

Vorlesung Algorithmische Geometrie

Polygone triangulieren

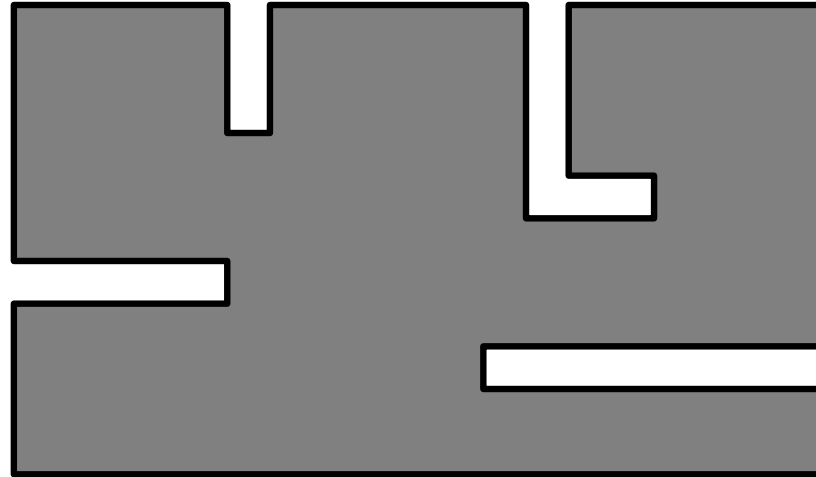
LEHRSTUHL FÜR ALGORITHMIK I · INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Martin Nöllenburg
26.04.2011



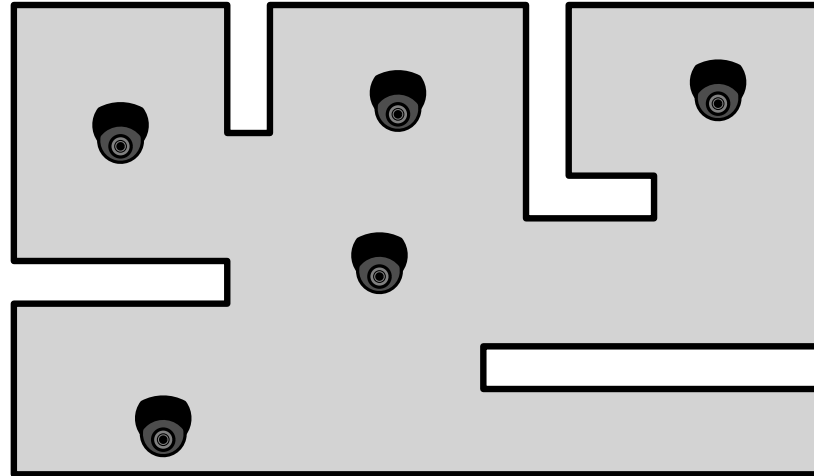
Das Kunstgalerie-Problem

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



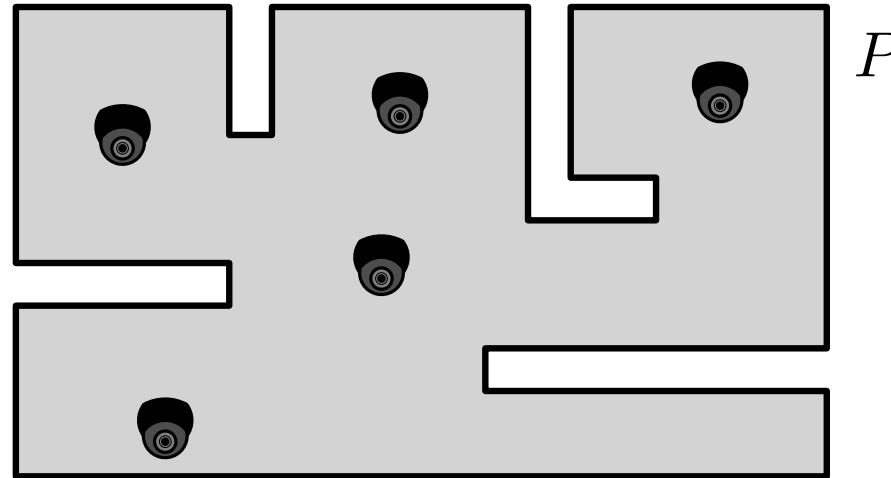
Das Kunstgalerie-Problem

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



Das Kunstgalerie-Problem

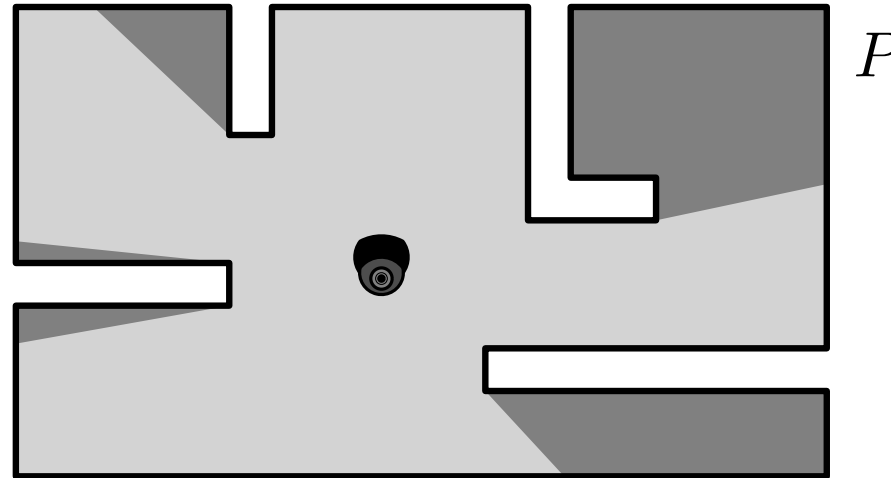
Aufgabe: Installiere ein Kamerasystem zur Überwachung einer Kunstgalerie, so dass jede Stelle der Galerie gesehen wird.



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

Das Kunstgalerie-Problem

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

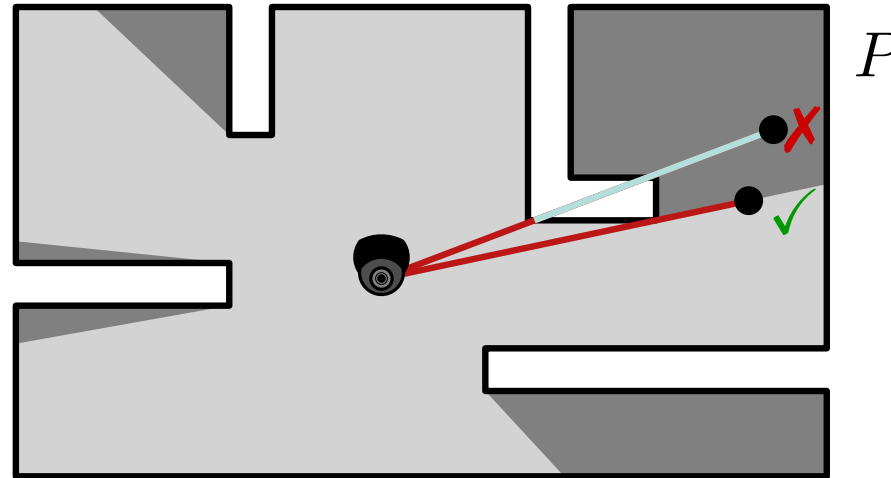


Annahme: Galerie ist ein *einfaches* Polygon P mit n Ecken
(keine Schnitte, keine Löcher)

Beobachtung: jede Kamera sieht sternförmiges Gebiet

Das Kunstgalerie-Problem

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



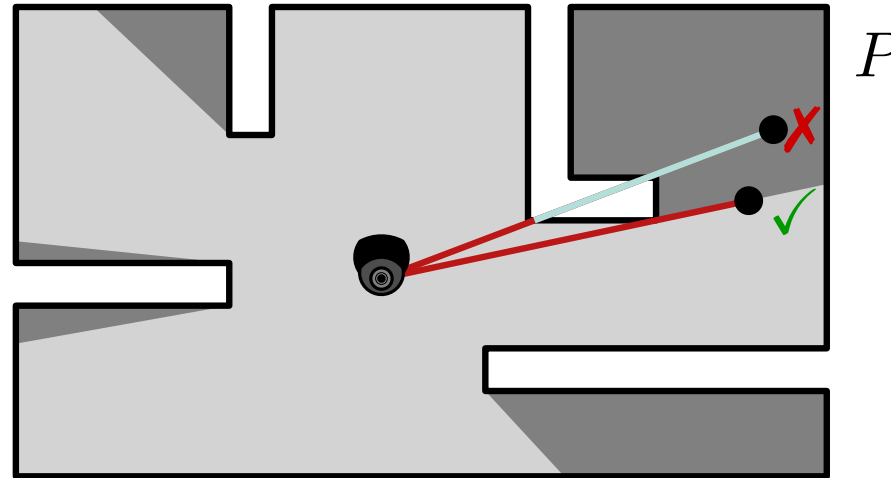
Annahme: Galerie ist ein *einfaches* Polygon P mit n Ecken
(keine Schnitte, keine Löcher)

Beobachtung: jede Kamera sieht sternförmiges Gebiet

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

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)

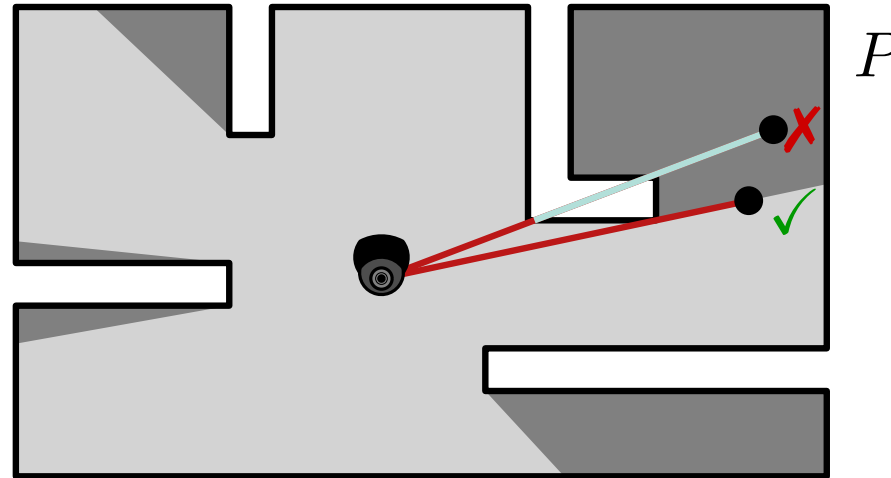
Beobachtung: jede Kamera sieht sternförmiges Gebiet

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

Ziel: Nutze möglichst wenige Kameras!

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

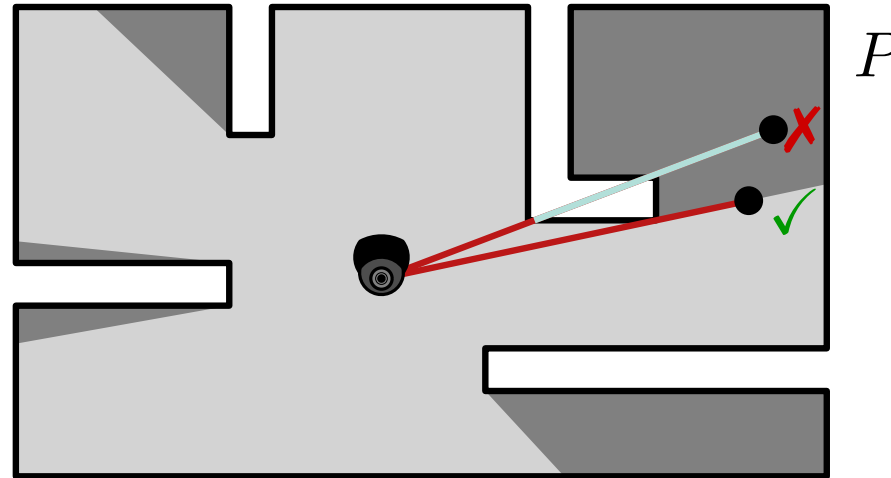
Definition: Punkt $p \in P$ ist *sichtbar* von $c \in P$ wenn $\overline{cp} \in P$

Ziel: Nutze möglichst wenige Kameras!

→ Anzahl hängt von der Komplexität n und der Form von P ab

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 $\overline{cp} \in P$

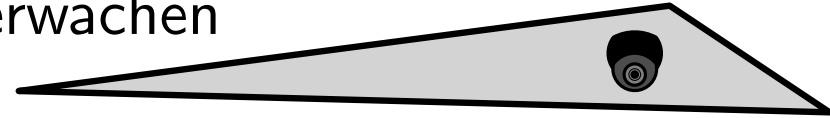
Ziel: Nutze möglichst wenige Kameras!

→ Anzahl hängt von der Komplexität n und der Form von P ab

NP-schwer!

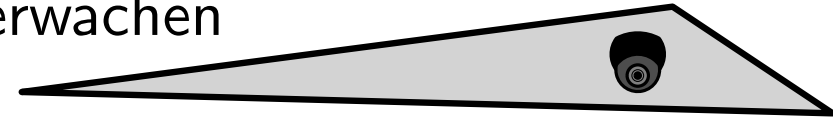
Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen

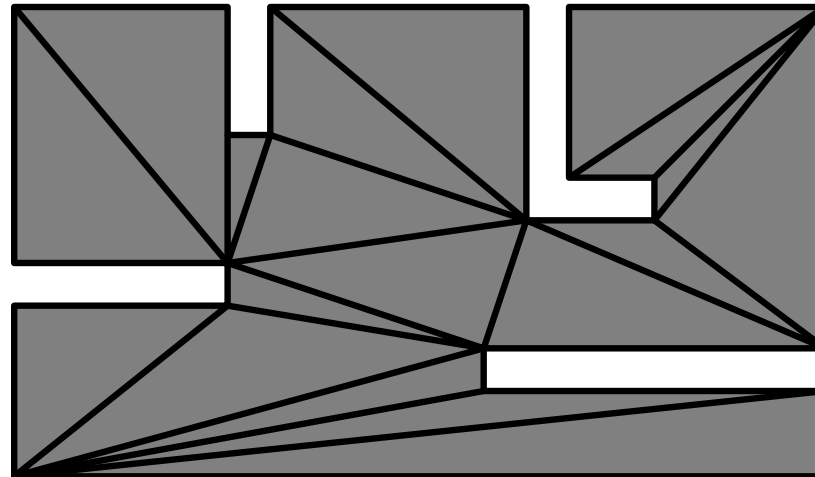


Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen

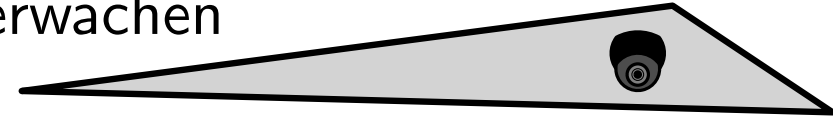


Idee: zerlege P in Dreiecke und überwache die Dreiecke

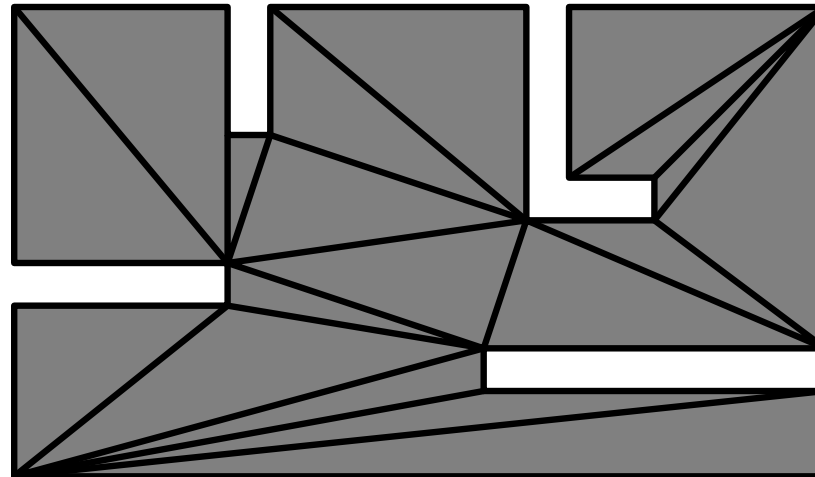


Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen



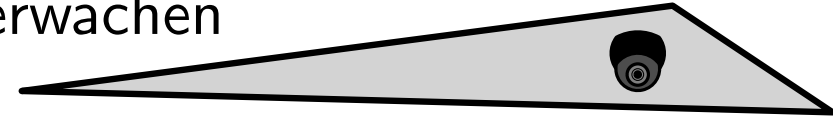
Idee: zerlege P in Dreiecke und überwache die Dreiecke



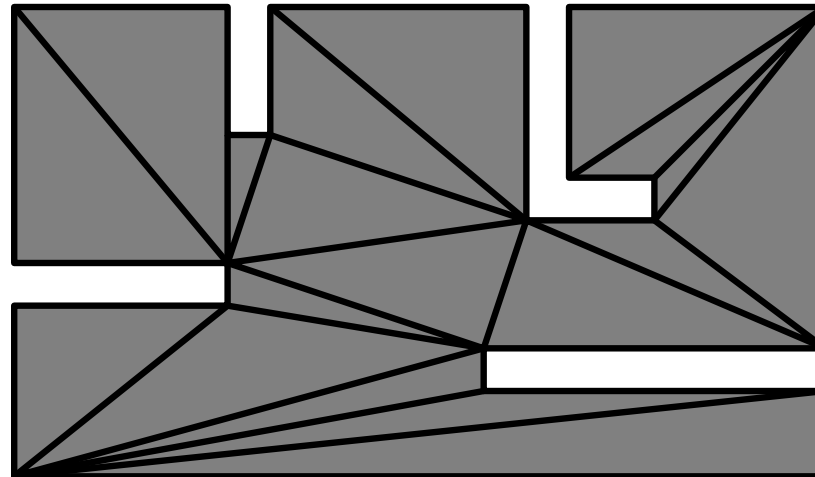
Satz: Jedes einfache Polygon mit n Ecken besitzt eine Triangulierung; jede Triangulierung besteht aus $n - 2$ Dreiecken.

Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen



Idee: zerlege P in Dreiecke und überwache die Dreiecke

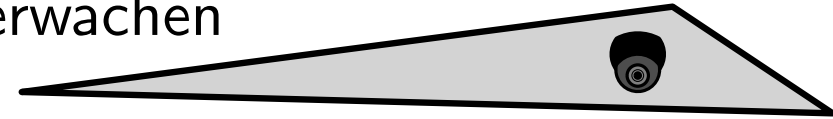


Satz: Jedes einfache Polygon mit n Ecken besitzt eine Triangulierung; jede Triangulierung besteht aus $n - 2$ Dreiecken.

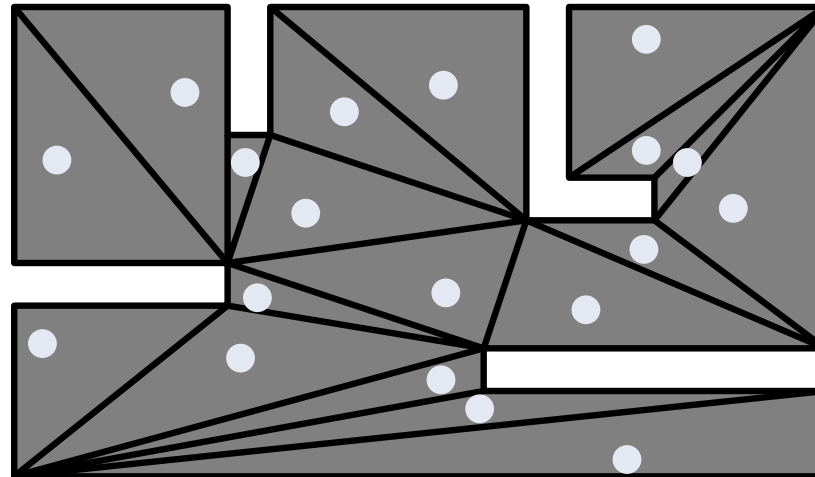
Beweis liefert rekursiven $O(n^2)$ -Algorithmus!

Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen



Idee: zerlege P in Dreiecke und überwache die Dreiecke

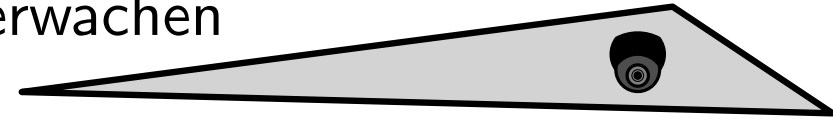


Satz: Jedes einfache Polygon mit n Ecken besitzt eine Triangulierung; jede Triangulierung besteht aus $n - 2$ Dreiecken.

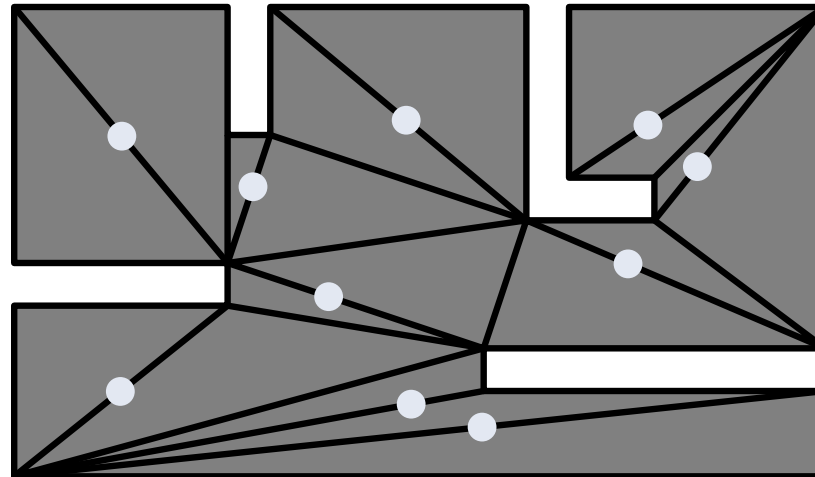
- P lässt sich mit $n - 2$ Kameras in den Dreiecken überwachen

Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen



Idee: zerlege P in Dreiecke und überwache die Dreiecke

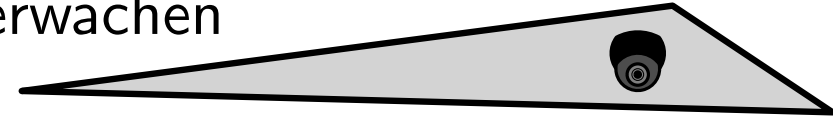


Satz: Jedes einfache Polygon mit n Ecken besitzt eine Triangulierung; jede Triangulierung besteht aus $n - 2$ Dreiecken.

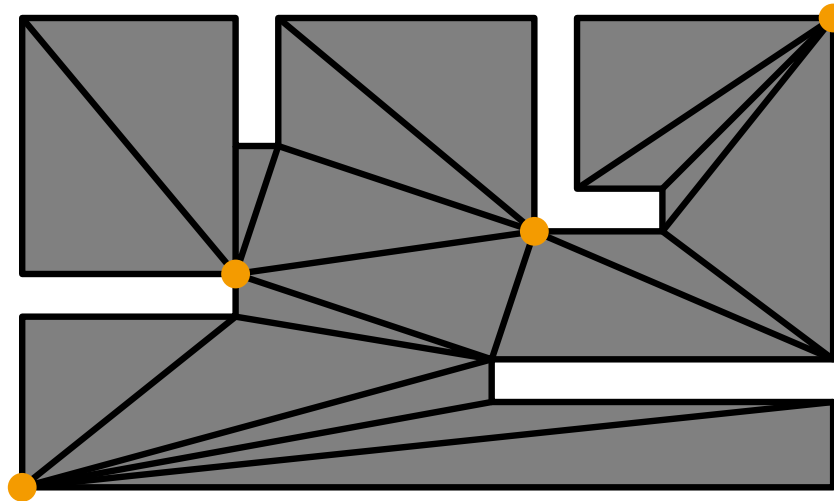
- P lässt sich mit $n - 2$ Kameras in den Dreiecken überwachen
- P lässt sich mit $\approx n/2$ Kameras auf den Diagonalen überwachen

Vereinfachung des Problems

Beobachtung: Dreiecke sind leicht zu überwachen



Idee: zerlege P in Dreiecke und überwache die Dreiecke



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

Das Art-Gallery-Theorem [Chvátal '75]

Satz: 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!

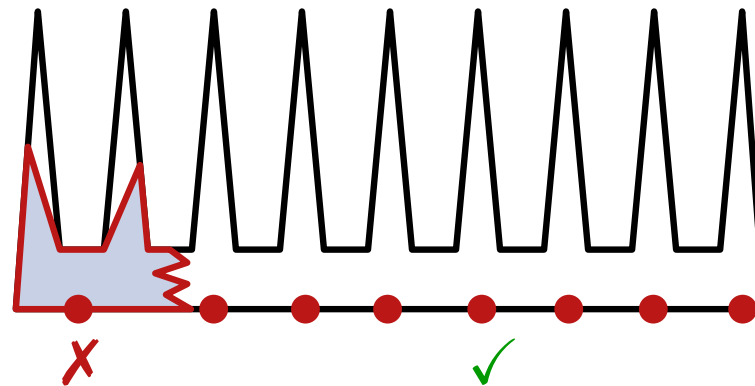
Überlege 2 Minuten mit Nachbarn

Das Art-Gallery-Theorem [Chvátal '75]

Satz: 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!

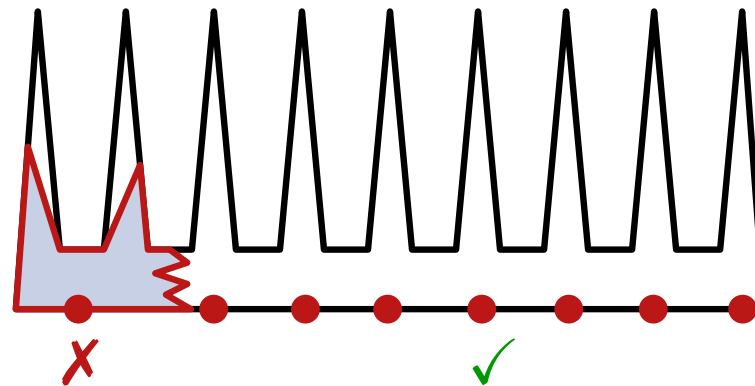


Das Art-Gallery-Theorem [Chvátal '75]

Satz: 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!



