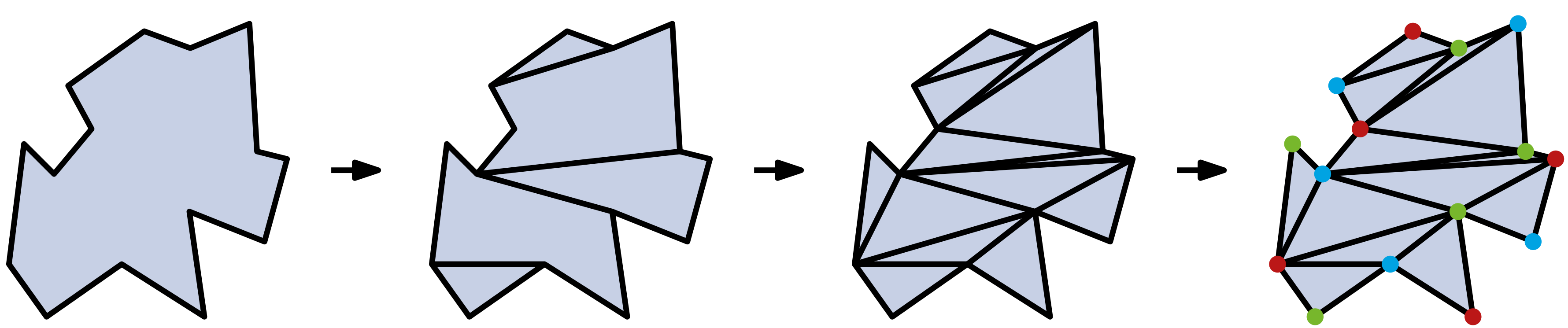
Dreistufiges Verfahren:

- Schritt 1: Zerlege P in y-monotone Teilpolygone ✓

Definition: Ein Polygon P ist y-monoton, falls der Schnitt $\ell \cap P$ für jede horizontale Gerade ℓ zusammenhängend ist.



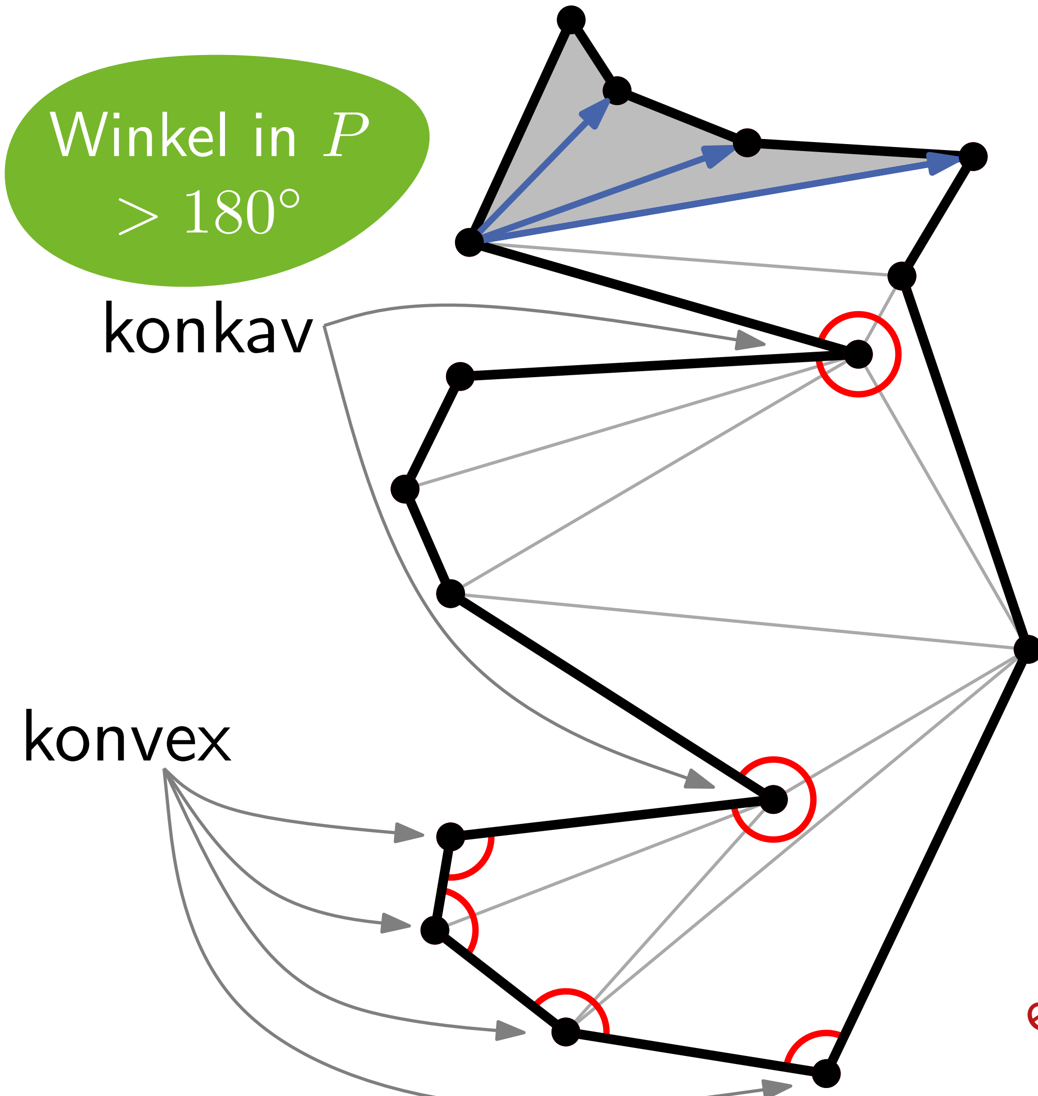
- Schritt 2: Trianguliere y-monotone Teilpolygone **ToDo!**
- Schritt 3: benutze DFS um Triangulierung zu färben ✓