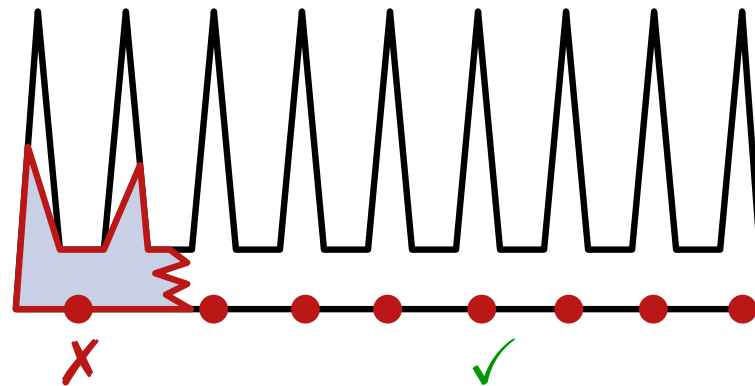
- Teil 2 an der Tafel.

Das Art-Gallery-Theorem [Chvátal '75]

Satz: 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.

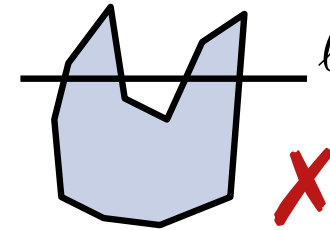
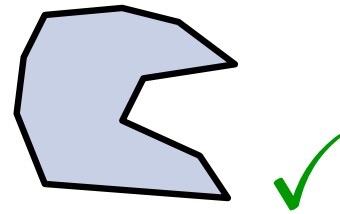
Fazit: Hat man eine Triangulierung, lassen sich $\lfloor n/3 \rfloor$ Kameras in $O(n)$ Zeit platzieren.

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.



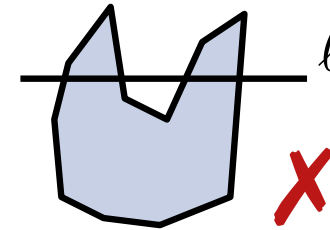
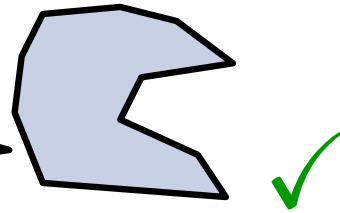
Triangulierung: Überblick

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.

Pfad vom obersten zum untersten Knoten geht nie aufwärts



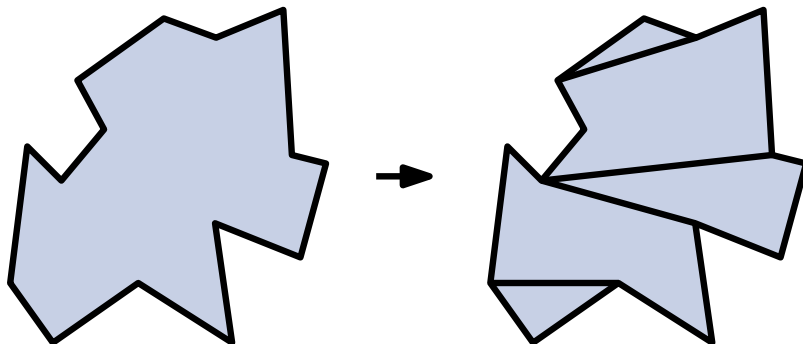
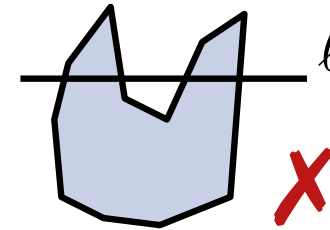
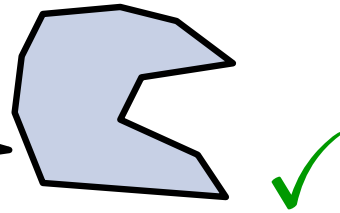
Triangulierung: Überblick

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.

Pfad vom obersten zum untersten Knoten geht nie aufwärts

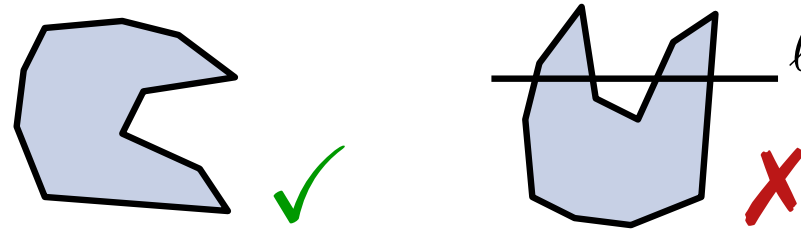


Triangulierung: Überblick

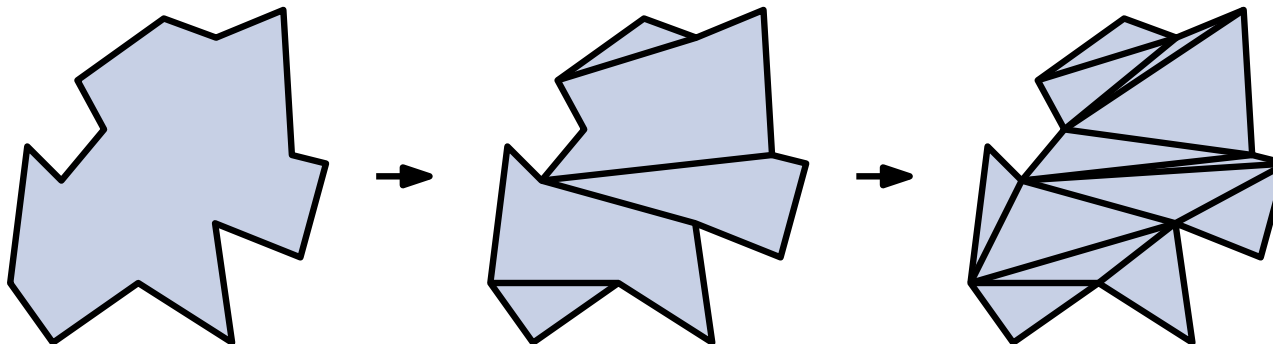
Dreistufiges Verfahren:

- 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

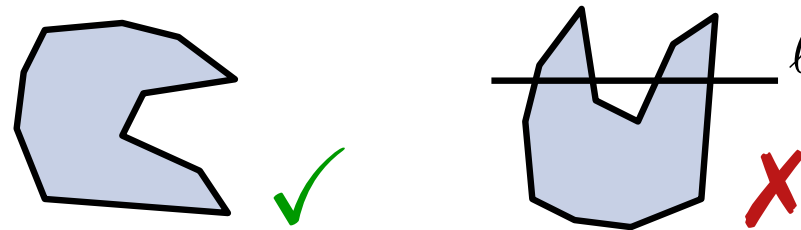


Triangulierung: Überblick

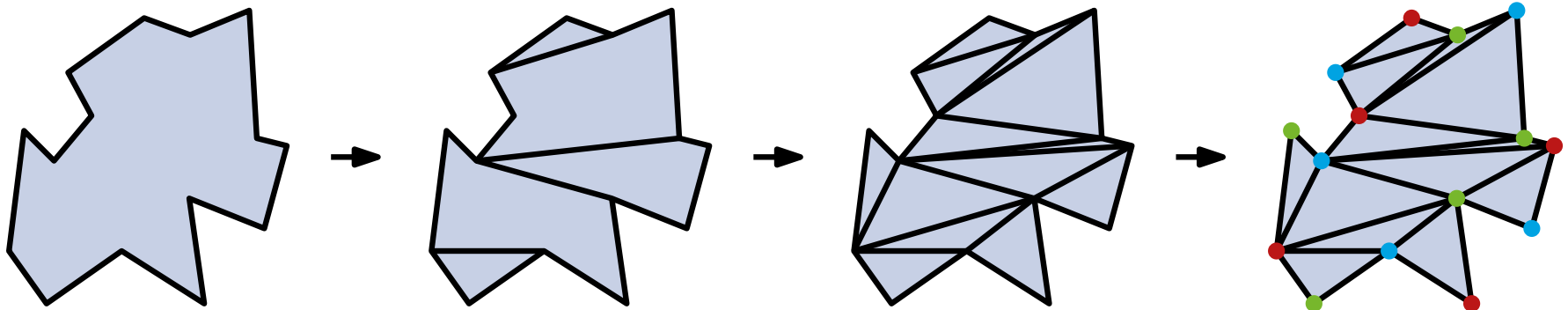
Dreistufiges Verfahren:

- 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.

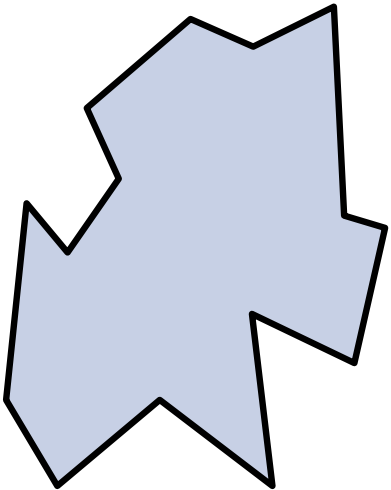


- Schritt 2: Trianguliere y -monotone Teilpolygone
- Schritt 3: benutze DFS um Triangulierung zu färben



Zerlegen in y -monotone Teile

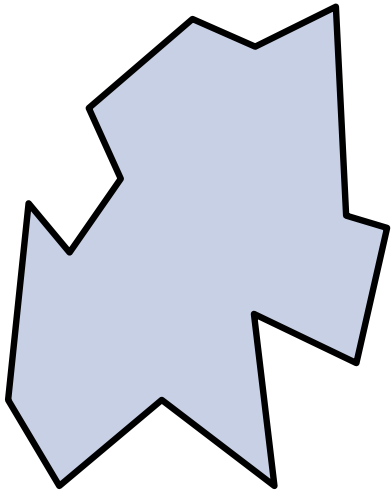
Idee: Unterscheide fünf verschiedene Knotenarten



Zerlegen in y -monotone Teile

Idee: Unterscheide fünf verschiedene Knotenarten

– *Wendeknoten:*

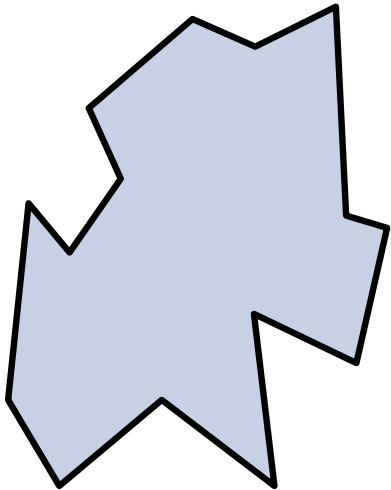


– *reguläre Knoten*

Zerlegen in y -monotone Teile

Idee: Unterscheide fünf verschiedene Knotenarten

- *Wendeknoten*:
vertikale Laufrichtung wechselt



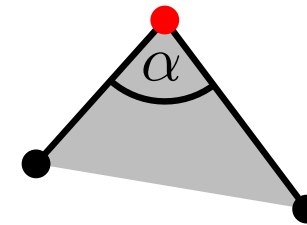
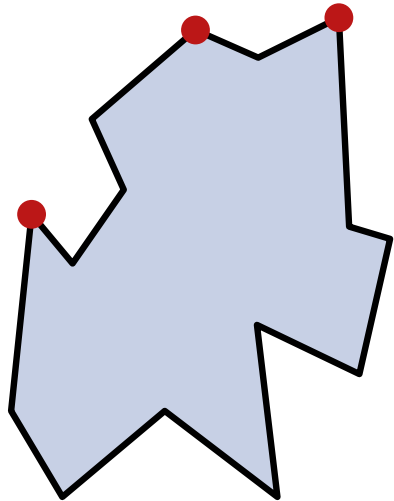
- *reguläre Knoten*

Zerlegen in y -monotone Teile

Idee: Unterscheide fünf verschiedene Knotenarten

– *Wendeknoten:*
vertikale Laufrichtung wechselt

■ *Startknoten*



falls $\alpha < 180^\circ$

– *reguläre Knoten*

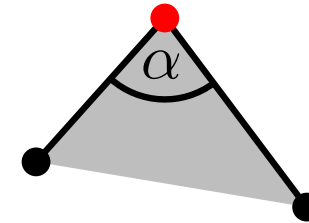
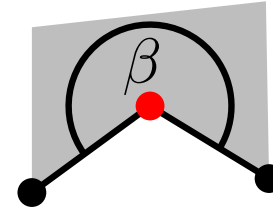
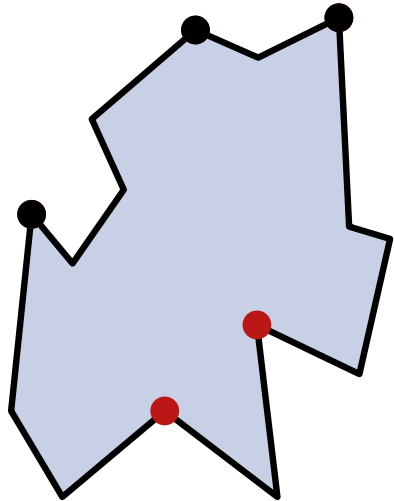
Zerlegen in y -monotone Teile

Idee: Unterscheide fünf verschiedene Knotenarten

– *Wendeknoten:*
vertikale Laufrichtung wechselt

■ *Startknoten*

■ *Splitknoten*



falls $\alpha < 180^\circ$

falls $\beta > 180^\circ$

– *reguläre Knoten*

Zerlegen in y -monotone Teile

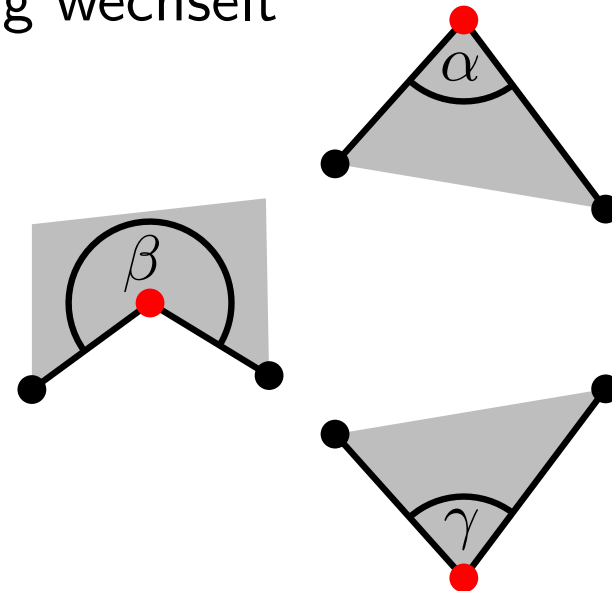
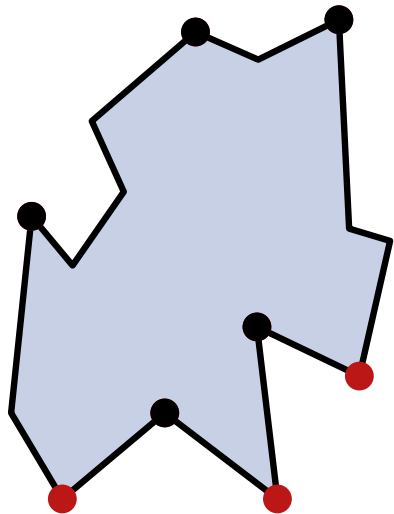
Idee: Unterscheide fünf verschiedene Knotenarten

– *Wendeknoten:*
vertikale Laufrichtung wechselt

■ *Startknoten*

■ *Splitknoten*

■ *Endknoten*



– *reguläre Knoten*

Zerlegen in y -monotone Teile

Idee: Unterscheide fünf verschiedene Knotenarten

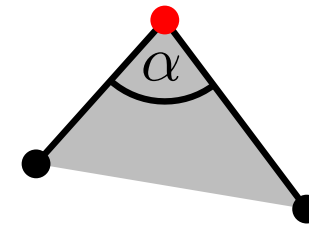
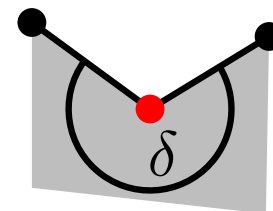
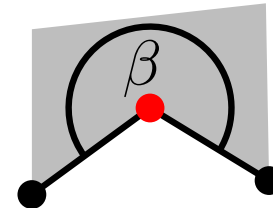
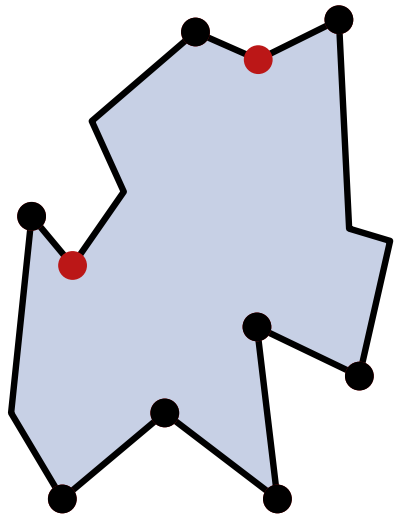
– *Wendeknoten:*
vertikale Laufrichtung wechselt

■ *Startknoten*

■ *Splitknoten*

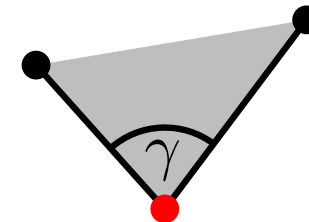
■ *Endknoten*

■ *Mergeknoten*



falls $\alpha < 180^\circ$

falls $\beta > 180^\circ$



falls $\gamma < 180^\circ$

falls $\delta > 180^\circ$

– *reguläre Knoten*

Zerlegen in y -monotone Teile

Idee: Unterscheide fünf verschiedene Knotenarten

– *Wendeknoten:*
vertikale Laufrichtung wechselt

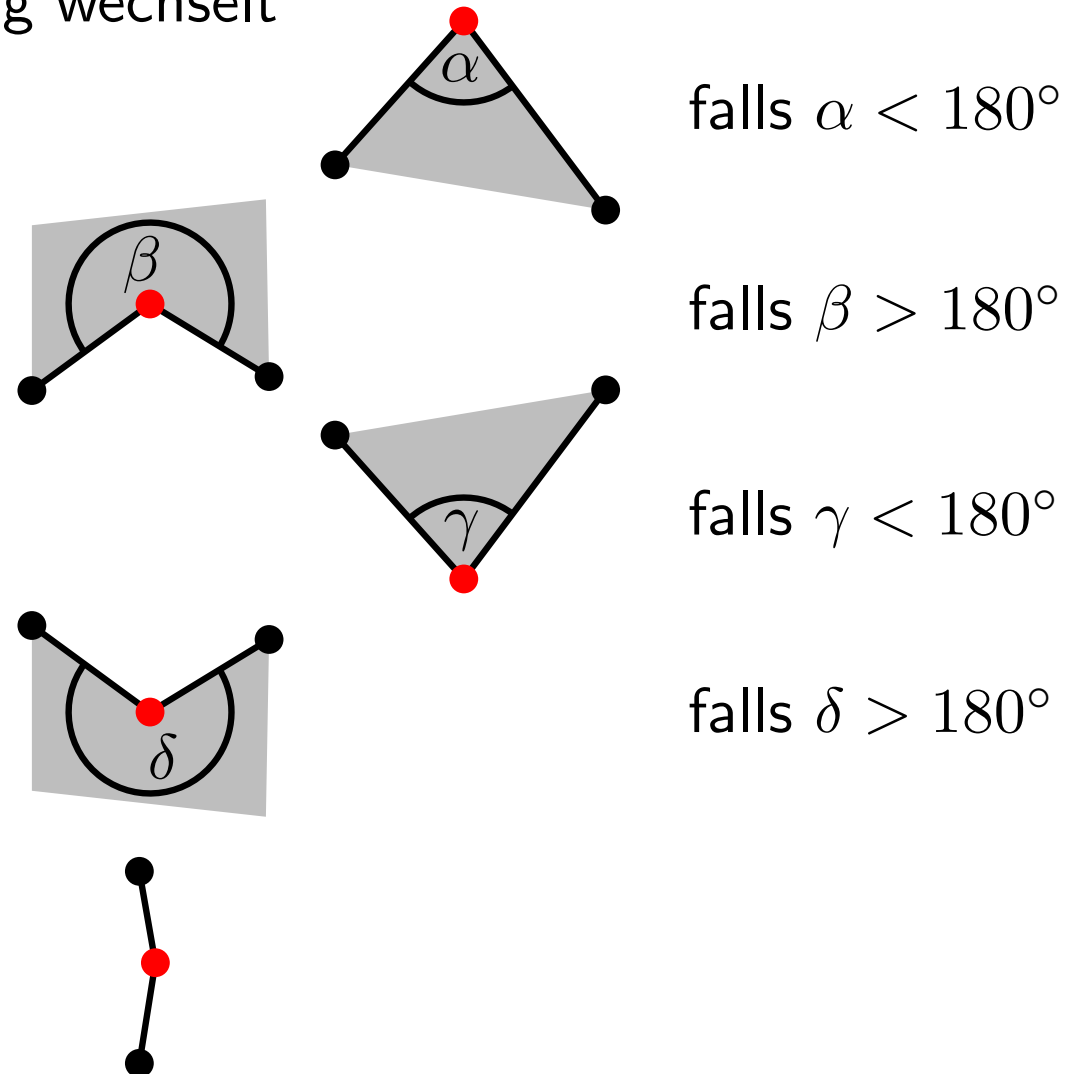
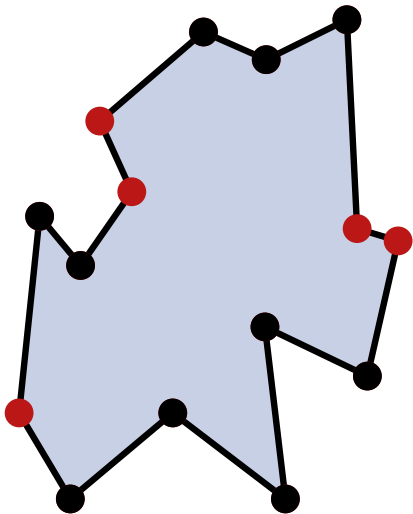
■ *Startknoten*

■ *Splitknoten*

■ *Endknoten*

■ *Mergeknoten*

– *reguläre Knoten*



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

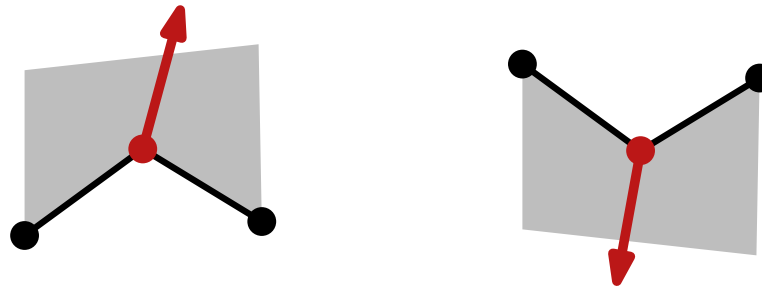
Lemma: Ein Polygon ist y -monoton, wenn es keine Split- oder Mergeknoten besitzt.

Beweis: an der Tafel

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

Beweis: an der Tafel

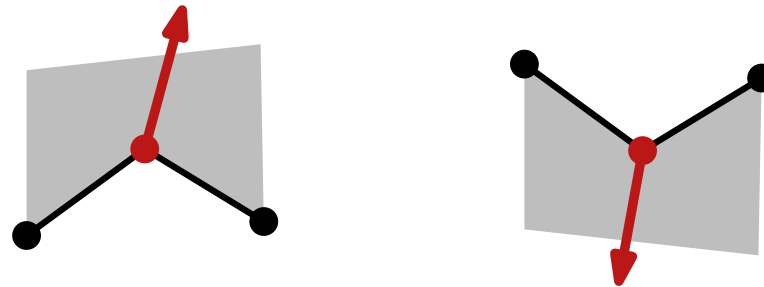
⇒ Wir müssen alle Split- und Mergeknoten durch Einfügen von Diagonalen entfernen



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

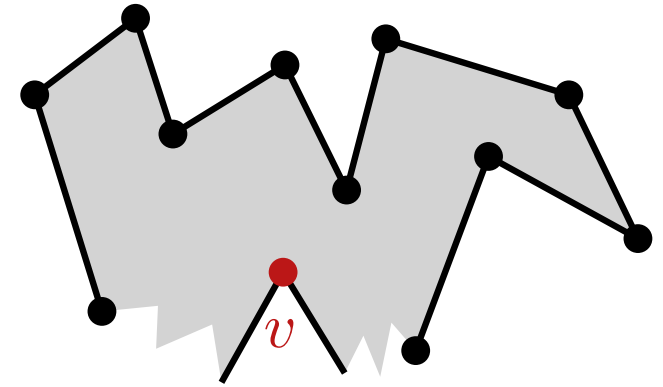
Beweis: an der Tafel

⇒ Wir müssen alle Split- und Mergeknoten durch Einfügen von Diagonalen entfernen



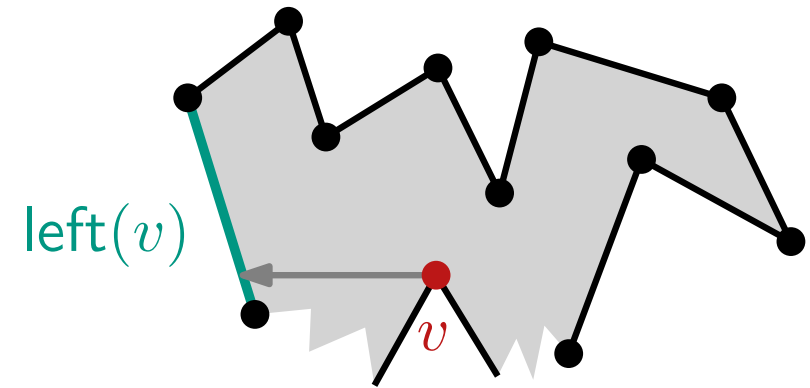
Vorsicht: Diagonalen dürfen weder Kanten von P noch andere Diagonalen schneiden

1) Diagonalen für Splitknoten



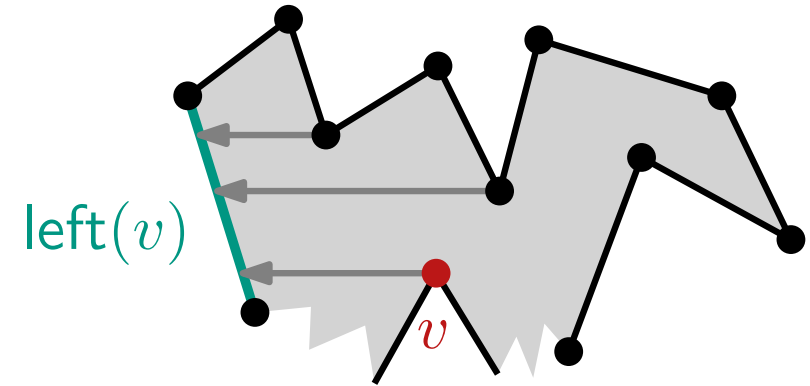
1) Diagonalen für Splitknoten

- betrachte für jeden Knoten v linke Nachbarkante $\text{left}(v)$ bzgl. horizontaler sweep line ℓ



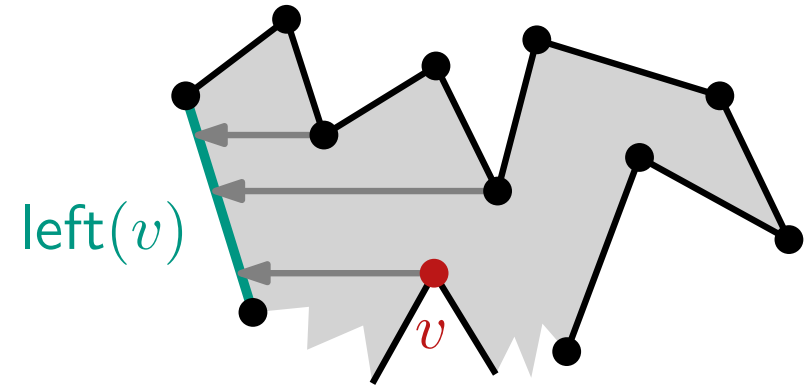
1) Diagonalen für Splitknoten

- betrachte für jeden Knoten v linke Nachbarkante $\text{left}(v)$ bzgl. horizontaler sweep line ℓ



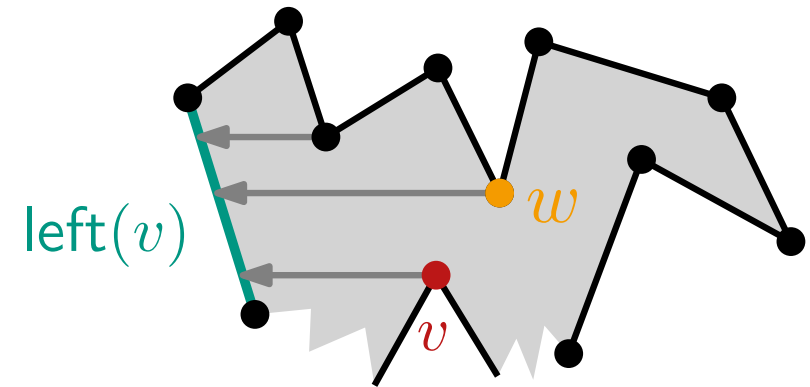
1) Diagonalen für Splitknoten

- betrachte für jeden Knoten v linke Nachbarkante $\text{left}(v)$ bzgl. horizontaler sweep line ℓ
- verbinde Splitknoten v zu niedrigstem Knoten w oberhalb v mit $\text{left}(w) = \text{left}(v)$



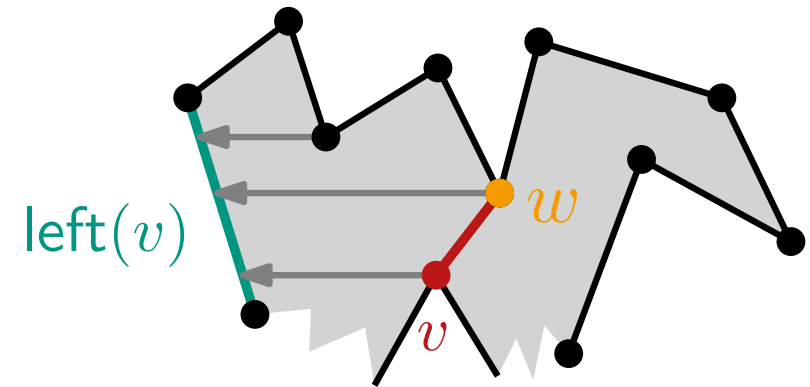
1) Diagonalen für Splitknoten

- betrachte für jeden Knoten v linke Nachbarkante $\text{left}(v)$ bzgl. horizontaler sweep line ℓ
- verbinde Splitknoten v zu niedrigstem Knoten w oberhalb v mit $\text{left}(w) = \text{left}(v)$



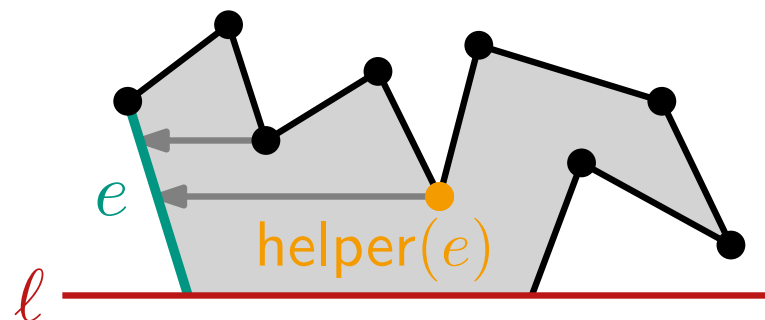
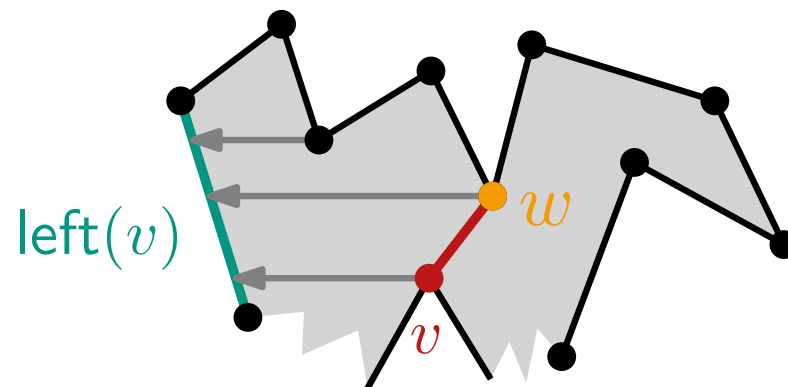
1) Diagonalen für Splitknoten

- betrachte für jeden Knoten v linke Nachbarkante $\text{left}(v)$ bzgl. horizontaler sweep line ℓ
- verbinde Splitknoten v zu niedrigstem Knoten w oberhalb v mit $\text{left}(w) = \text{left}(v)$



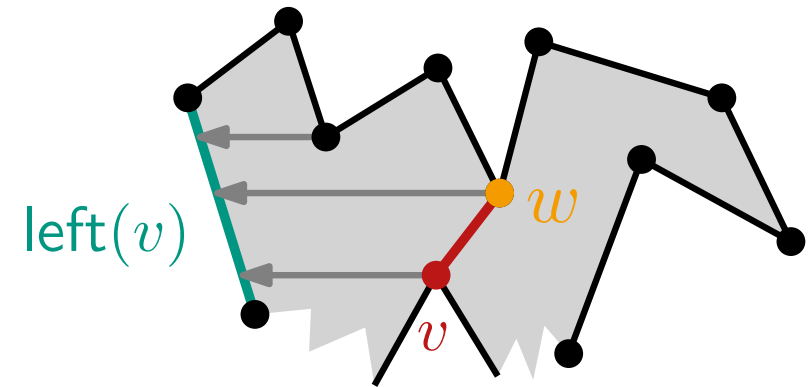
1) Diagonalen für Splitknoten

- betrachte für jeden Knoten v linke Nachbarkante $\text{left}(v)$ bzgl. horizontaler sweep line ℓ
- verbinde Splitknoten v zu niedrigstem Knoten w oberhalb v mit $\text{left}(w) = \text{left}(v)$
- speichere für jede Kante e den untersten Knoten w mit $\text{left}(w) = e$ als $\text{helper}(e)$



1) Diagonalen für Splitknoten

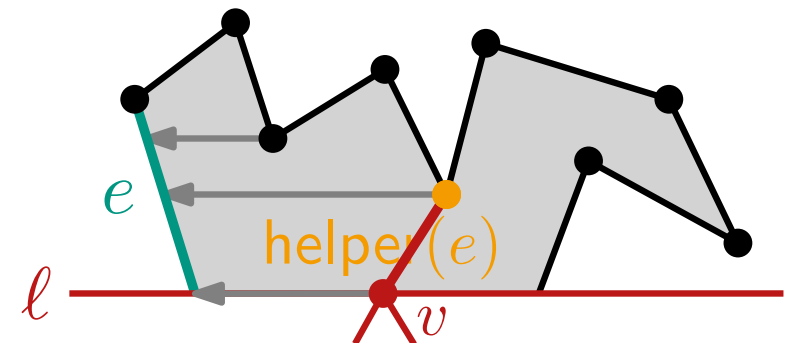
- betrachte für jeden Knoten v linke Nachbarkante $\text{left}(v)$ bzgl. horizontaler sweep line ℓ



- verbinde Splitknoten v zu niedrigstem Knoten w oberhalb v mit $\text{left}(w) = \text{left}(v)$

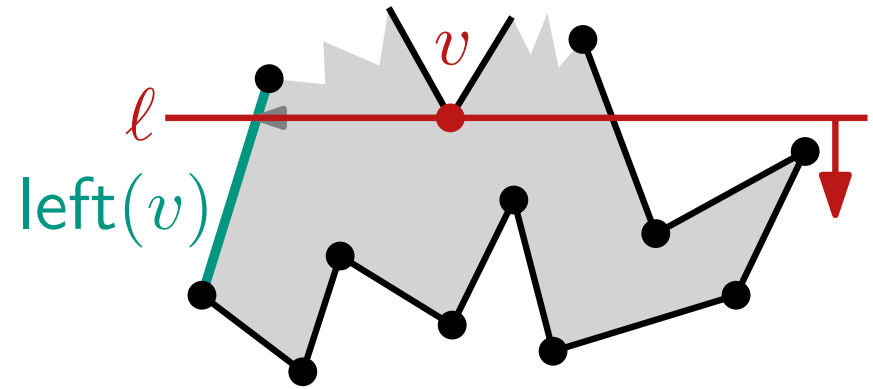
- speichere für jede Kante e den untersten Knoten w mit $\text{left}(w) = e$ als $\text{helper}(e)$

- trifft ℓ auf Splitknoten v :
verbinde v mit $\text{helper}(\text{left}(v))$



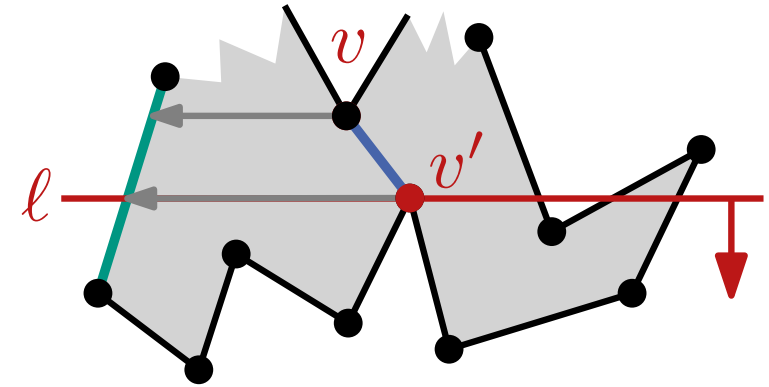
2) Diagonalen für Mergeknoten

- erreicht man Mergeknoten v
wird $\text{helper}(\text{left}(v)) = v$



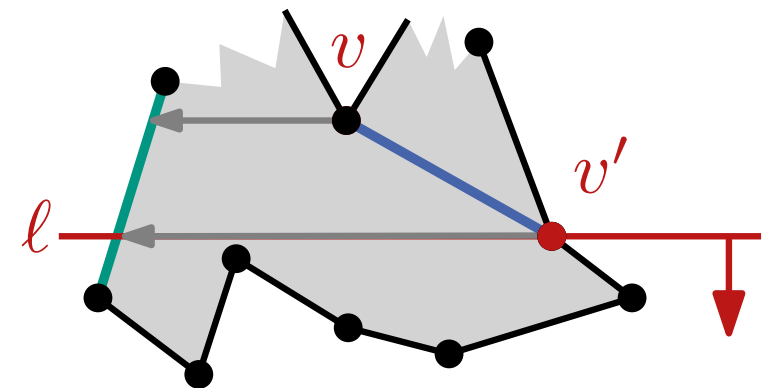
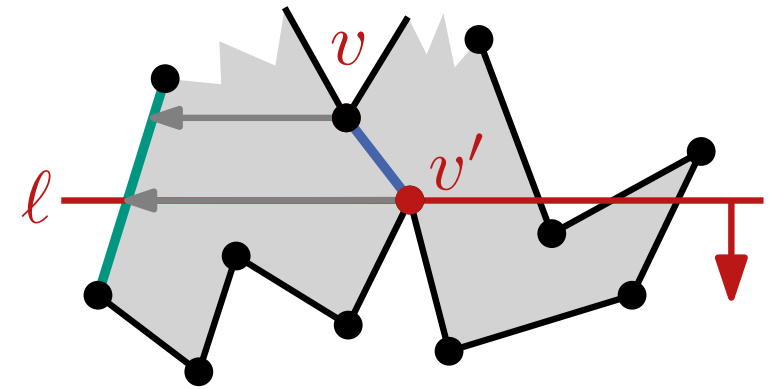
2) Diagonalen für Mergeknoten

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



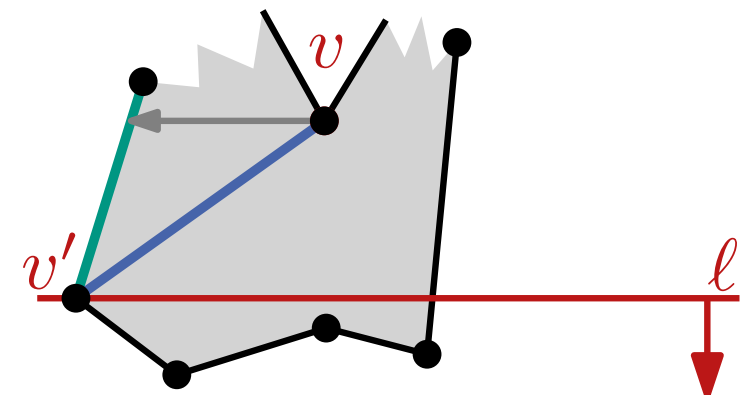
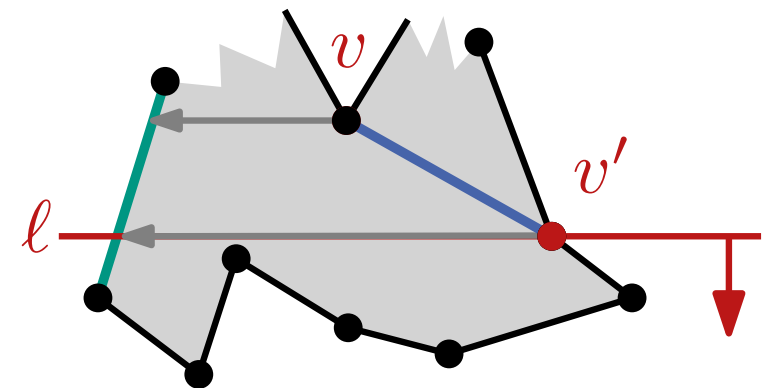
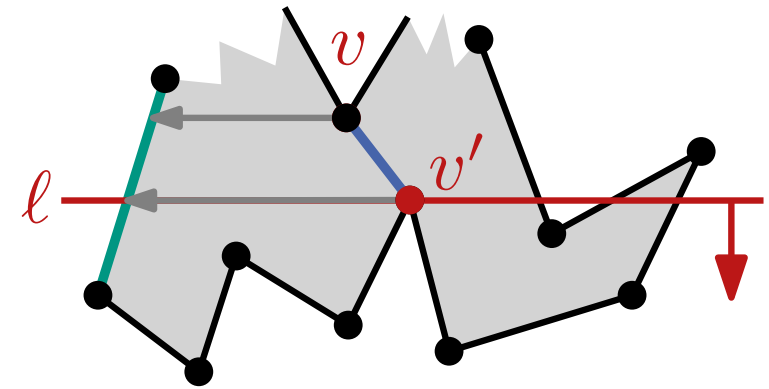
2) Diagonalen für Mergeknoten

- erreicht man Mergeknoten v
wird $\text{helper}(\text{left}(v)) = v$
- erreicht man Splitknoten v'
mit $\text{left}(v') = \text{left}(v)$ wird
Diagonale (v, v') eingefügt
- ersetzt man $\text{helper}(\text{left}(v))$ durch v'
wird Diagonale (v, v') eingefügt



2) Diagonalen für Mergeknoten

- erreicht man Mergeknoten v
wird $\text{helper}(\text{left}(v)) = v$
- erreicht man Splitknoten v'
mit $\text{left}(v') = \text{left}(v)$ wird
Diagonale (v, v') eingefügt
- ersetzt man $\text{helper}(\text{left}(v))$ durch v'
wird Diagonale (v, v') eingefügt
- erreicht man das Ende v' von $\text{left}(v)$
wird Diagonale (v, v') eingefügt



Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$

$Q \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)

while $Q \neq \emptyset$ **do**

```
|  $v \leftarrow Q.\text{nextVertex}()$   
|  $Q.\text{deleteVertex}(v)$   
|  $\text{handleVertex}(v)$ 
```

return \mathcal{D}

Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$

$Q \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)

while $Q \neq \emptyset$ **do**

$v \leftarrow Q.\text{nextVertex}()$
 $Q.\text{deleteVertex}(v)$
 $\text{handleVertex}(v)$

return \mathcal{D}

Datenstruktur
für eingebettete
Graphen mit
Zugriff auf
Kanten, Knoten,
zugeh. Facetten
usw. (s. Kap.
2.2 im Buch)

Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$

$Q \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)

while $Q \neq \emptyset$ **do**

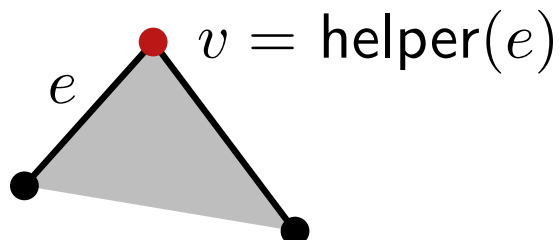
$v \leftarrow Q.\text{nextVertex}()$
 $Q.\text{deleteVertex}(v)$
 $\text{handleVertex}(v)$

return \mathcal{D}

handleStartVertex(vertex v)

$\mathcal{T} \leftarrow$ füge linke Kante e ein

helper(e) $\leftarrow v$



Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$

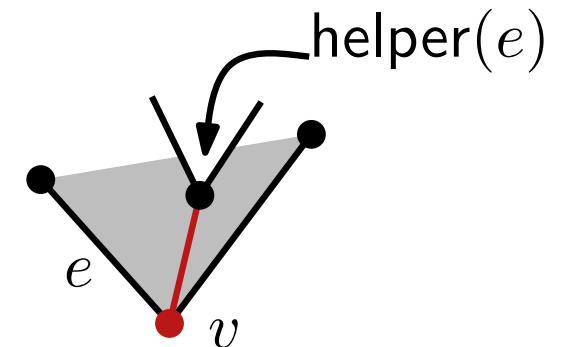
$\mathcal{Q} \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)

while $\mathcal{Q} \neq \emptyset$ **do**

$v \leftarrow \mathcal{Q}.\text{nextVertex}()$
 $\mathcal{Q}.\text{deleteVertex}(v)$
 handleVertex(v)

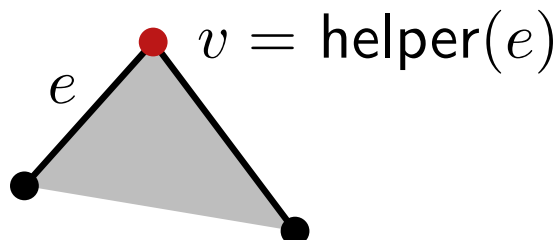
return \mathcal{D}



handleStartVertex(vertex v)

$\mathcal{T} \leftarrow$ füge linke Kante e ein

$\text{helper}(e) \leftarrow v$



handleEndVertex(vertex v)

$e \leftarrow$ linke Kante

if isMergeVertex($\text{helper}(e)$) **then**

$\mathcal{D} \leftarrow$ füge $(\text{helper}(e), v)$ 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**

$v \leftarrow \mathcal{Q}.\text{nextVertex}()$
 $\mathcal{Q}.\text{deleteVertex}(v)$
 handleVertex(v)

return \mathcal{D}

handleSplitVertex(vertex v)

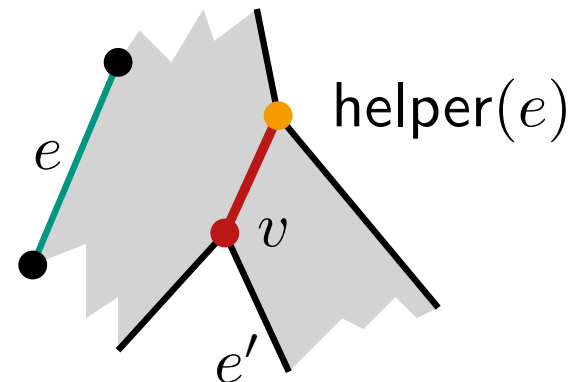
$e \leftarrow$ Kante links von v in \mathcal{T}

$\mathcal{D} \leftarrow$ füge $(\text{helper}(e), v)$ ein

$\text{helper}(e) \leftarrow v$

$\mathcal{T} \leftarrow$ füge rechte Kante e' von v ein

$\text{helper}(e') \leftarrow v$



Algorithmus MakeMonotone(P)

MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ doppelt-verkettete Kantenliste für $(V(P), E(P))$

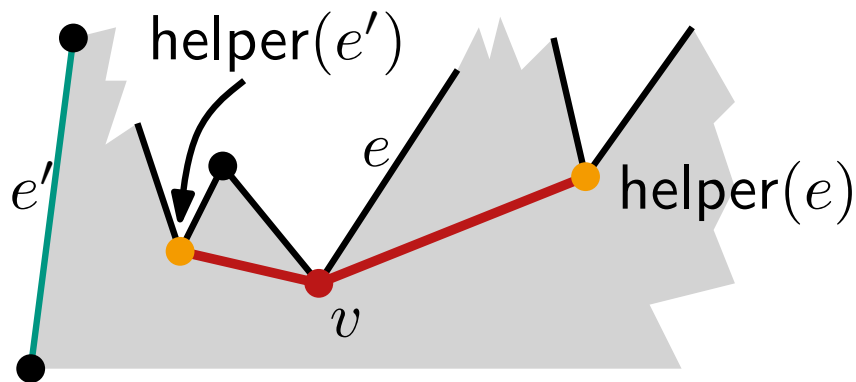
$Q \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)

while $Q \neq \emptyset$ **do**

$v \leftarrow Q.\text{nextVertex}()$
 $Q.\text{deleteVertex}(v)$
 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 $(V(P), E(P))$

$Q \leftarrow$ priority queue für $V(P)$ lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$ (binärer Suchbaum für Sweep-Line Status)

while $Q \neq \emptyset$ **do**

$v \leftarrow Q.\text{nextVertex}()$

$Q.\text{deleteVertex}(v)$

 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') $\leftarrow v$

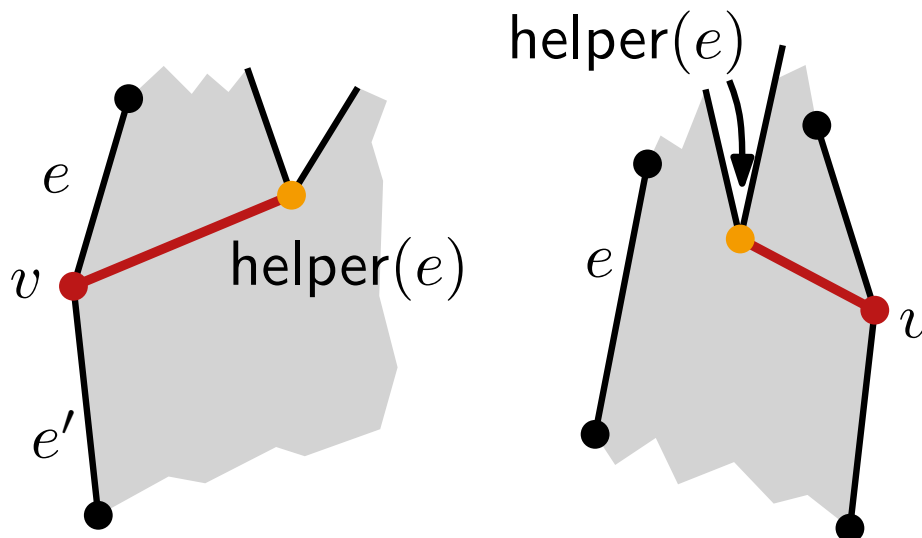
else

$e \leftarrow$ Kante links von v in \mathcal{T}

if isMergeVertex(helper(e)) **then**

$\mathcal{D} \leftarrow$ füge (helper(e), v) ein

 helper(e) $\leftarrow v$



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

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

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

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

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

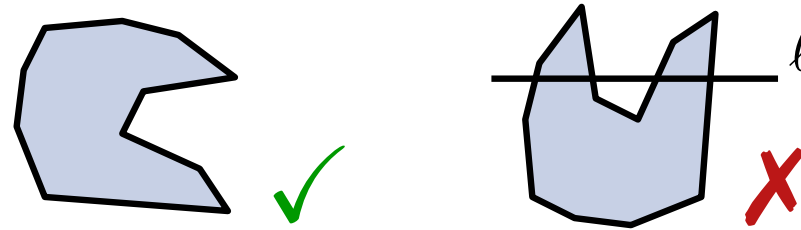
- priority queue Q erzeugen: $O(n)$ Zeit
- Sweep-Line Status \mathcal{T} initialisieren: $O(1)$ Zeit
- Eventbehandlung pro Event: $O(\log n)$ Zeit
 - $Q.deleteMax$: $O(\log n)$ Zeit
 - Element aus \mathcal{T} suchen, löschen, einfügen: $O(\log n)$ Zeit
 - ≤ 2 Diagonalen in \mathcal{D} einfügen: $O(1)$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

Triangulierung: Überblick

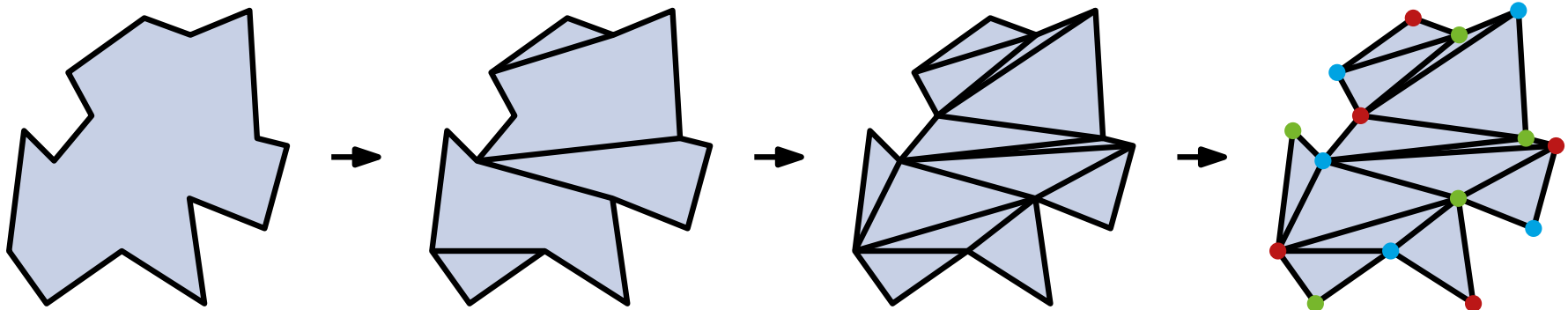
Dreistufiges Verfahren:

- 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.