Triangulieren von y-monotone Polygonen

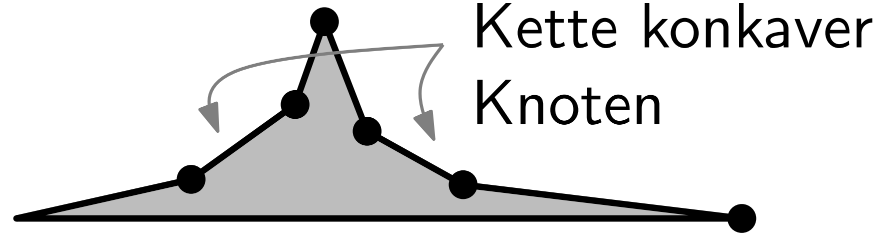
Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

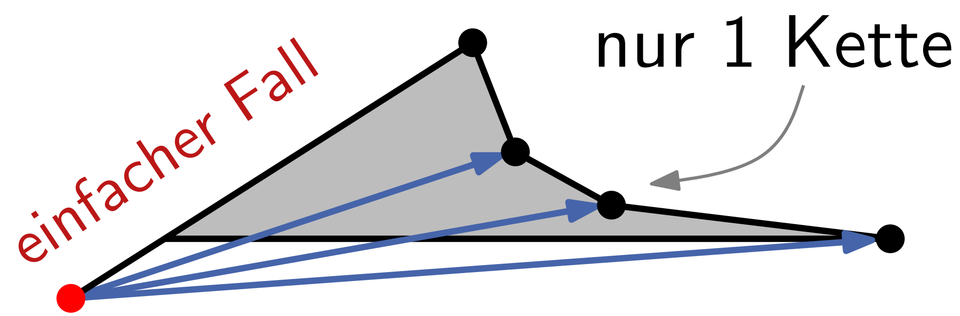


Invariante?

Der besuchte, aber noch nicht triangulierte Teil von P ist *trichterförmig*.



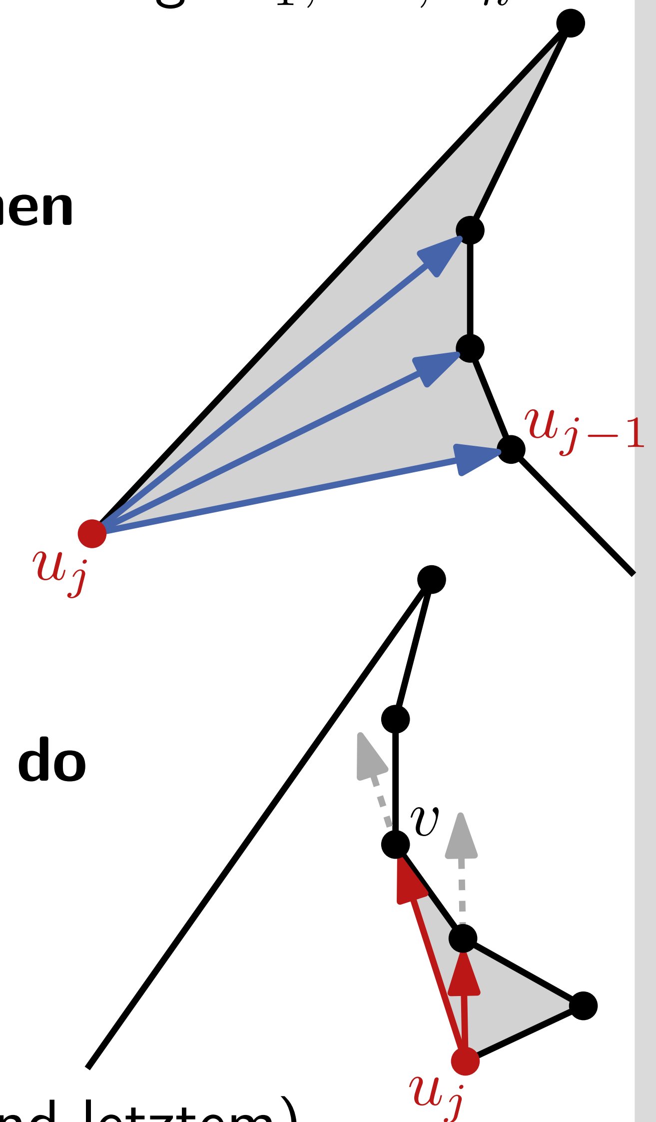
genauer hinschauen:



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

```
verschmelze linken und rechten Pfad → absteigende Folge u_1, ..., u_n
Stack S ← ∅; S.push(u_1); S.push(u_2)
for j ← 3 to n − 1 do
    if u_j und S.top() auf verschiedenen Pfaden then
        while not S.empty() do
            v ← S.pop()
            if not S.empty() then zeichne (u_j, v)
        S.push(u_{j−1}); S.push(u_j)
    else
        v ← S.pop()
        while not S.empty() and u_j sieht S.top() do
            v ← S.pop()
            zeichne Diagonale (u_j, v)
        S.push(v); S.push(u_j)
verbinde u_n zu allen Knoten in S (außer erstem und letztem)
```



Zusammenfassung

Satz 4: Ein y-monotones Polygon mit n Knoten lässt sich in $O(n)$ Zeit triangulieren.

Satz 3: Ein einfaches Polygon mit n Knoten kann in $O(n \log n)$ Zeit und $O(n)$ Platz in y-monotone Teilpolygone zerlegt werden.

(alt)

$\Downarrow$

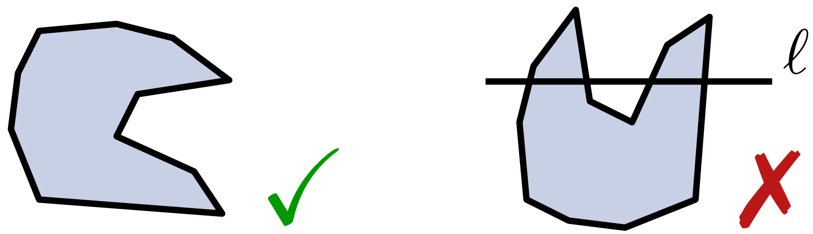
Satz 5: Ein einfaches Polygon mit n Knoten kann in $O(n \log n)$ Zeit und $O(n)$ Platz trianguliert werden.

Triangulierung: Überblick

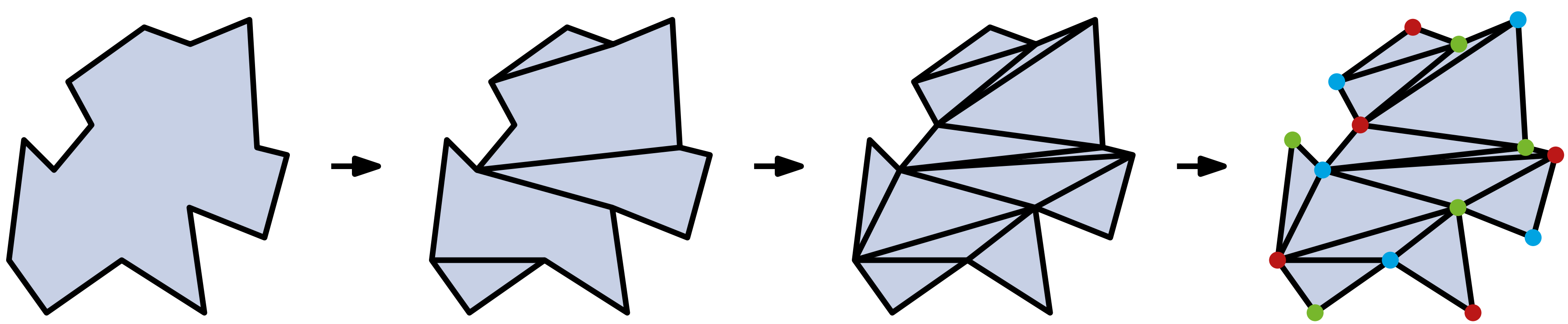
Dreistufiges Verfahren:

- Schritt 1: Zerlege P in y-monotone Teilpolygone ✓

  Definition: Ein Polygon P ist y-monoton, falls der Schnitt $\ell \cap P$ für jede horizontale Gerade ℓ zusammenhängend ist.



- Schritt 2: Trianguliere y-monotone Teilpolygone ✓
- Schritt 3: benutze DFS um Triangulierung zu färben ✓



Diskussion

Lässt sich der Triangulierungs-Algorithmus auch auf Polygone mit Löchern erweitern?

- Triangulierung: ja
- Aber reichen weiterhin $\lfloor n/3 \rfloor$ Kameras aus?
  Nein, eine Verallgemeinerung des Art-Gallery-Theorems besagt, dass manchmal $\lfloor (n+h)/3 \rfloor$ Kameras nötig, aber immer ausreichend sind, wobei h die Anzahl der Löcher ist. [Hoffmann et al., 91]

Geht es für allgemeine einfache Polygone noch schneller?

Ja. Nachdem das Problem lange offen war, und Ende der 1980er Jahre nach und nach schnellere (z.T. randomisierte) Algorithmen vorgestelt wurden, beschrieb Chazelle [1990] einen (komplizierten) deterministischen Linearzeit-Algorithmus.