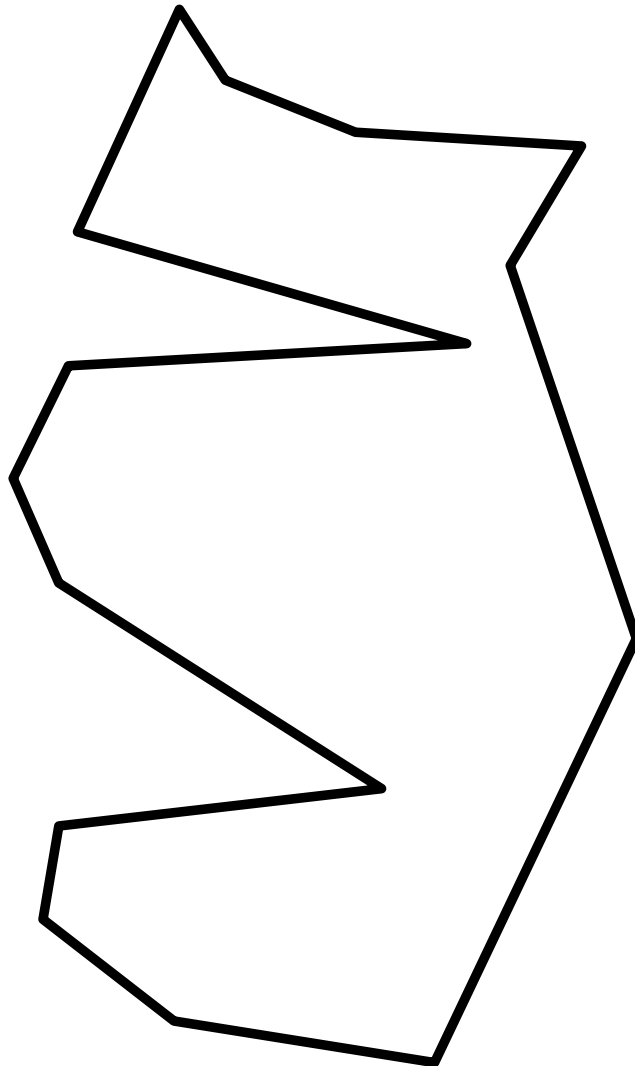
- Schritt 2: Trianguliere y -monotone Teilpolygone **ToDo!**
- Schritt 3: benutze DFS um Triangulierung zu färben ✓



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

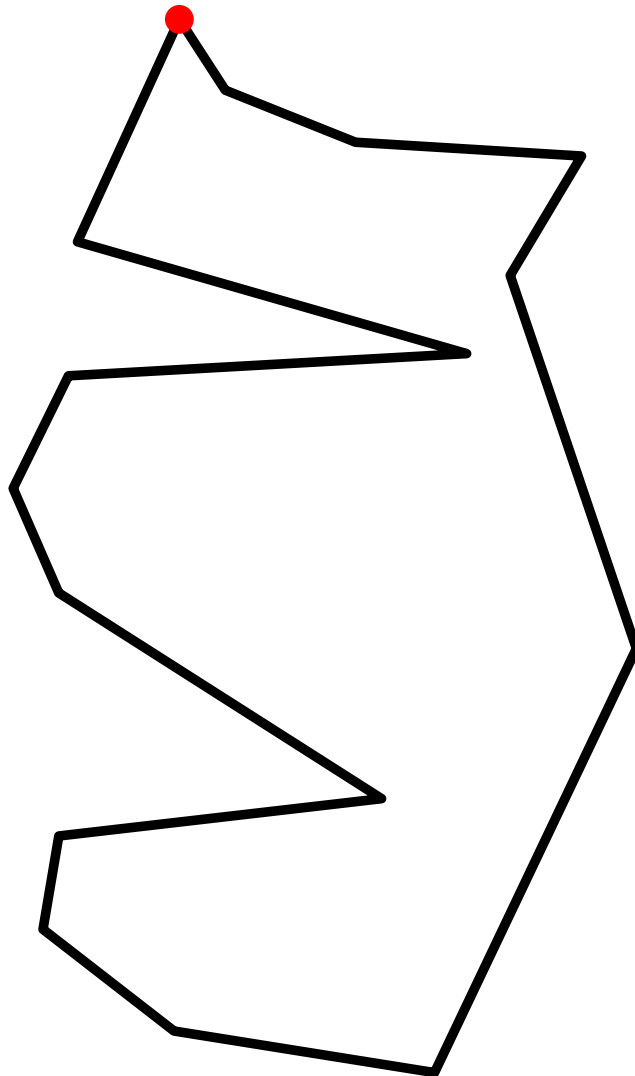
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

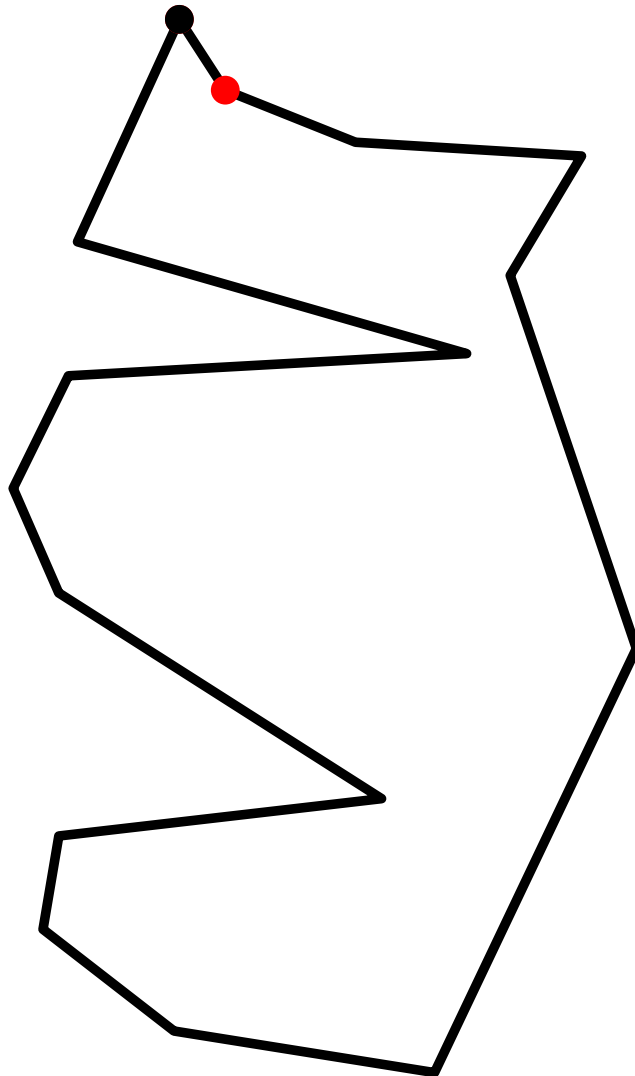
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

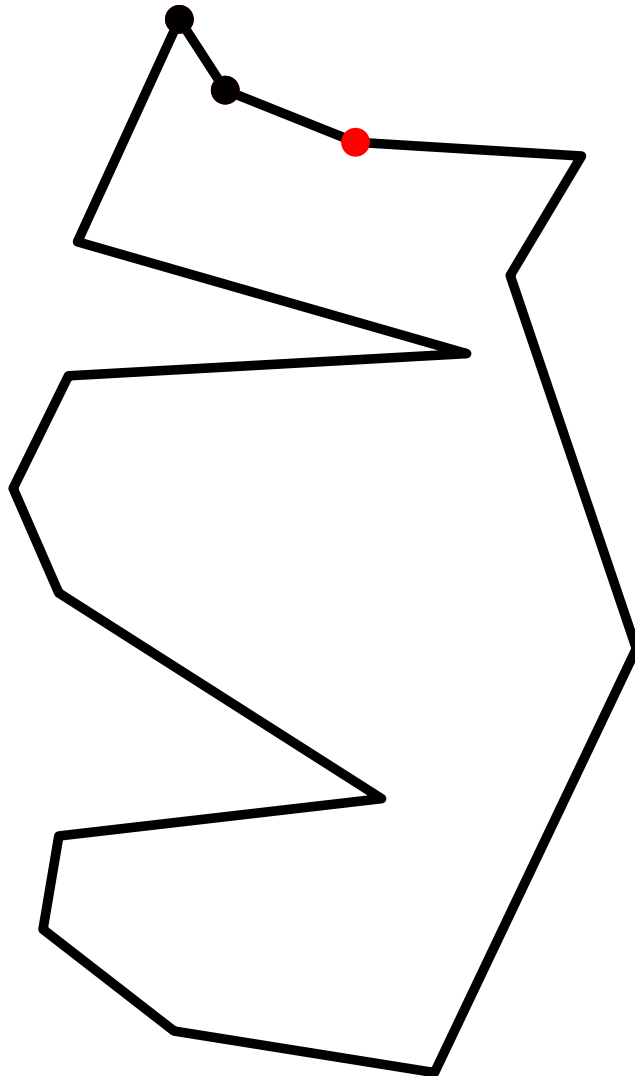
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

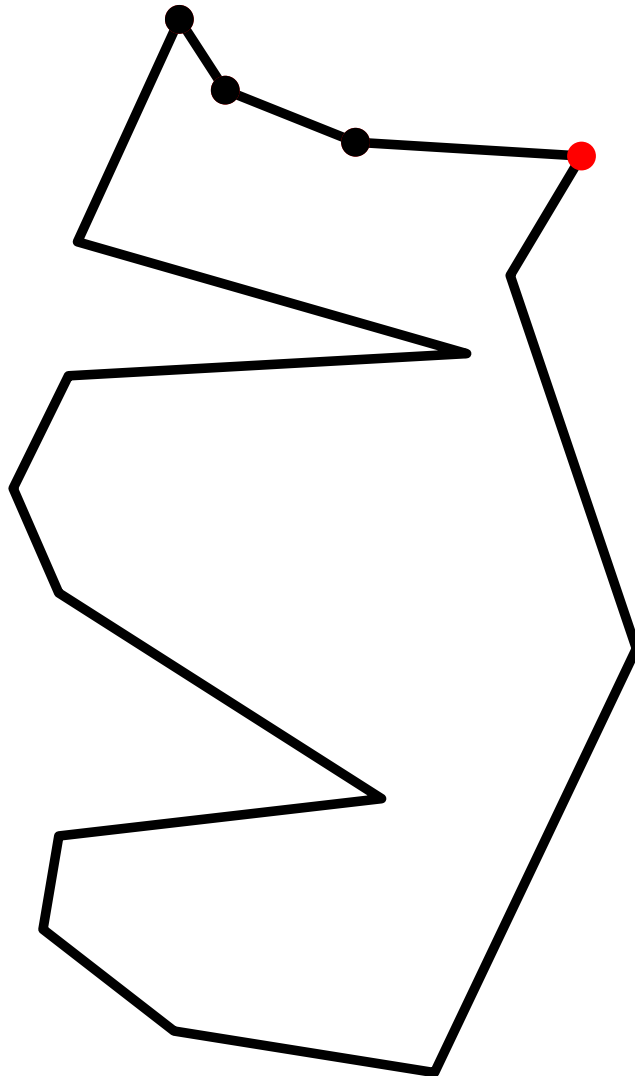
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

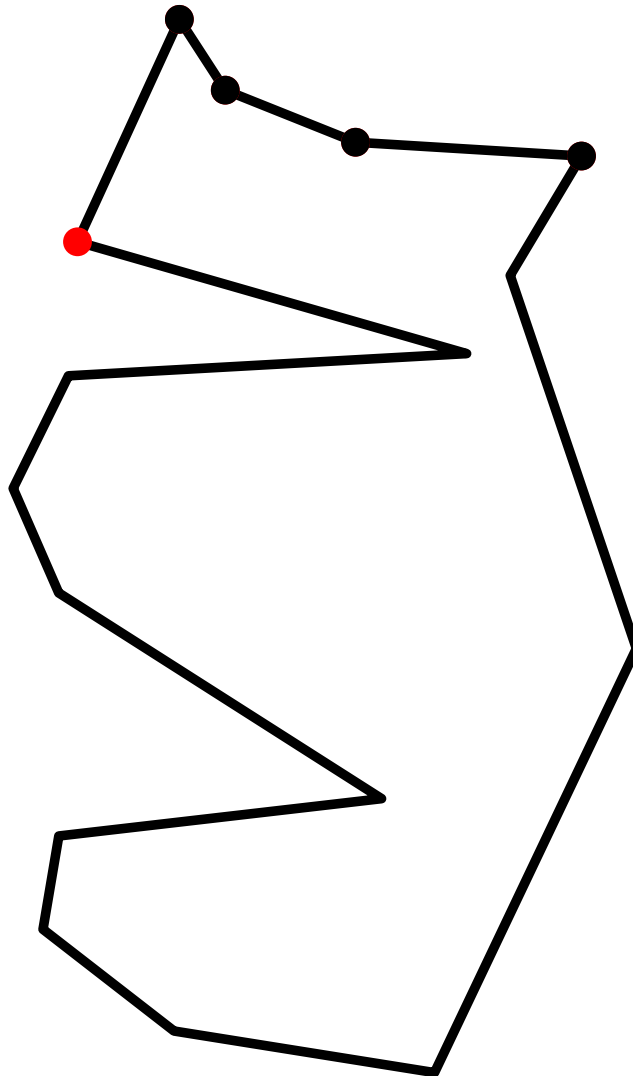
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

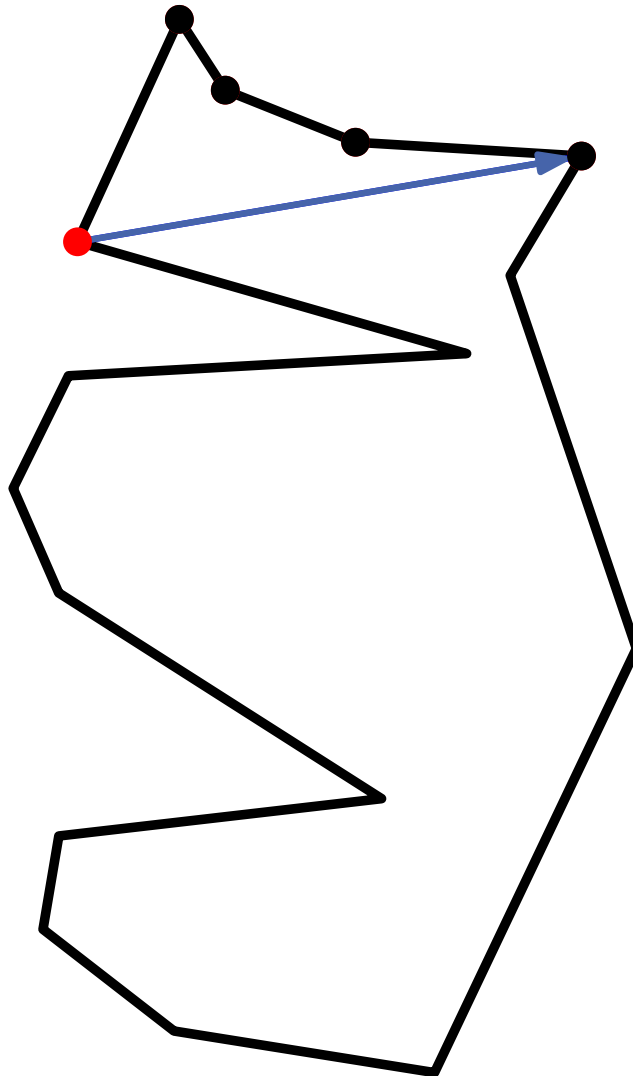
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

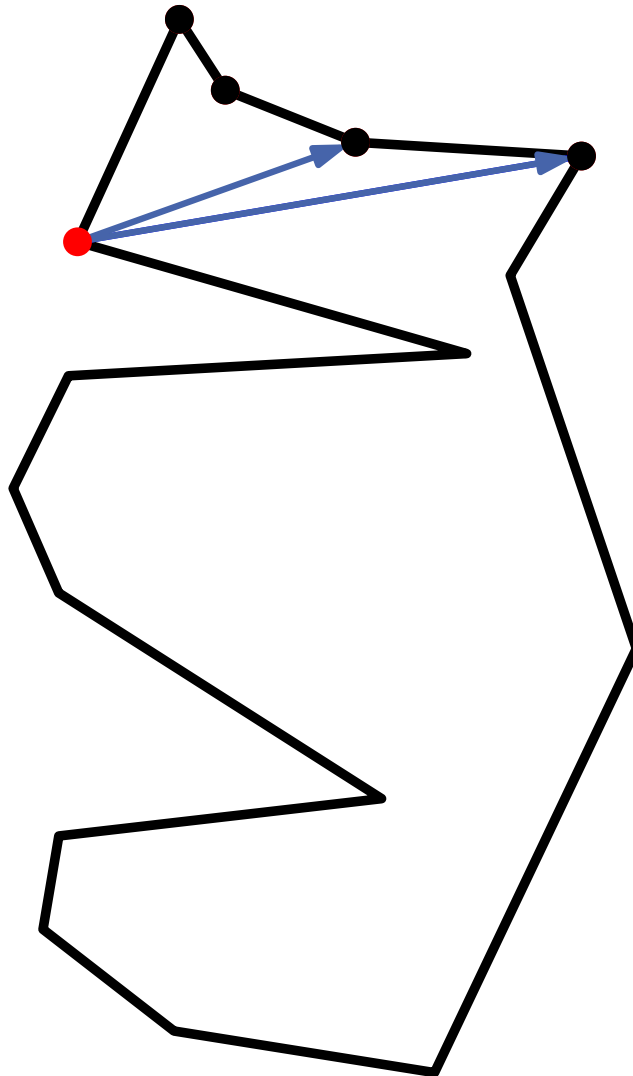
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

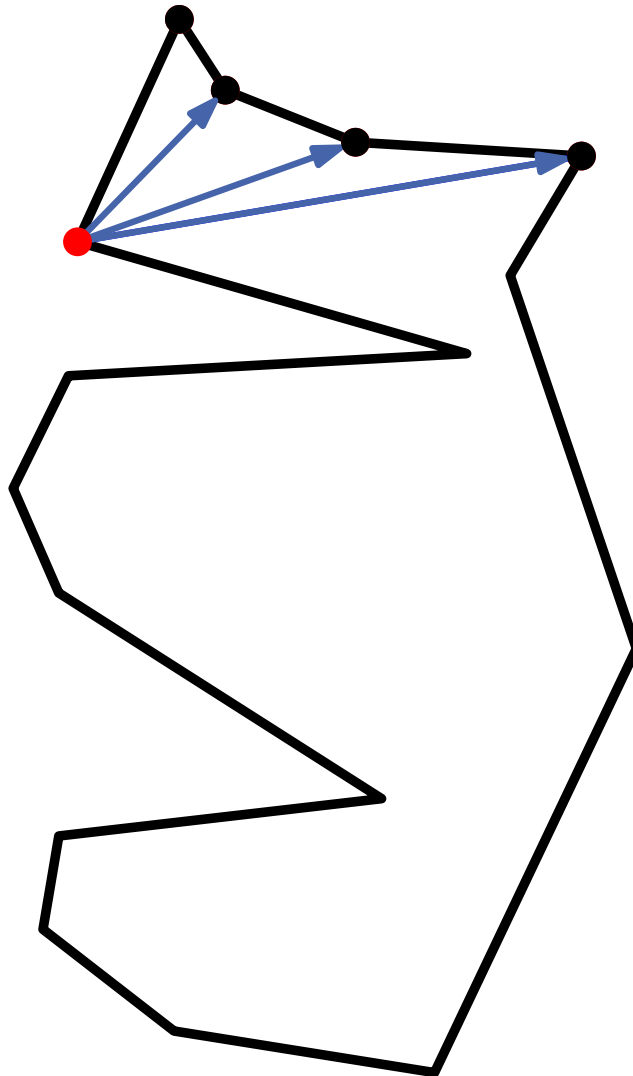
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

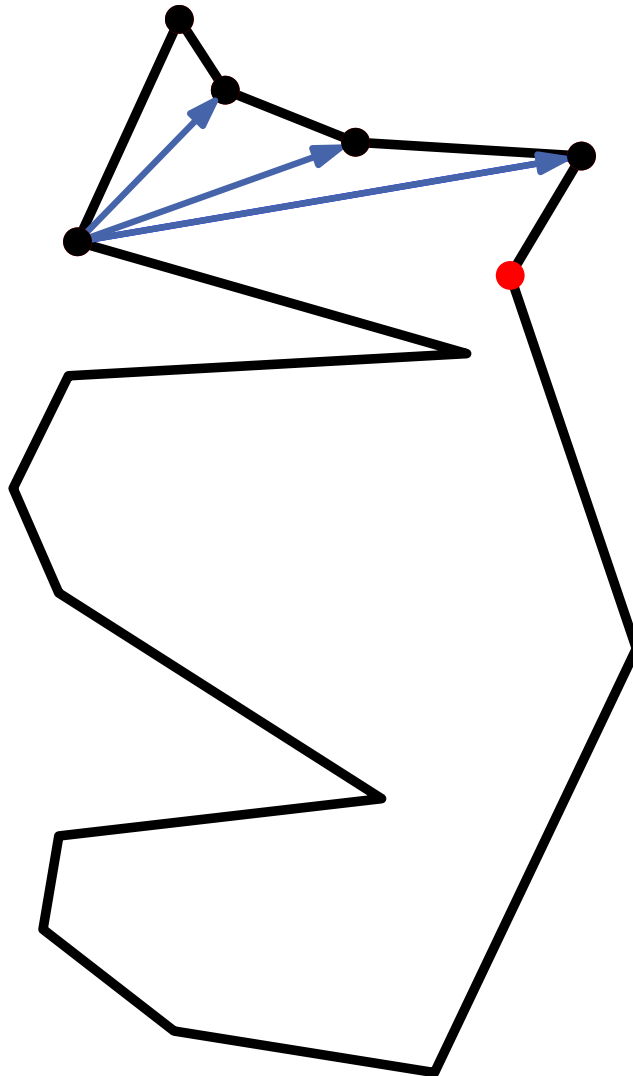
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

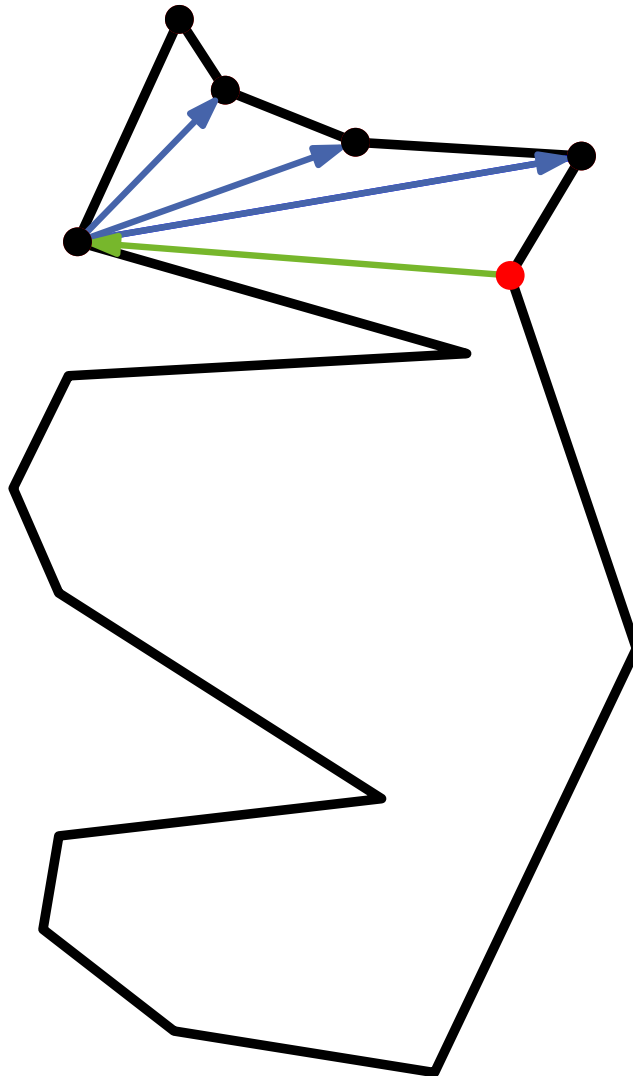
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

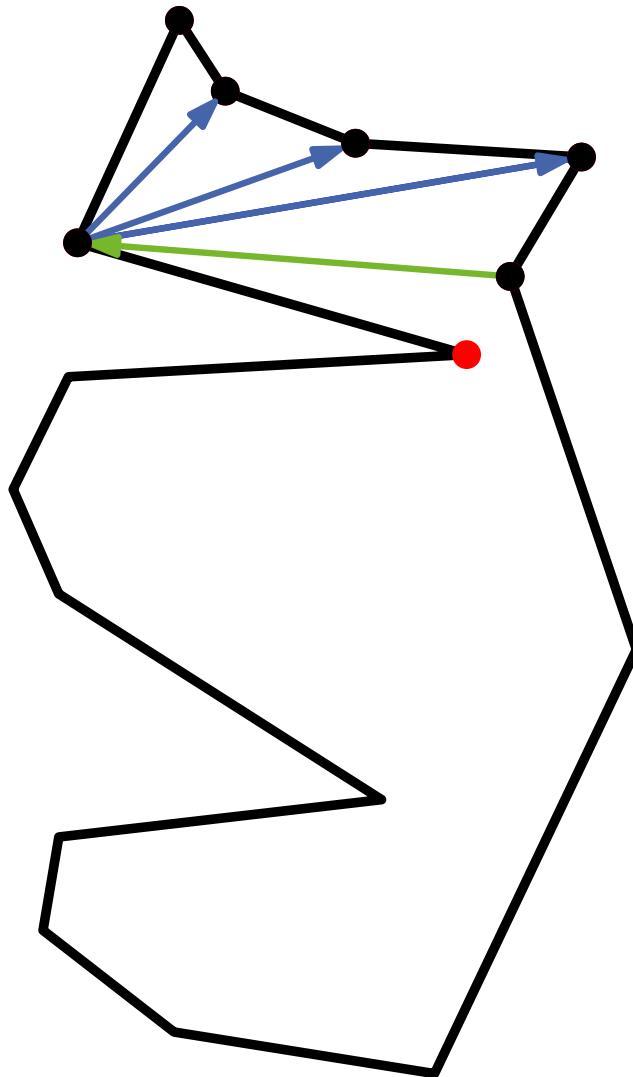
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

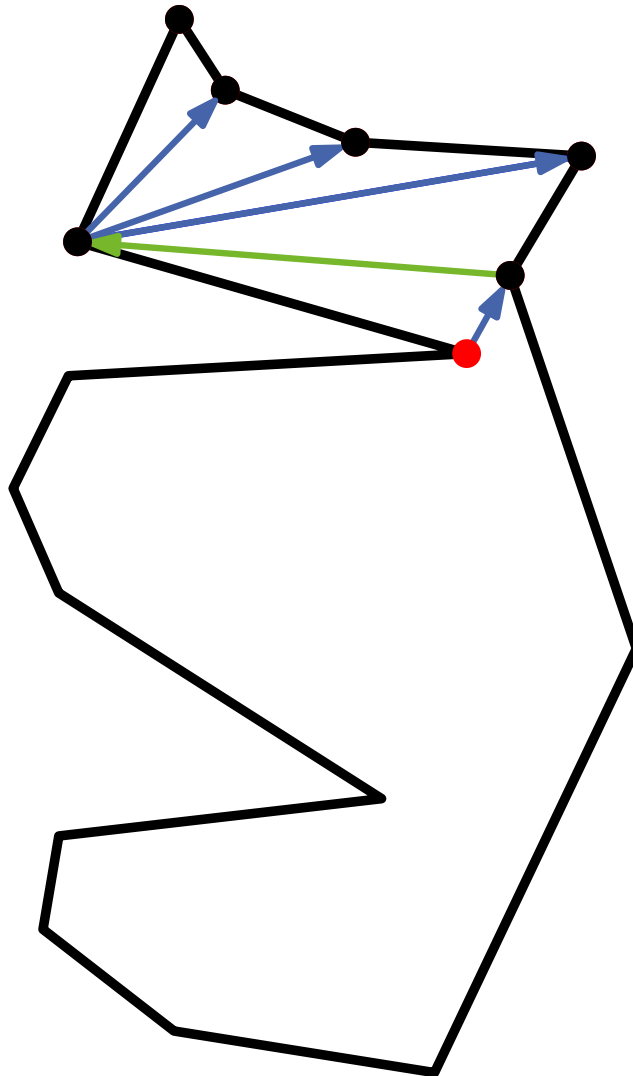
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

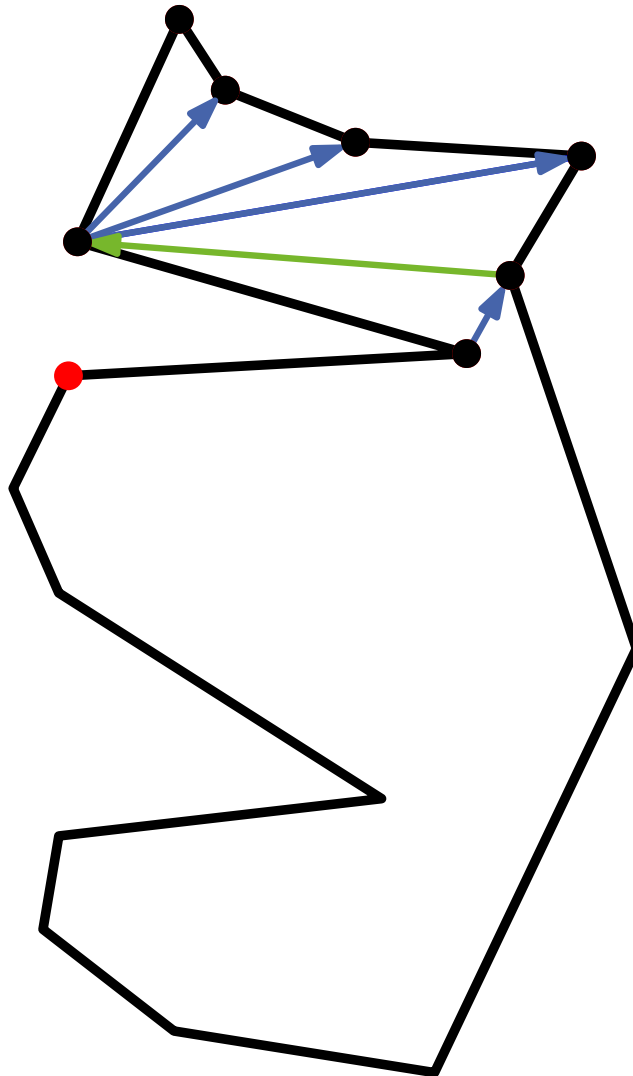
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

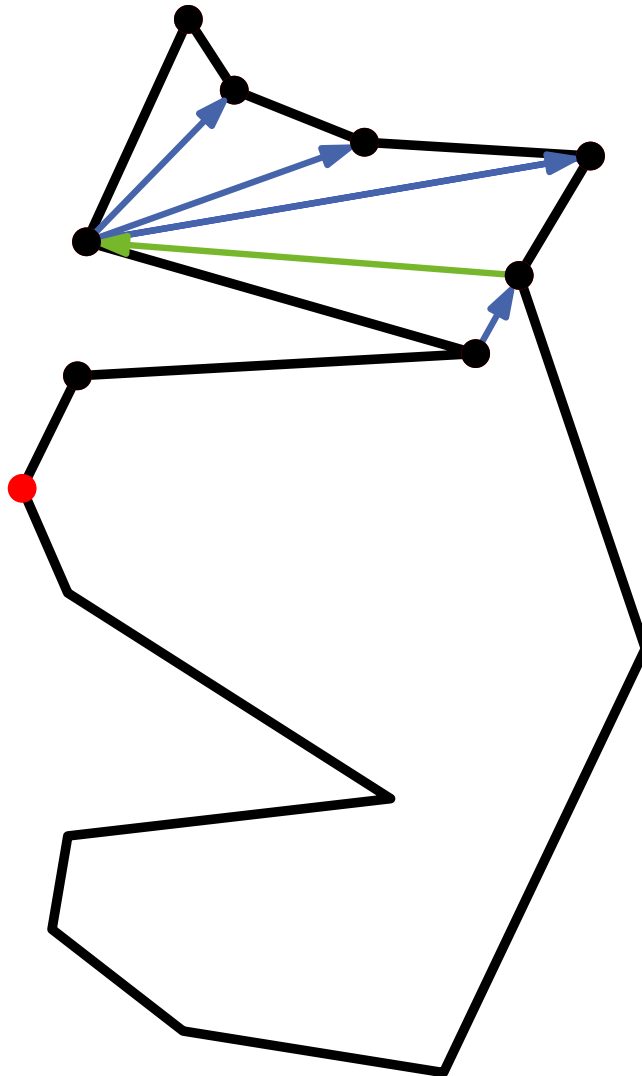
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

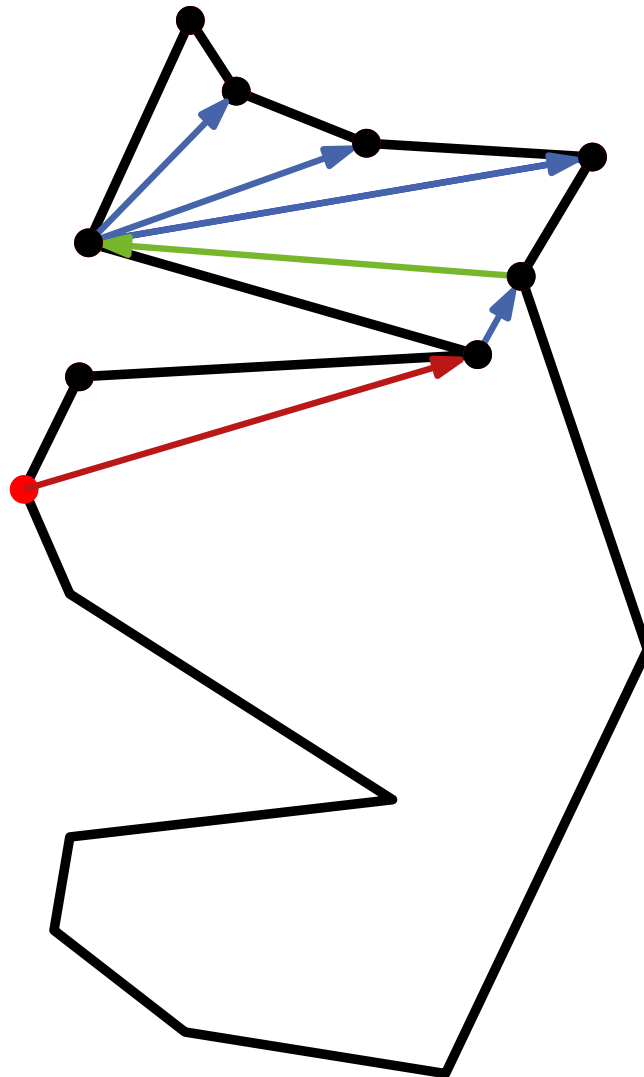
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

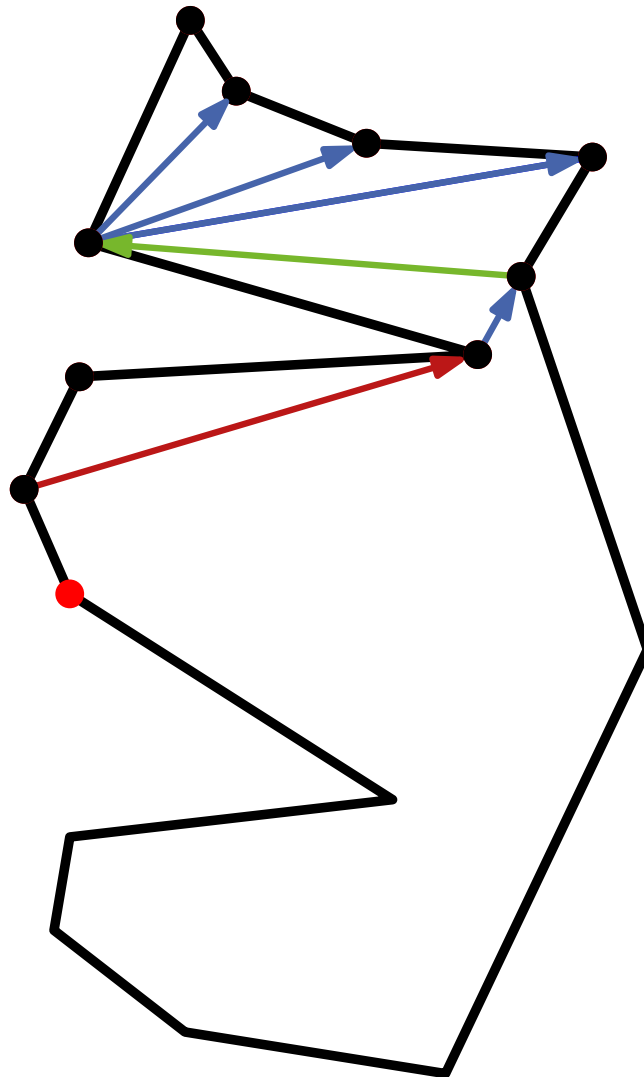
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

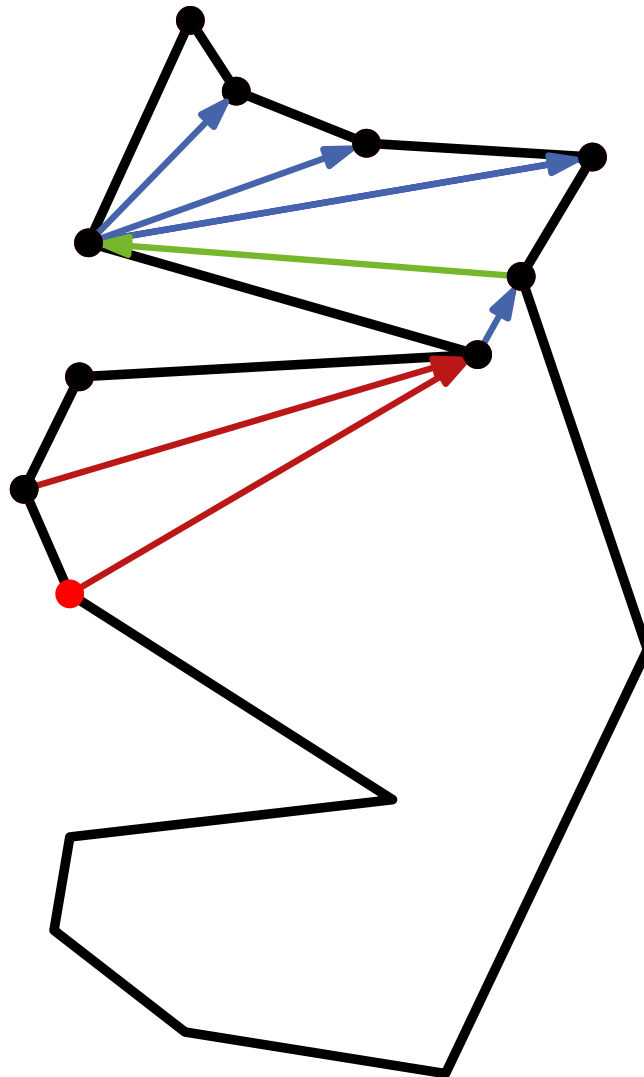
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

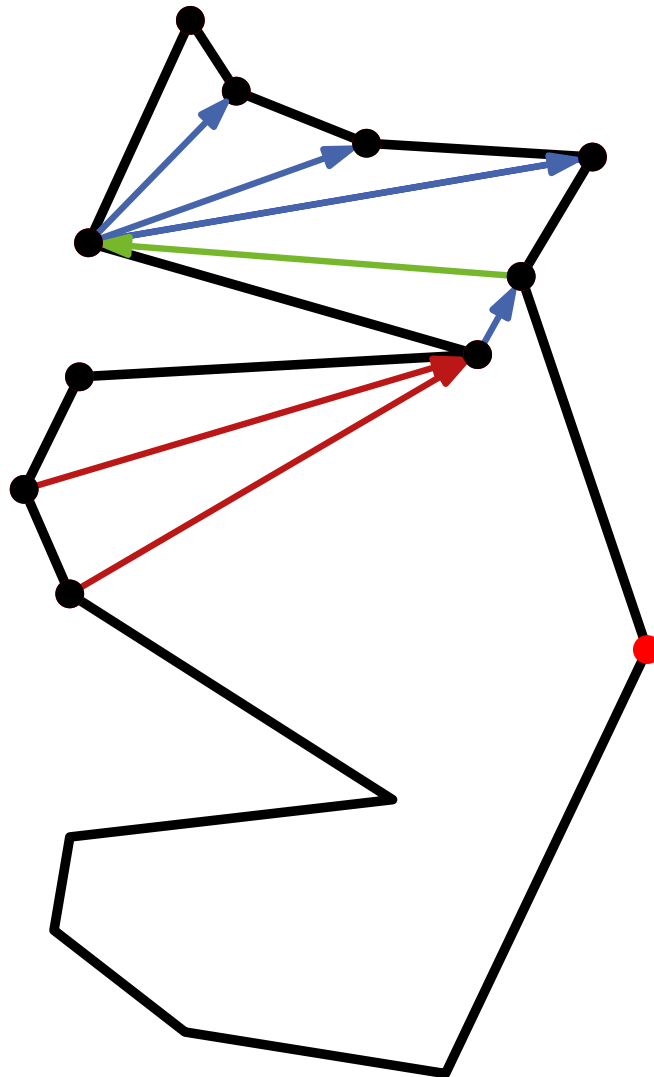
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

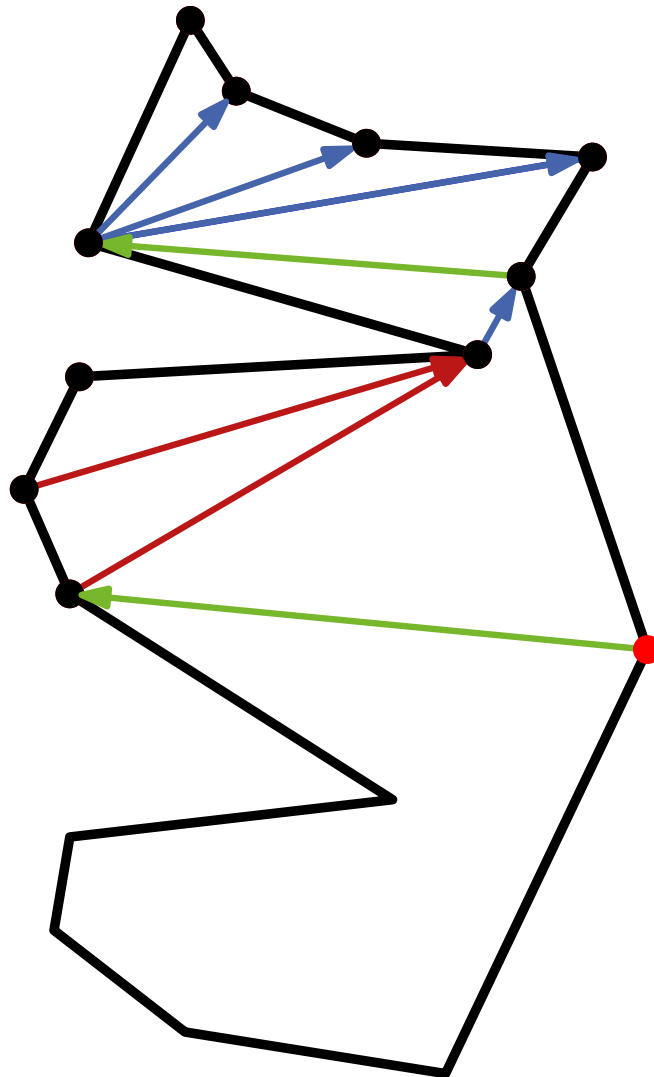
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

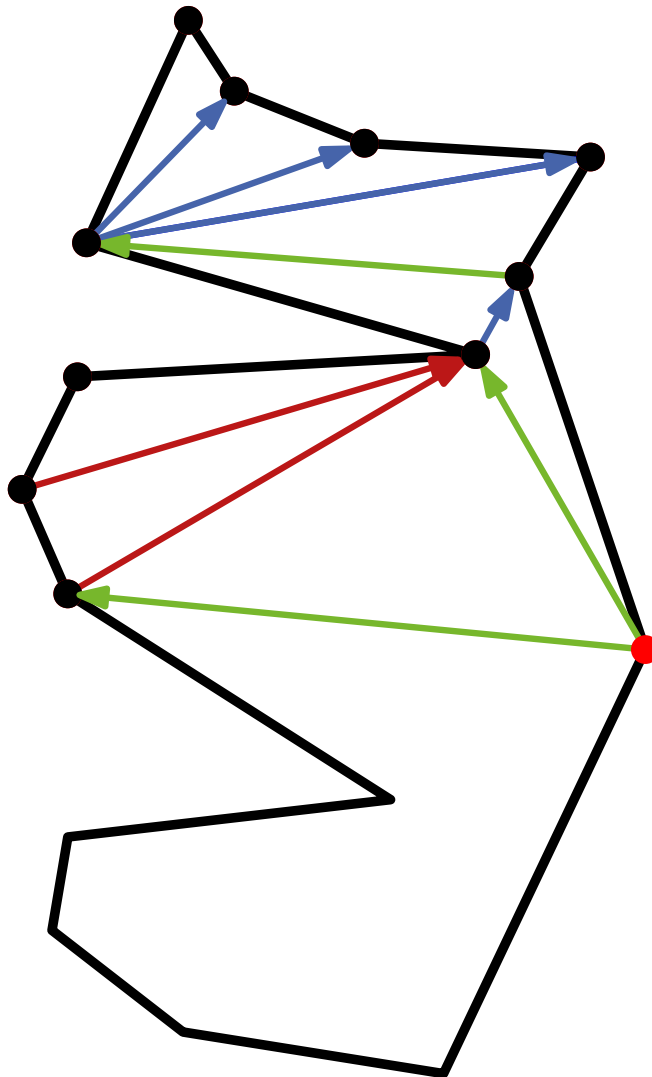
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

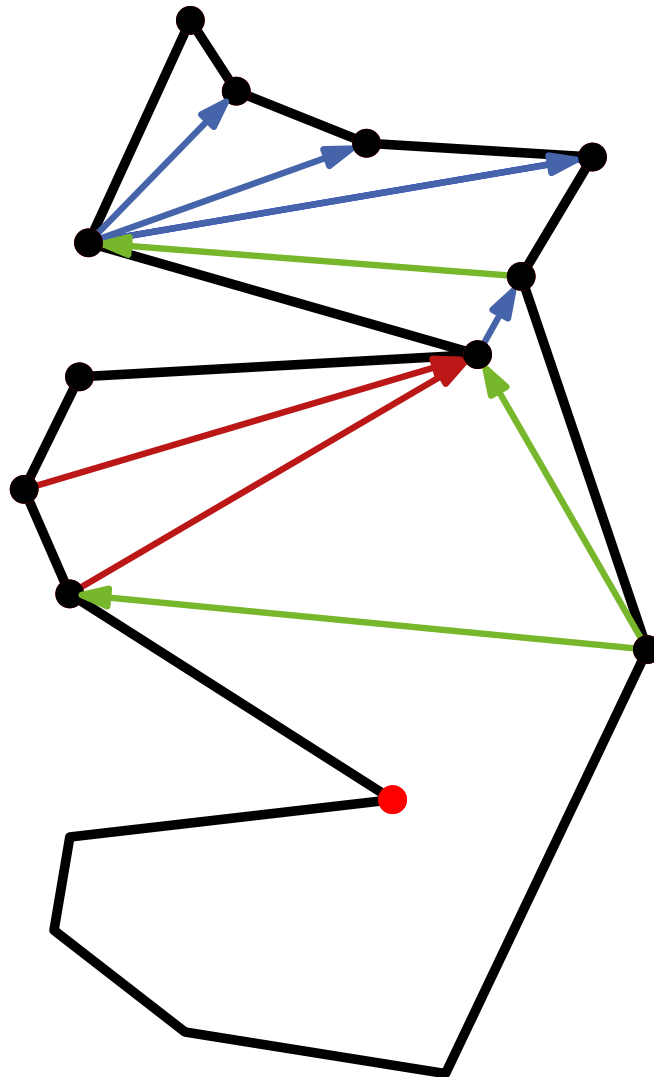
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

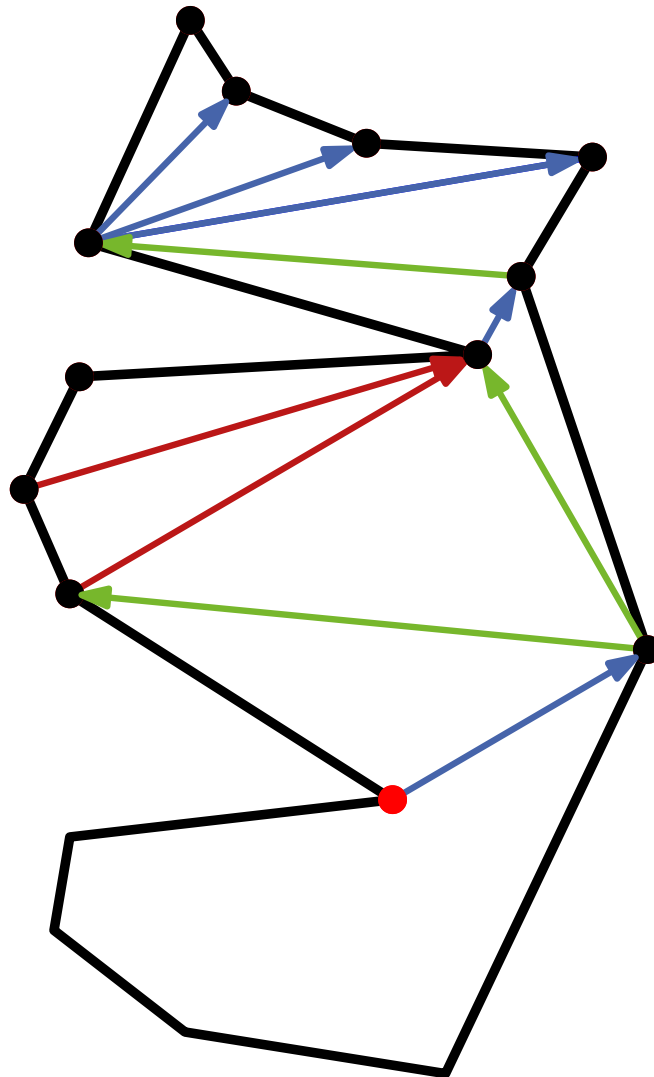
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

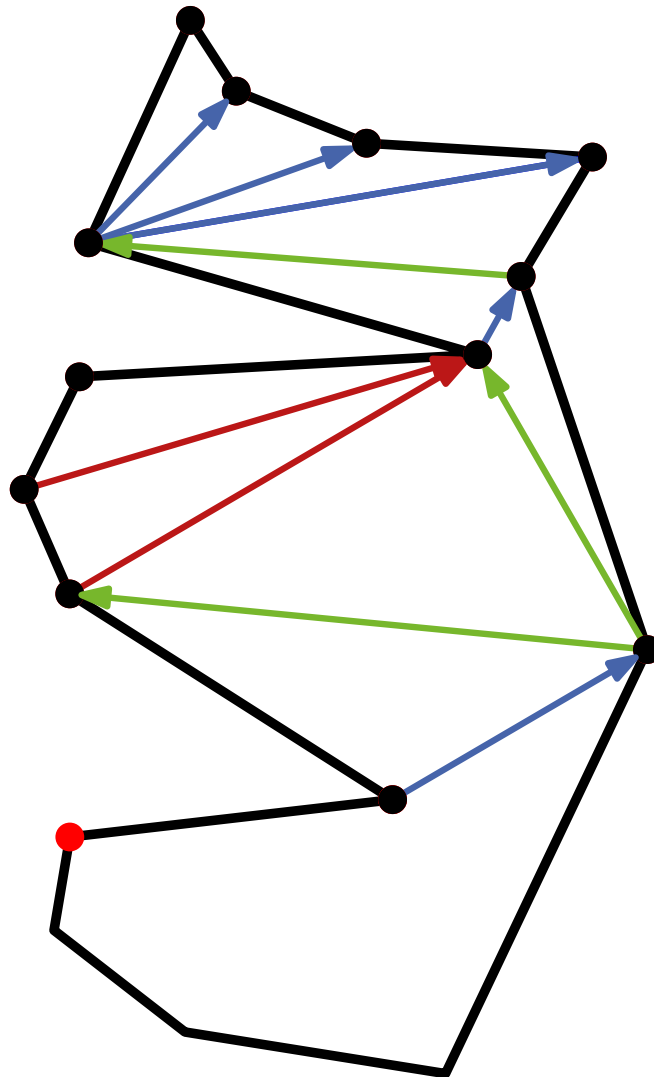
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

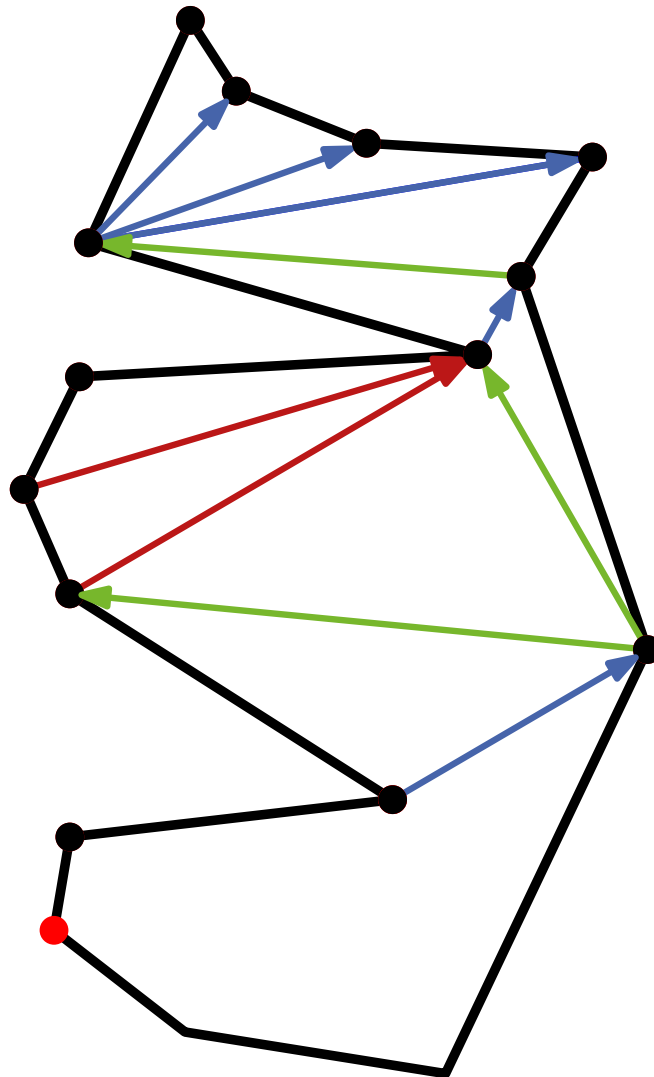
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

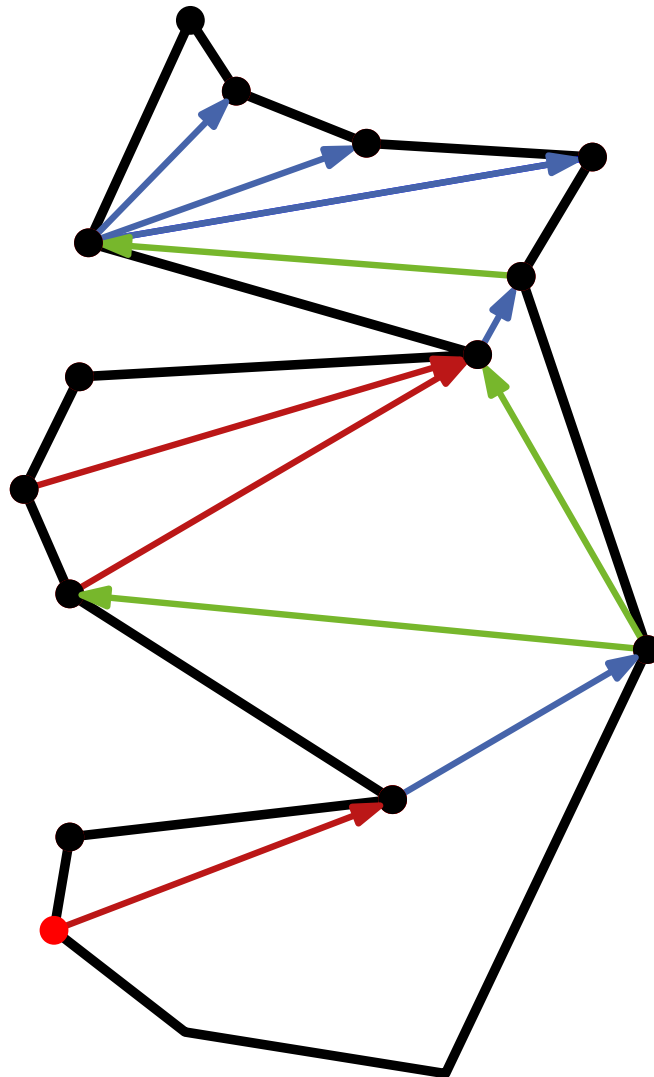
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

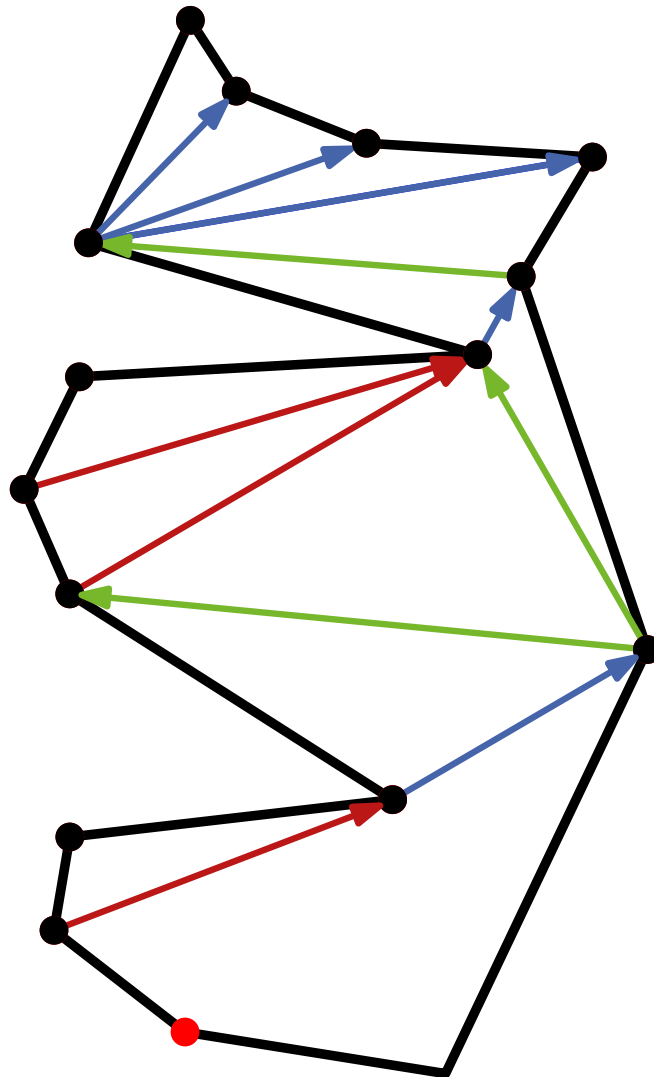
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

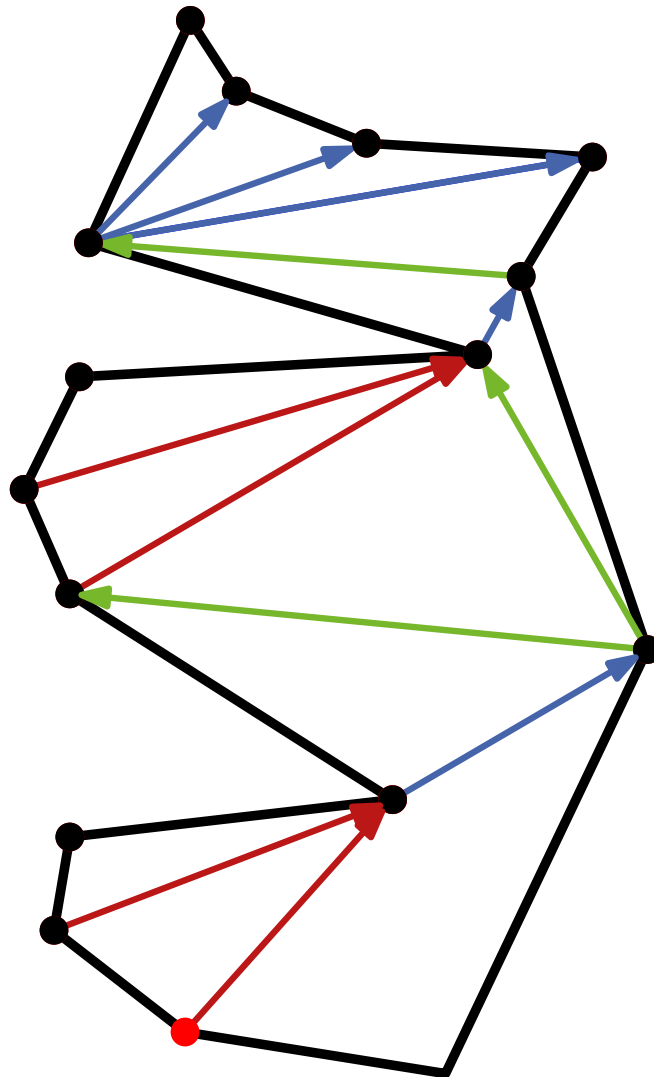
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

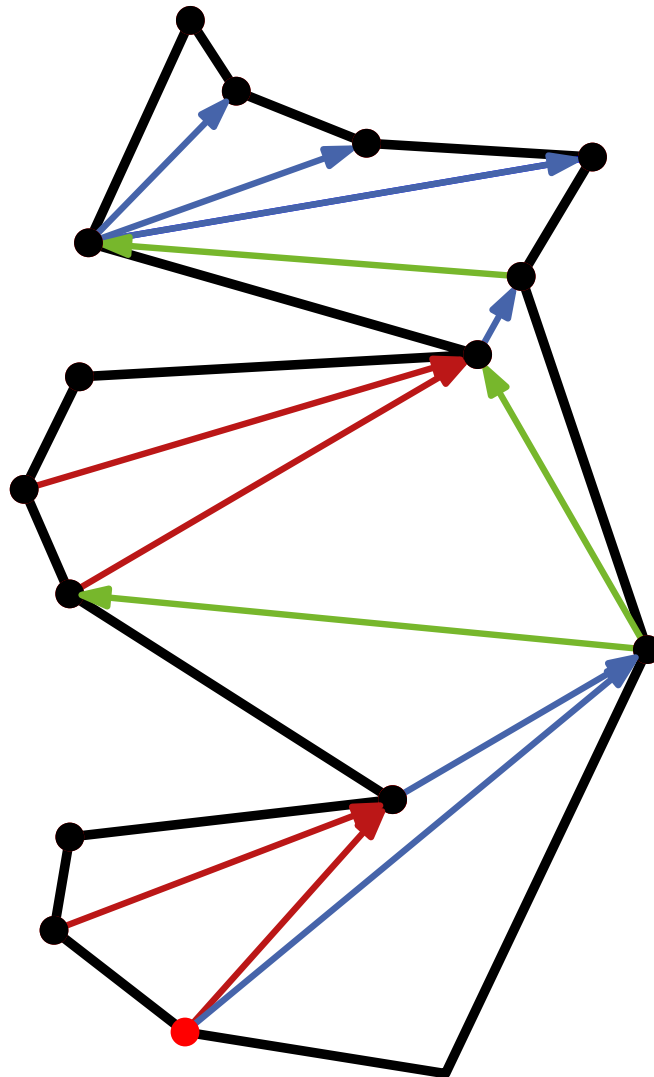
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

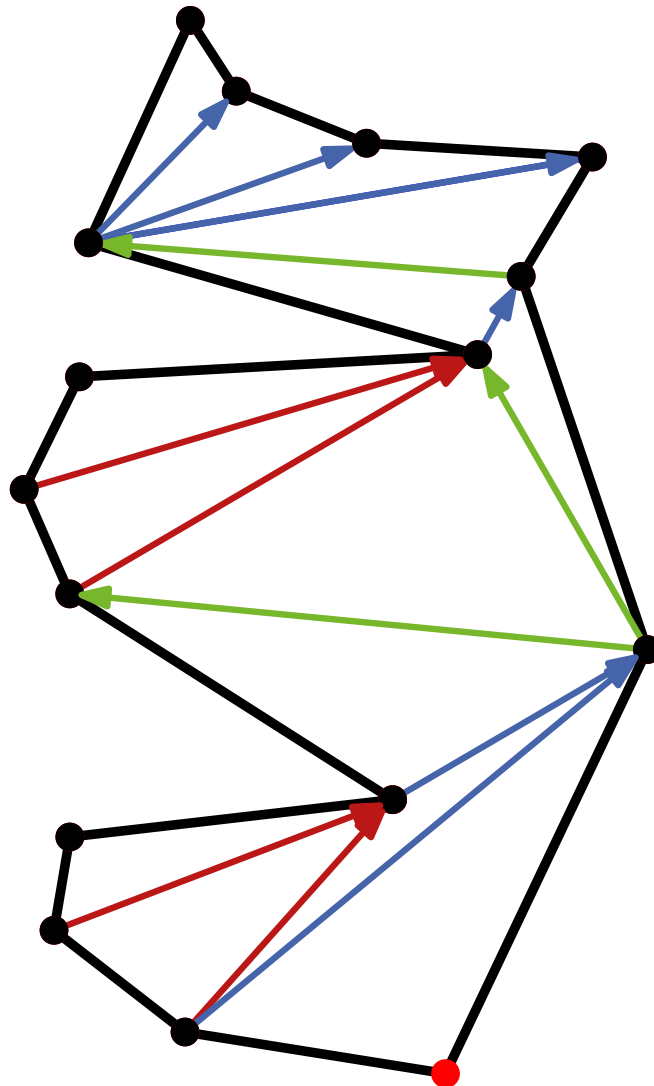
Ansatz: greedy, auf beiden Seiten von oben nach unten



Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

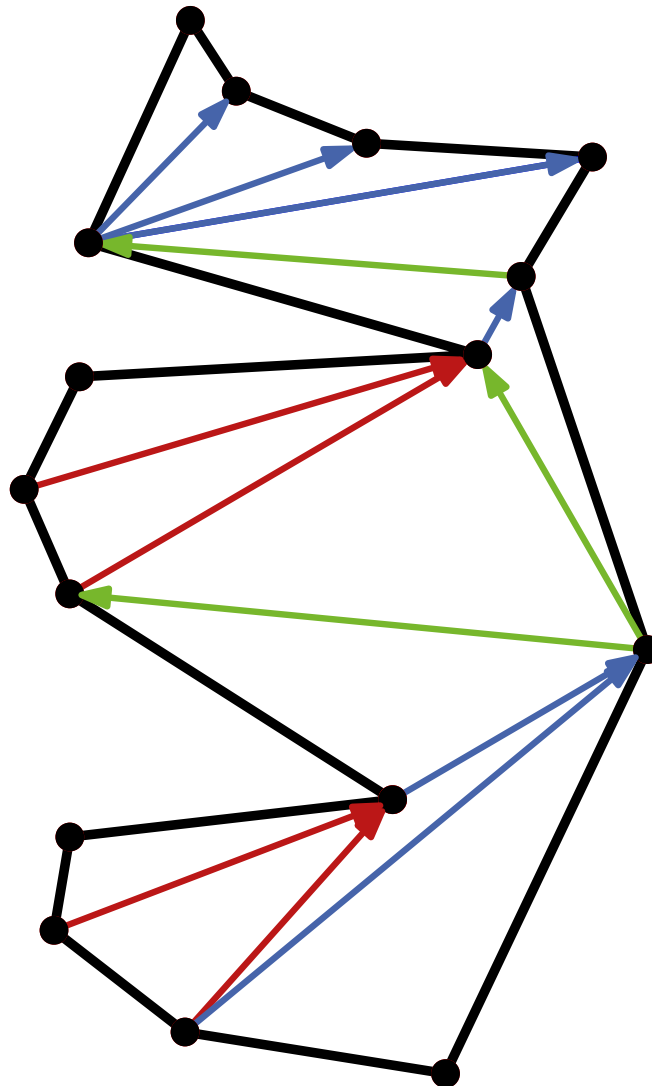


Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Invariante?

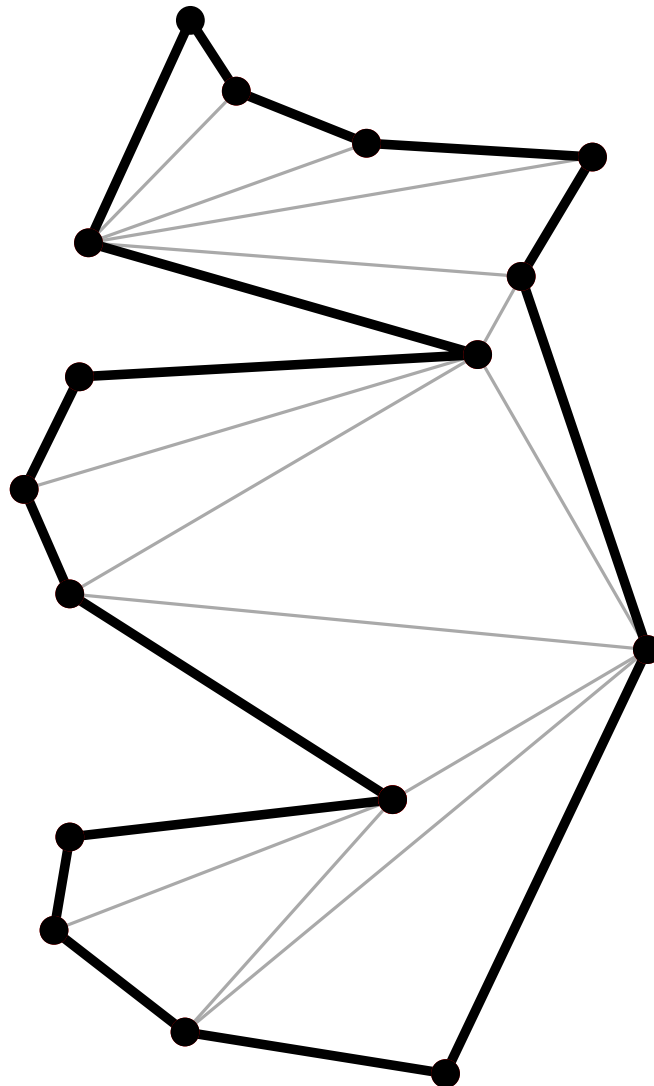


Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Invariante?

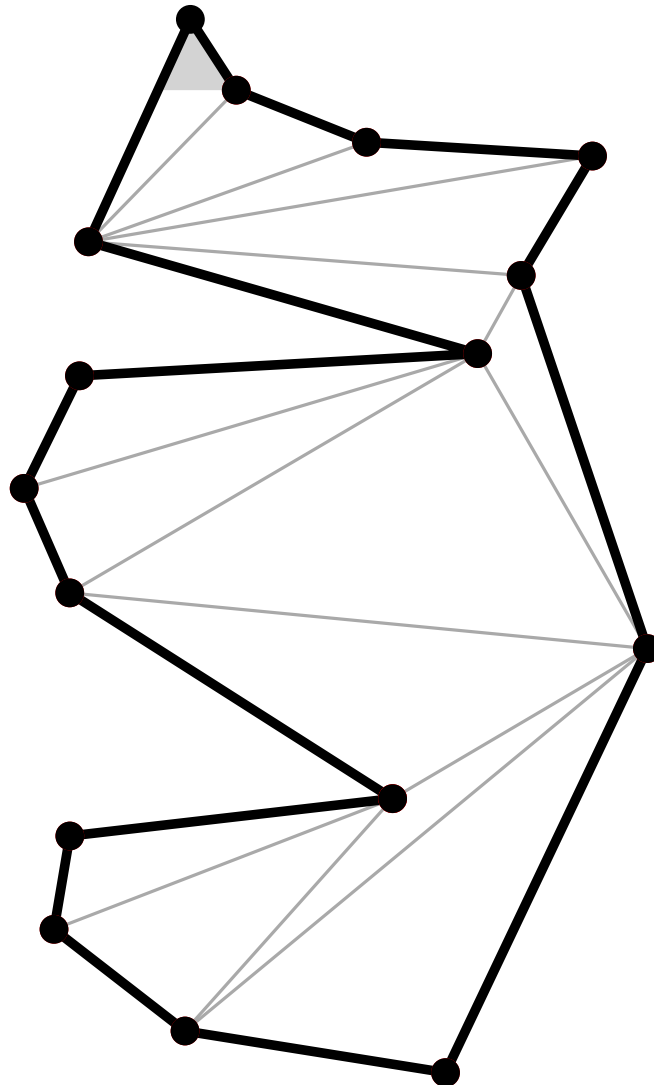


Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Invariante?

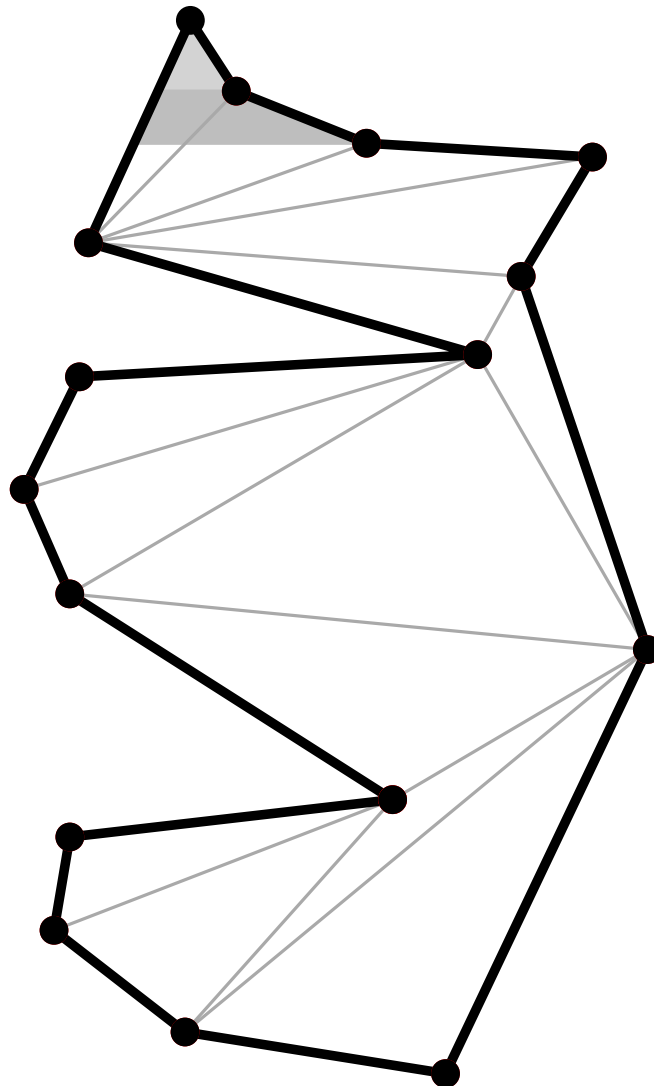


Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Invariante?

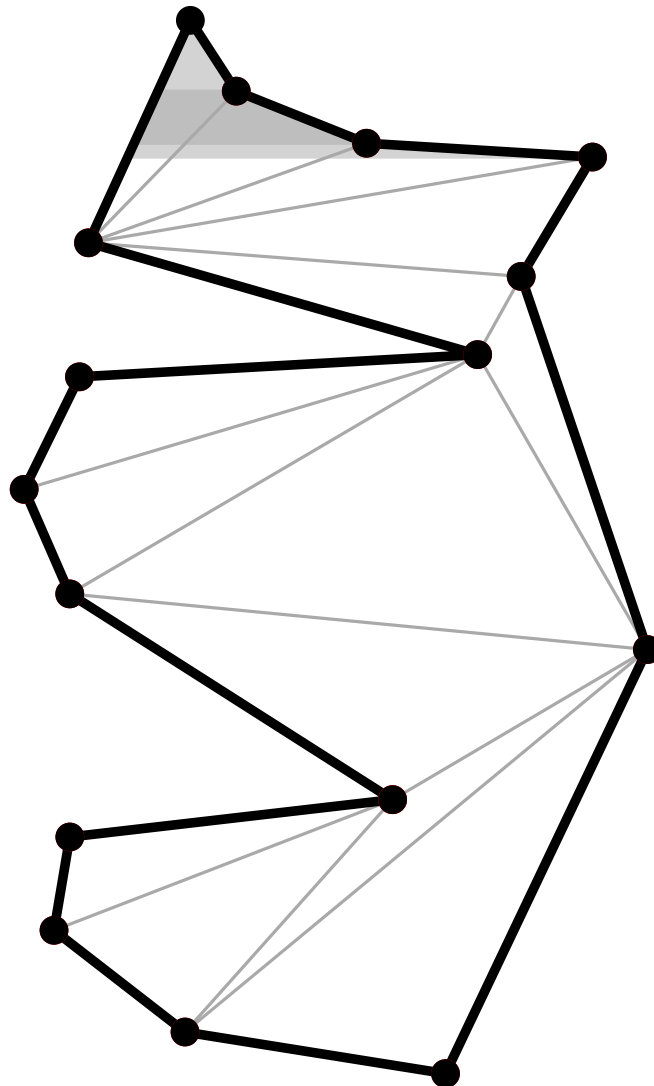


Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Invariante?

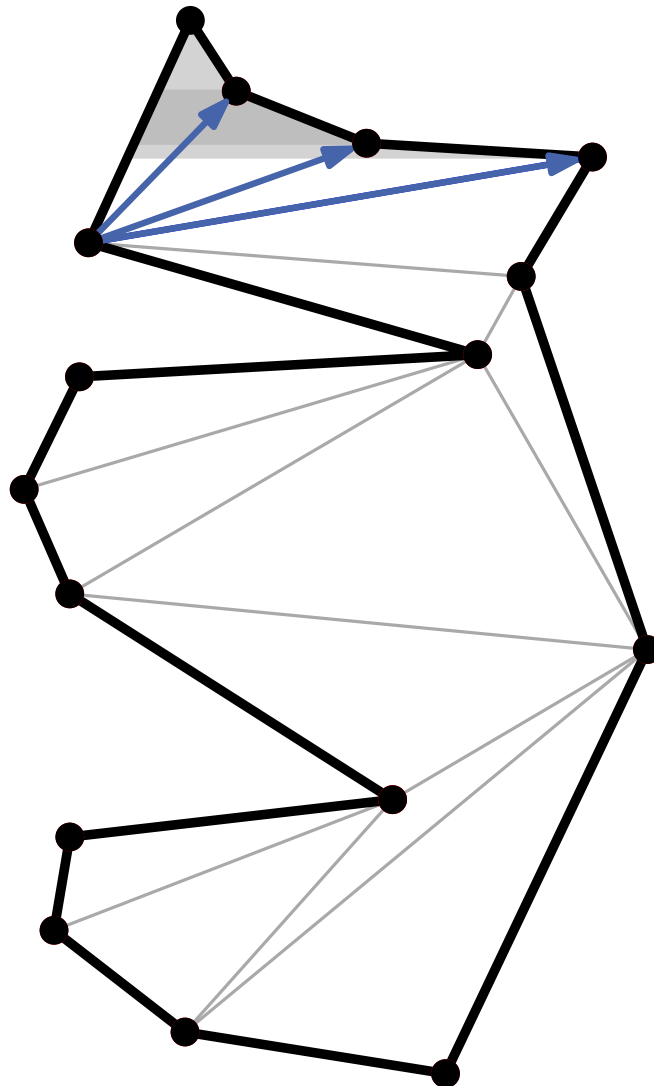


Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

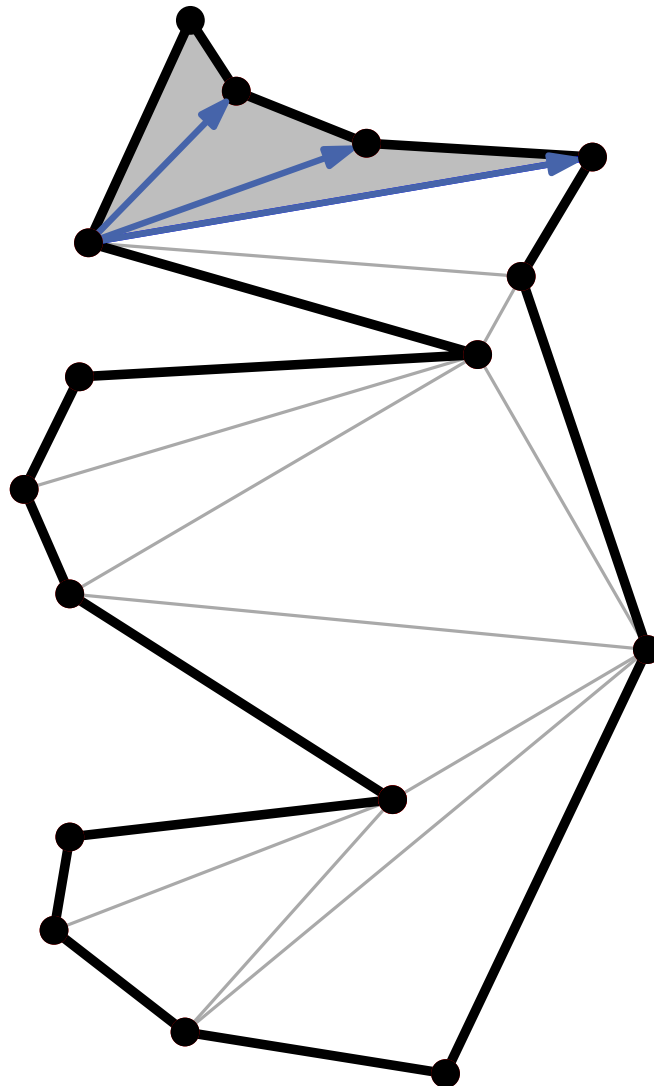
Invariante?



Triangulieren von y -monotone Polygonen

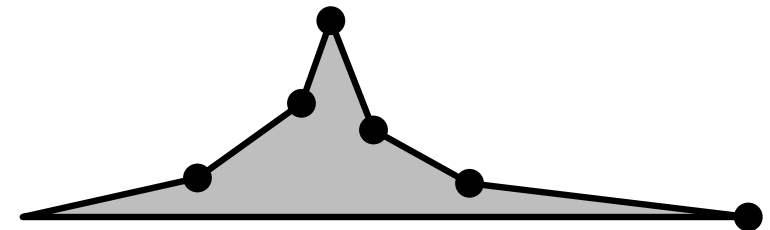
Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten



Invariante?

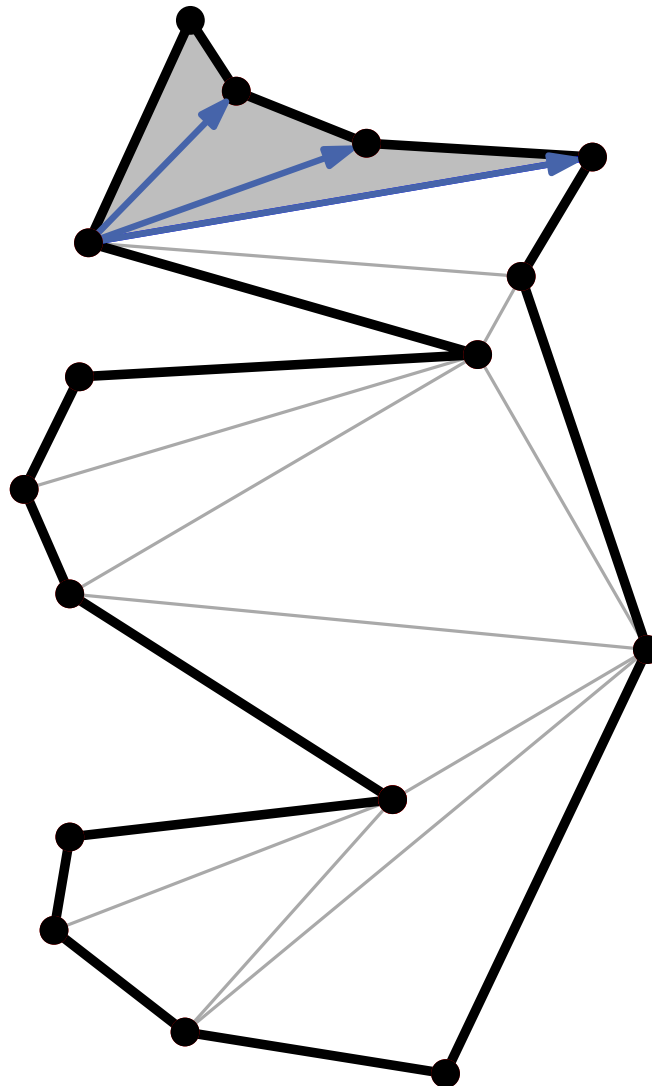
Der besuchte, aber noch nicht triangulierte Teil von P ist *trichterförmig*.



Triangulieren von y -monotone Polygonen

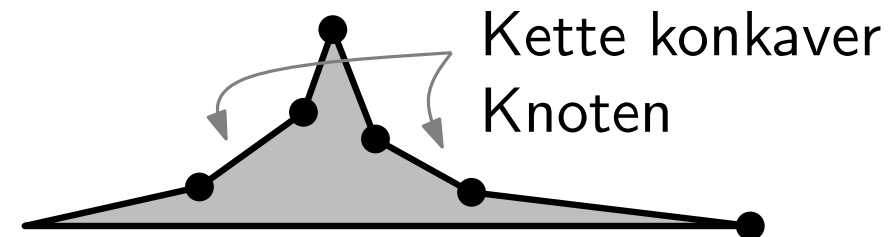
Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten



Invariante?

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



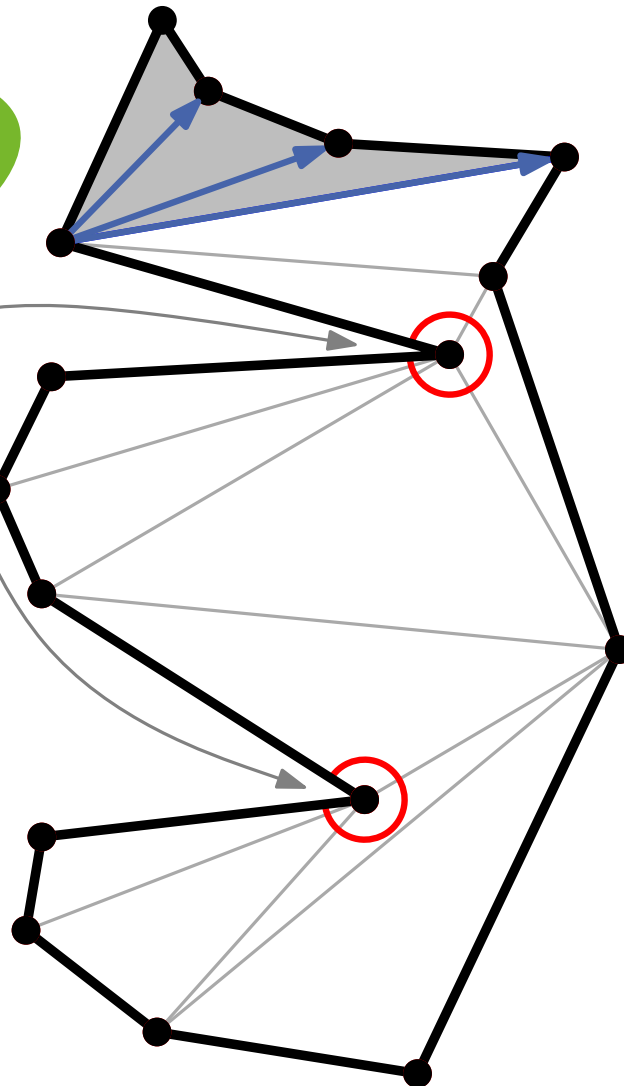
Triangulieren von y -monotone Polygonen

Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

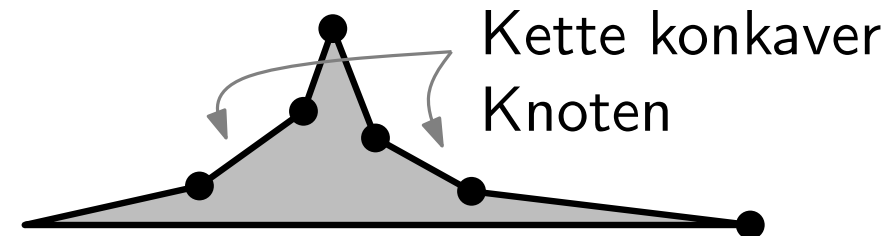
Winkel in P
 $> 180^\circ$

konkav



Invariante?

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



Triangulieren von y -monotone Polygonen

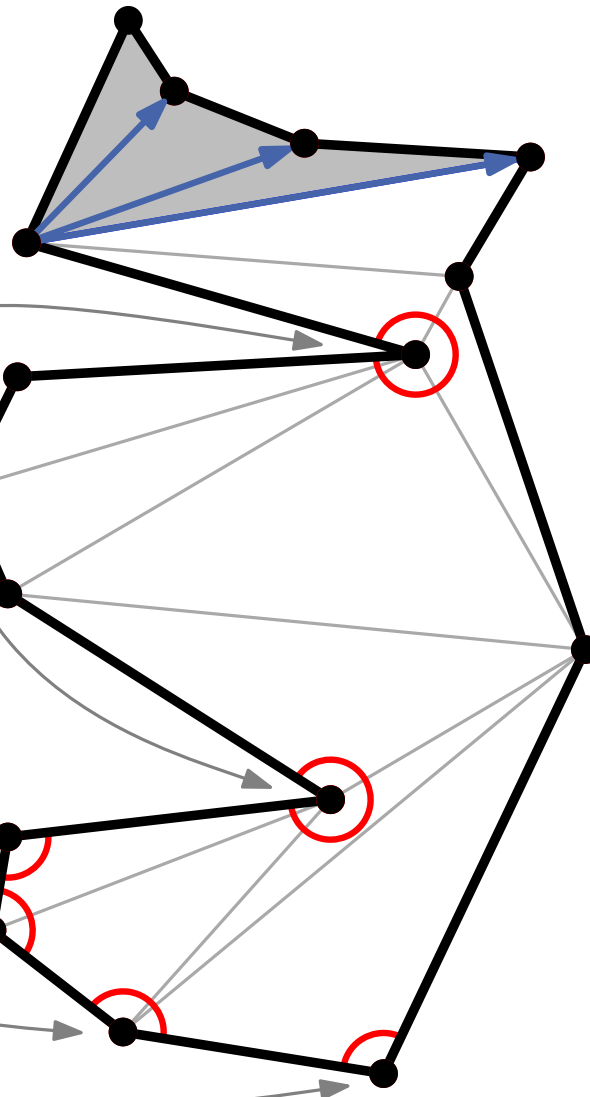
Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Winkel in P
 $> 180^\circ$

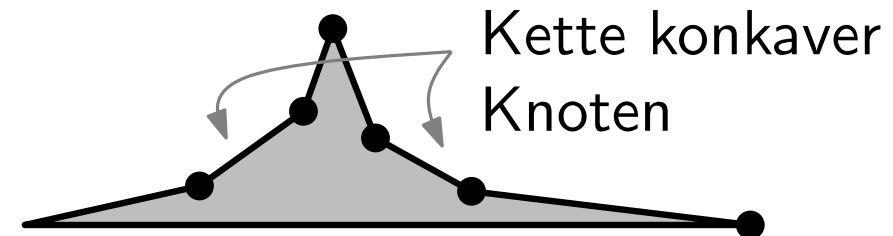
konkav

konvex



Invariante?

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



Triangulieren von y -monotone Polygonen

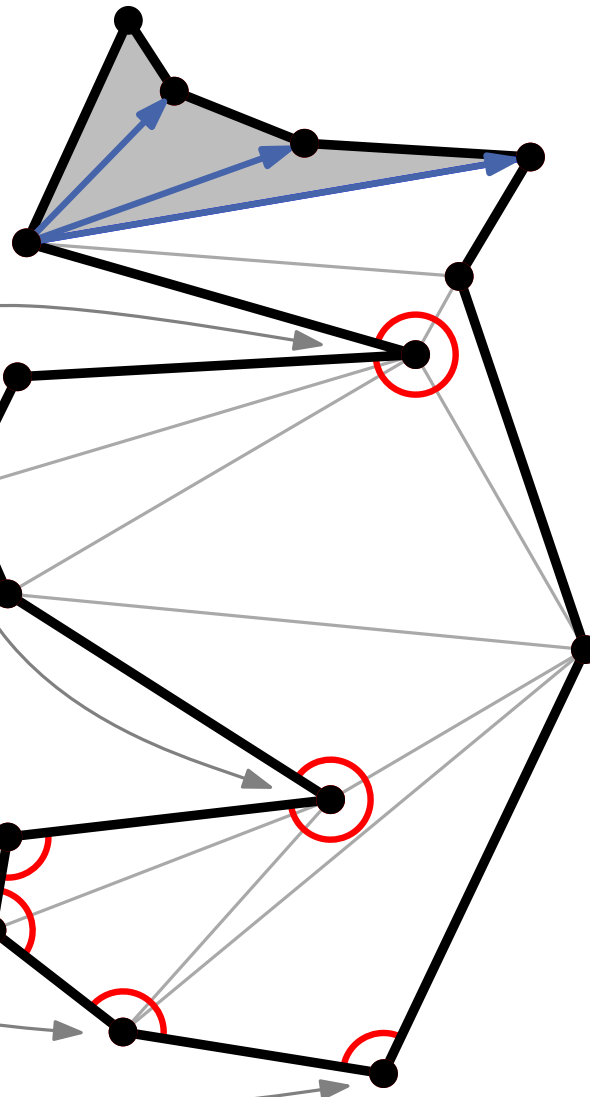
Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Winkel in P
 $> 180^\circ$

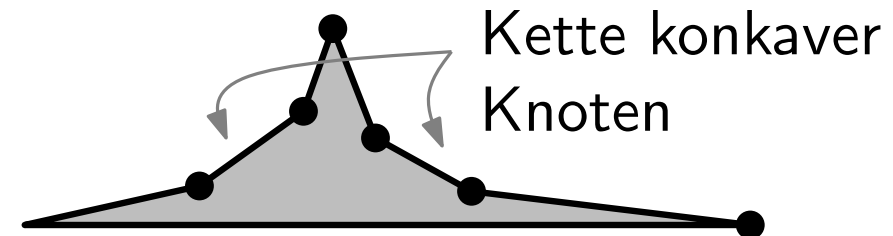
konkav

konvex

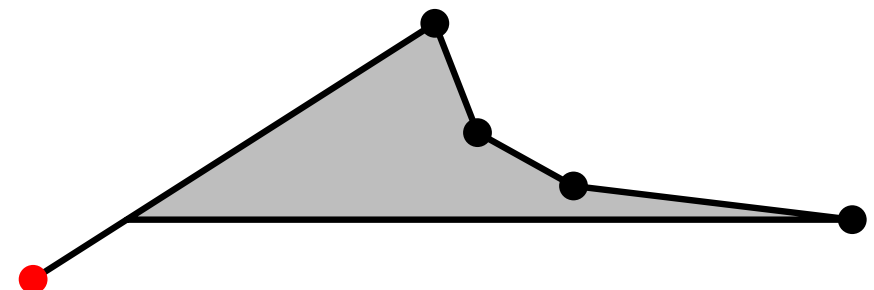


Invariante?

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



genauer hinschauen:



Triangulieren von y -monotone Polygonen

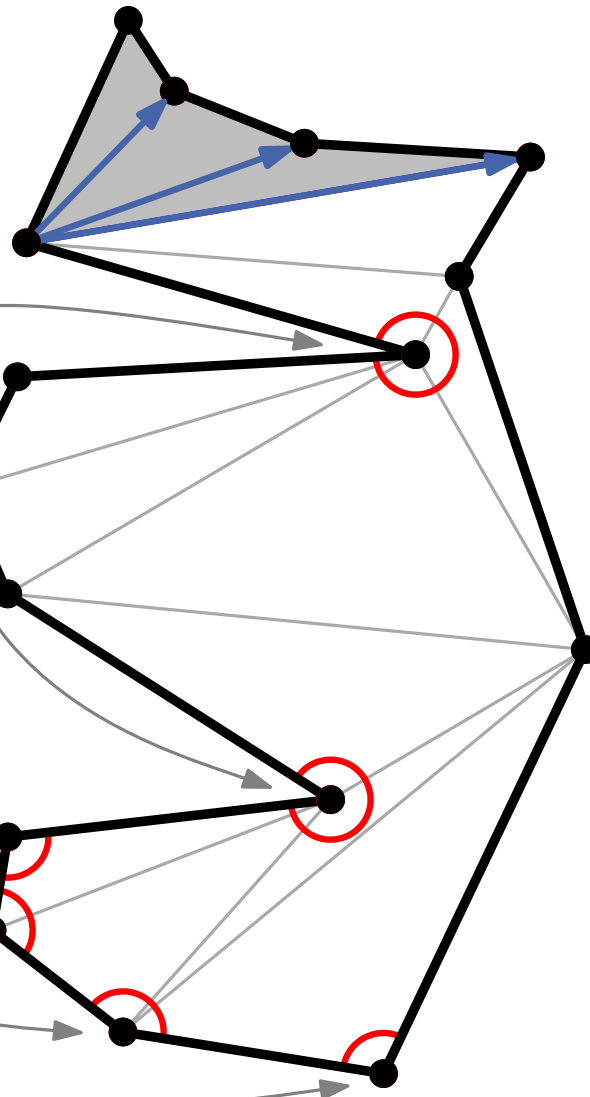
Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Winkel in P
 $> 180^\circ$

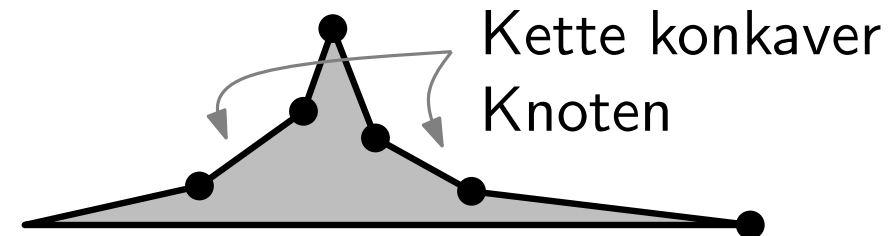
konkav

konvex

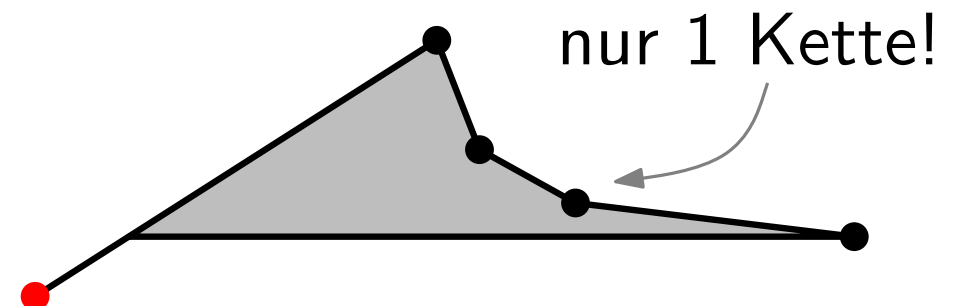


Invariante?

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



genauer hinschauen:



Triangulieren von y -monotone Polygonen

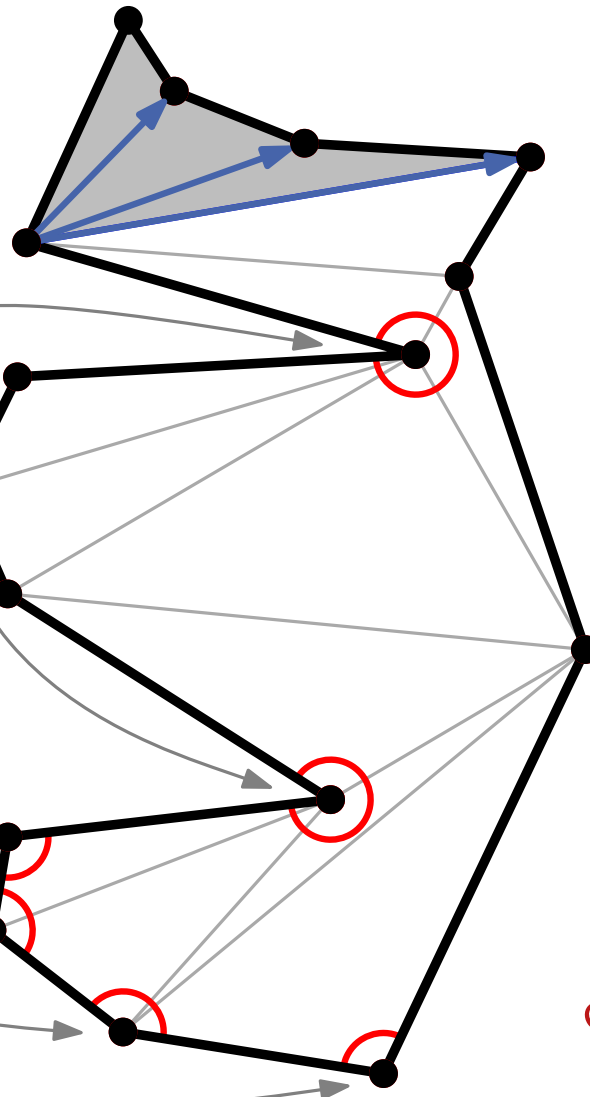
Erinnerung: linker und rechter Grenzpfad sind absteigend

Ansatz: greedy, auf beiden Seiten von oben nach unten

Winkel in P
 $> 180^\circ$

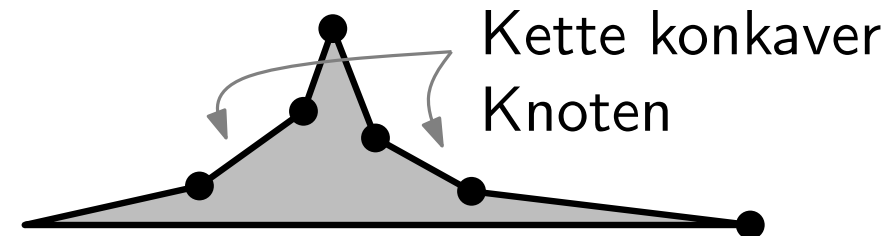
konkav

konvex

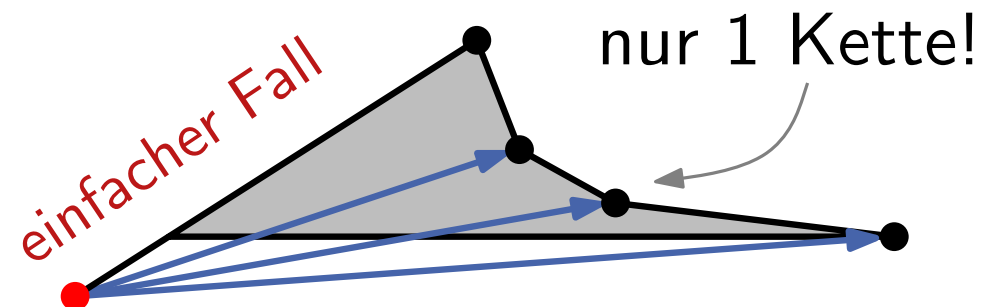


Invariante?

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 \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

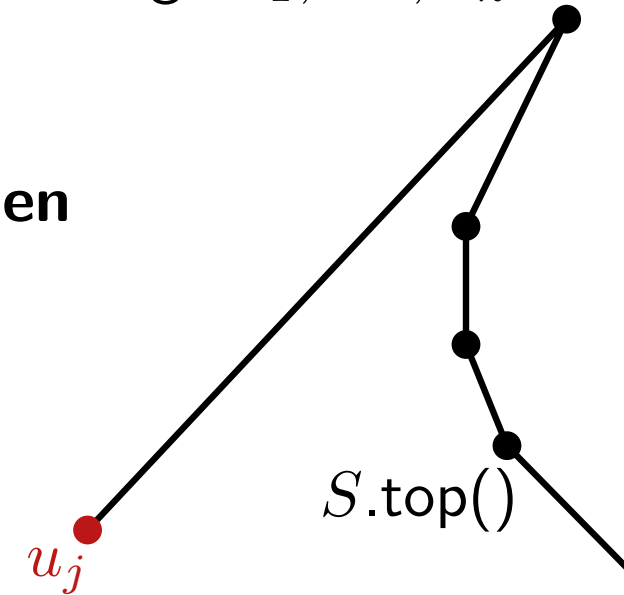
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

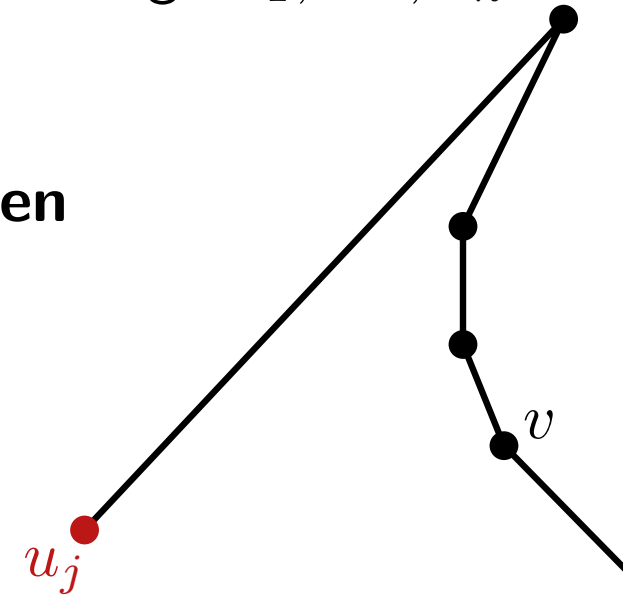
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

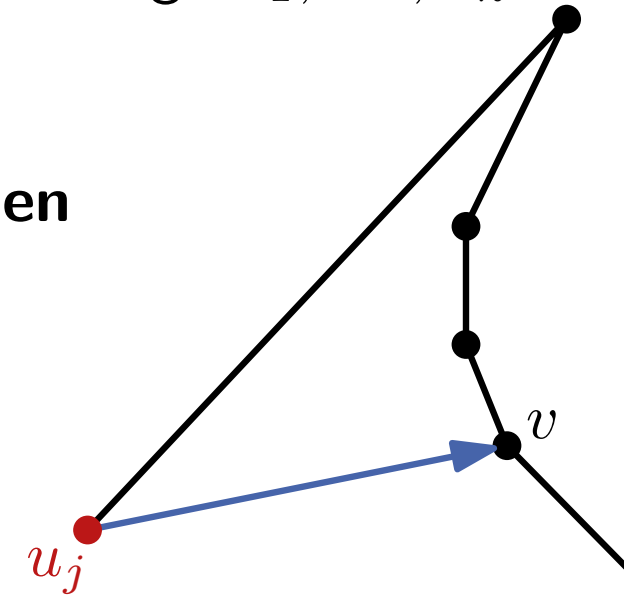
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

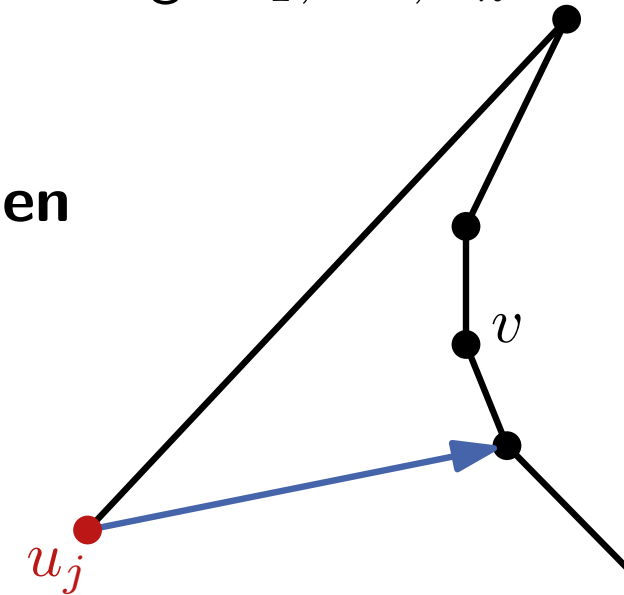
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

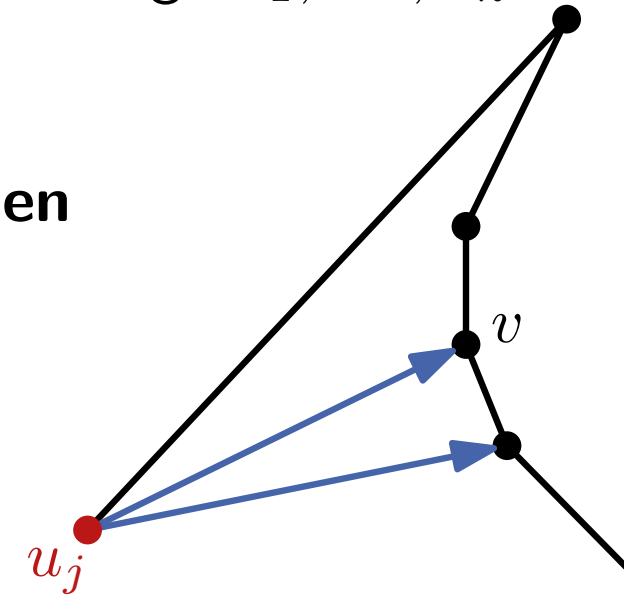
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

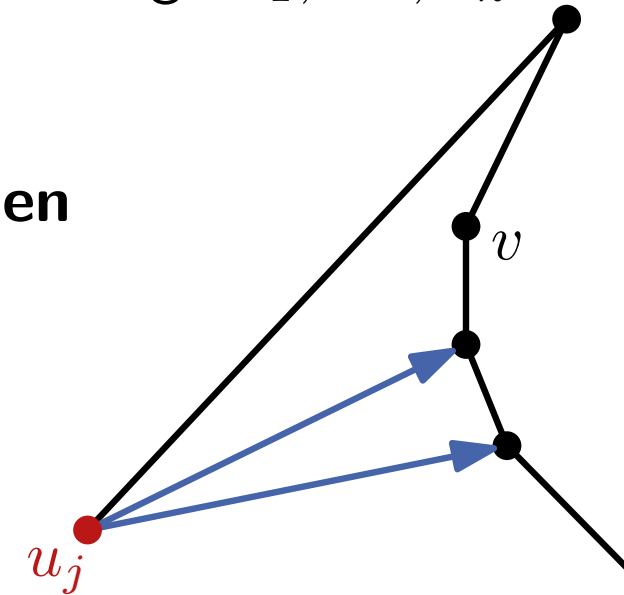
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

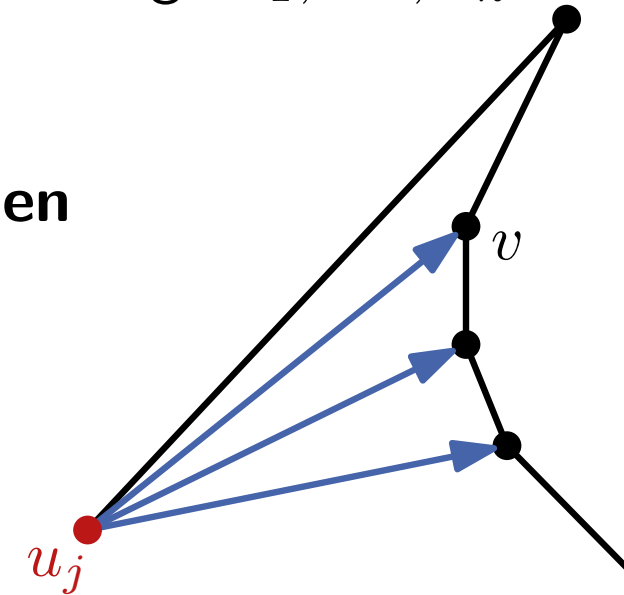
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

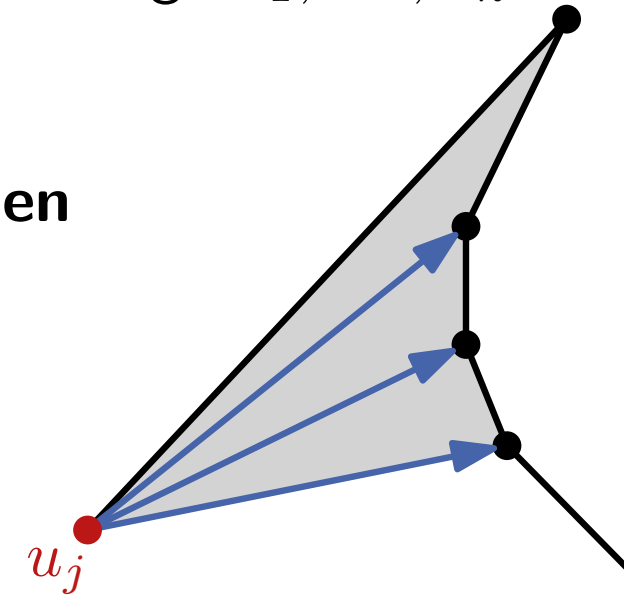
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

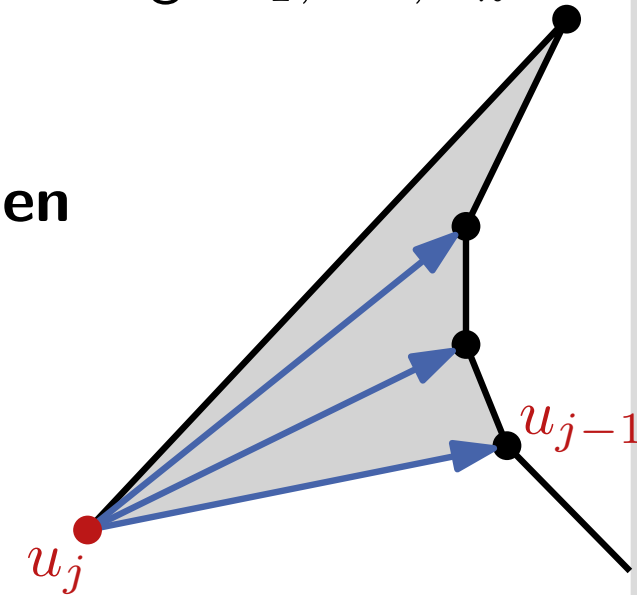
while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

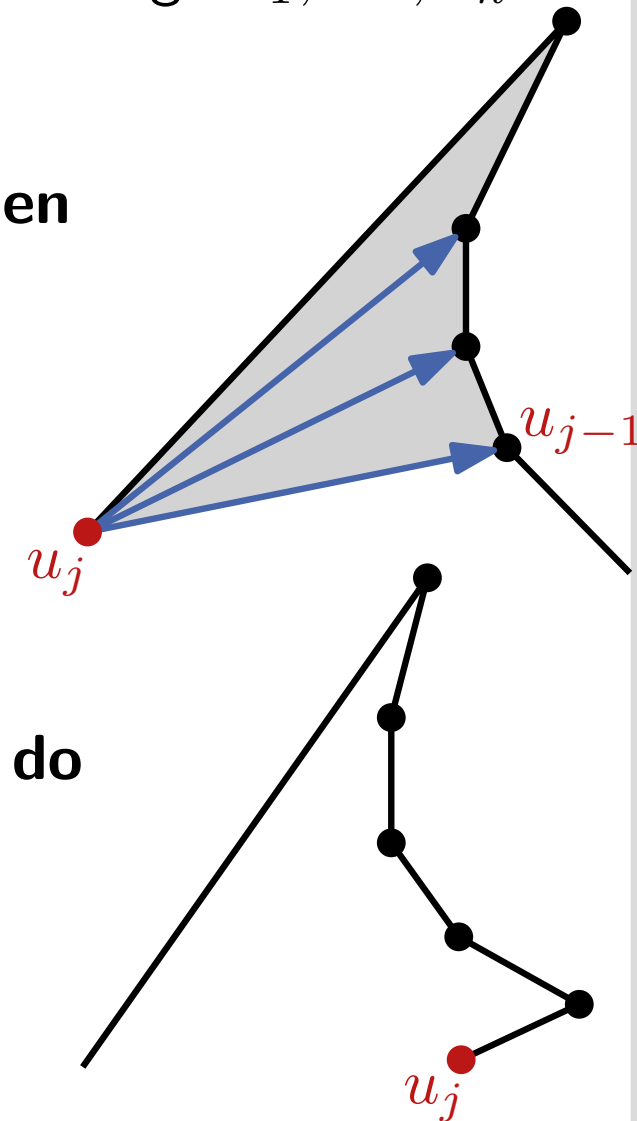
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

 zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

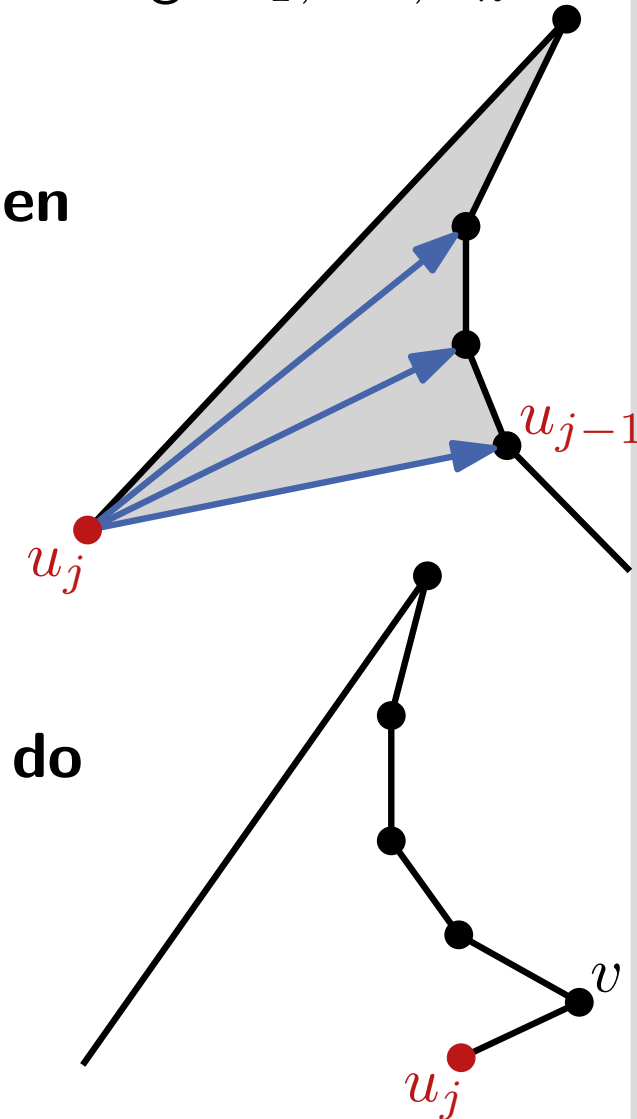
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

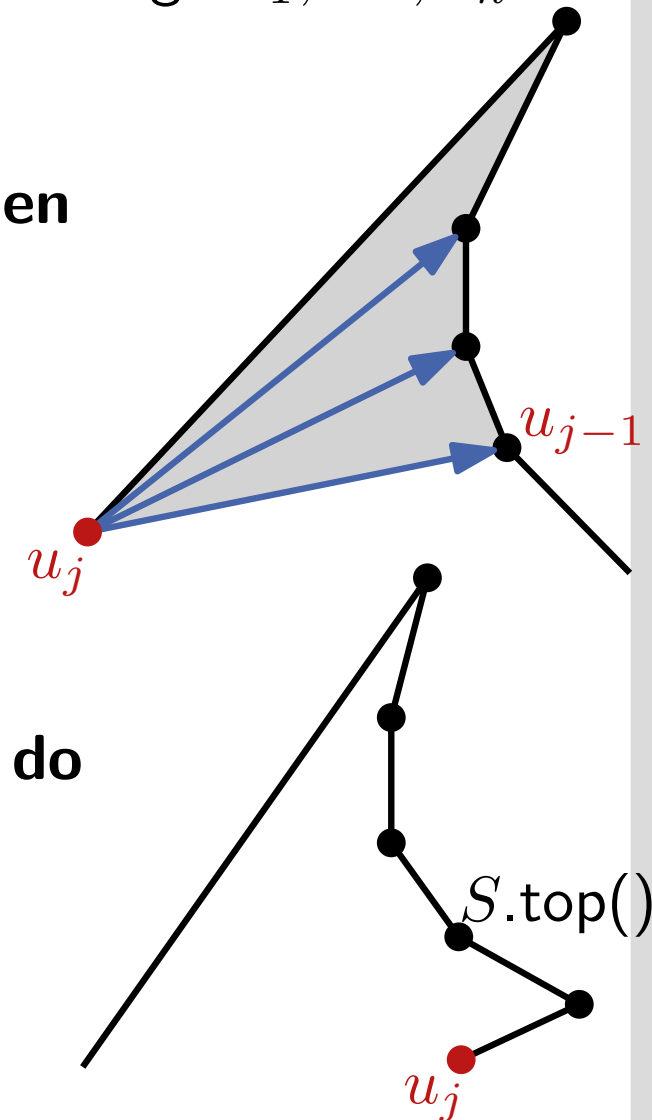
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

 zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

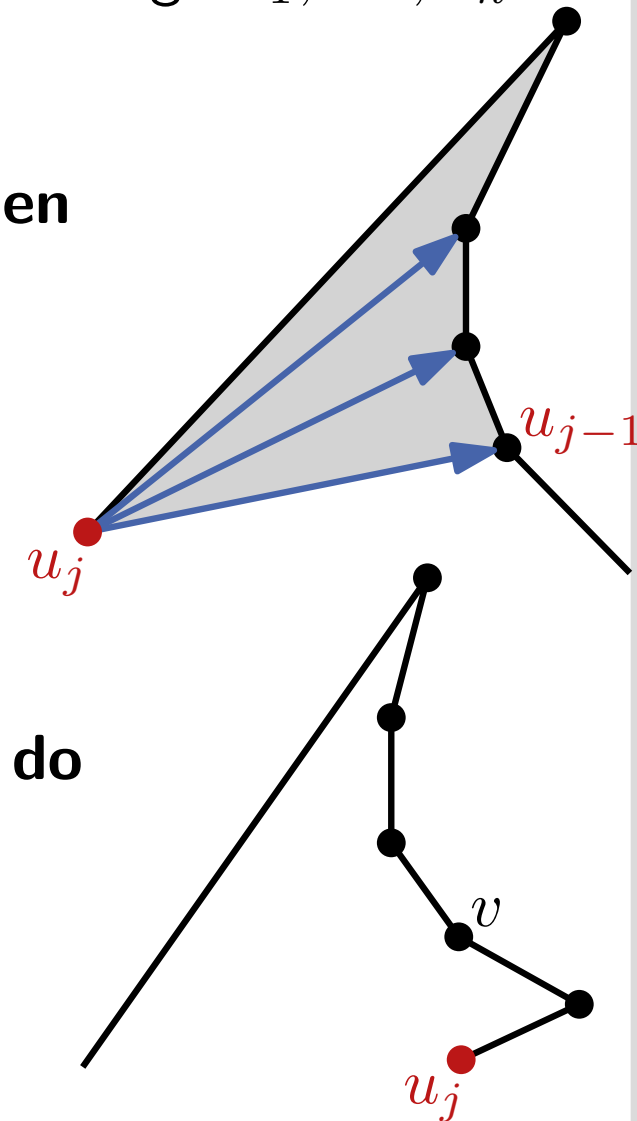
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

 zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

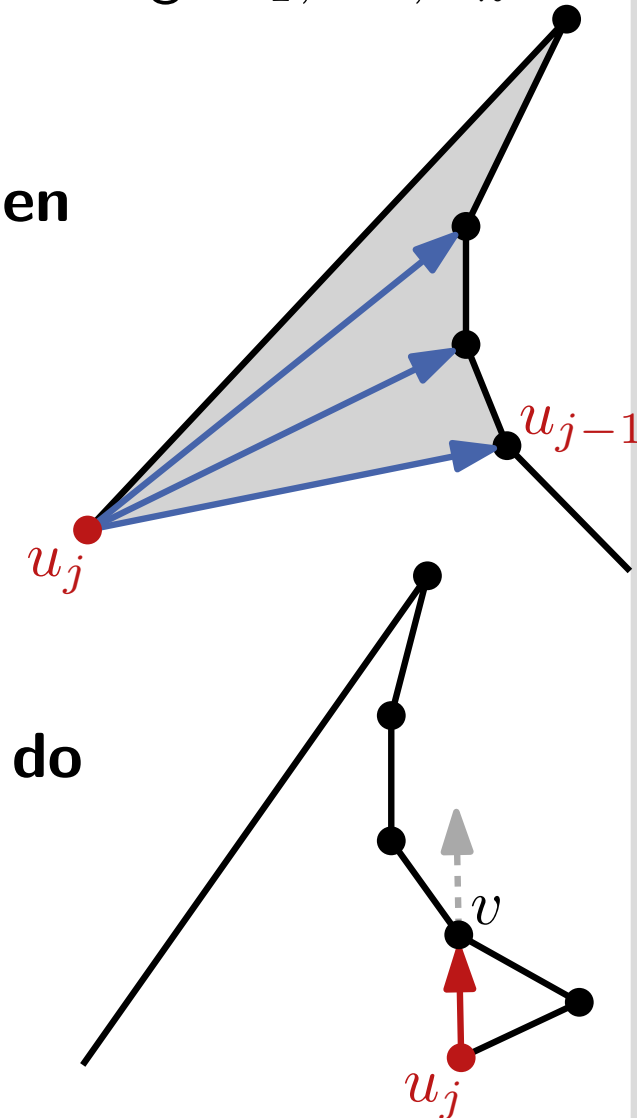
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

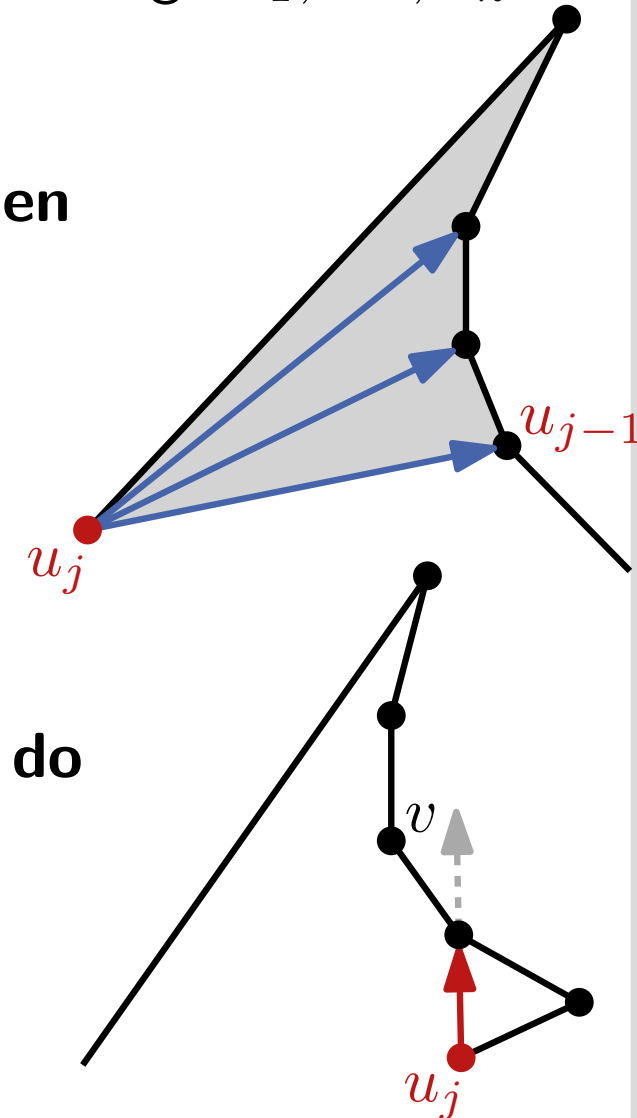
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

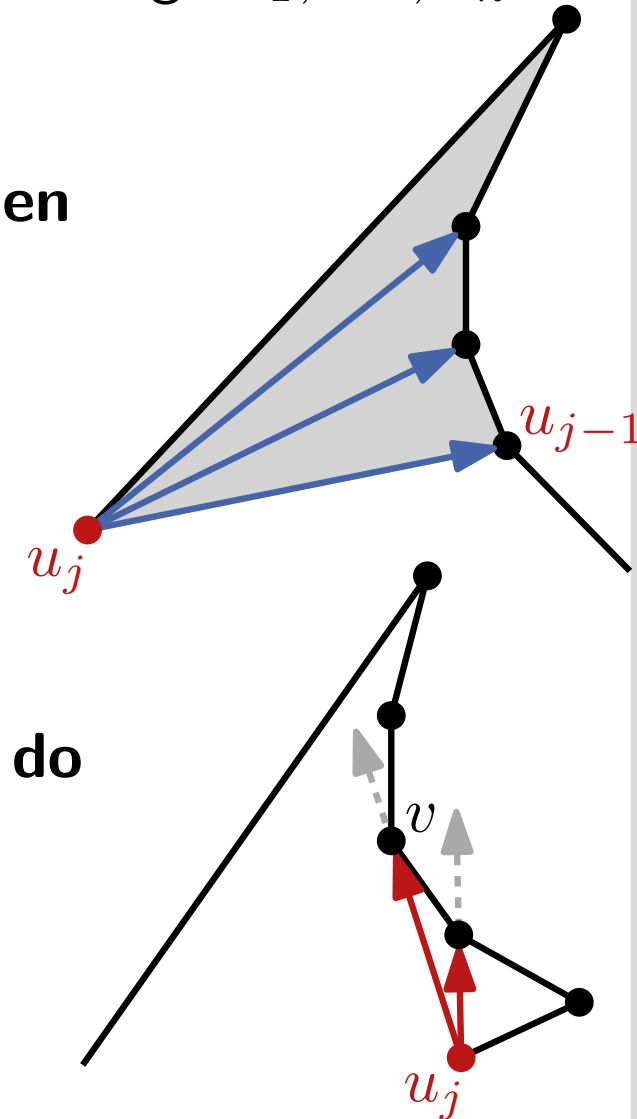
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

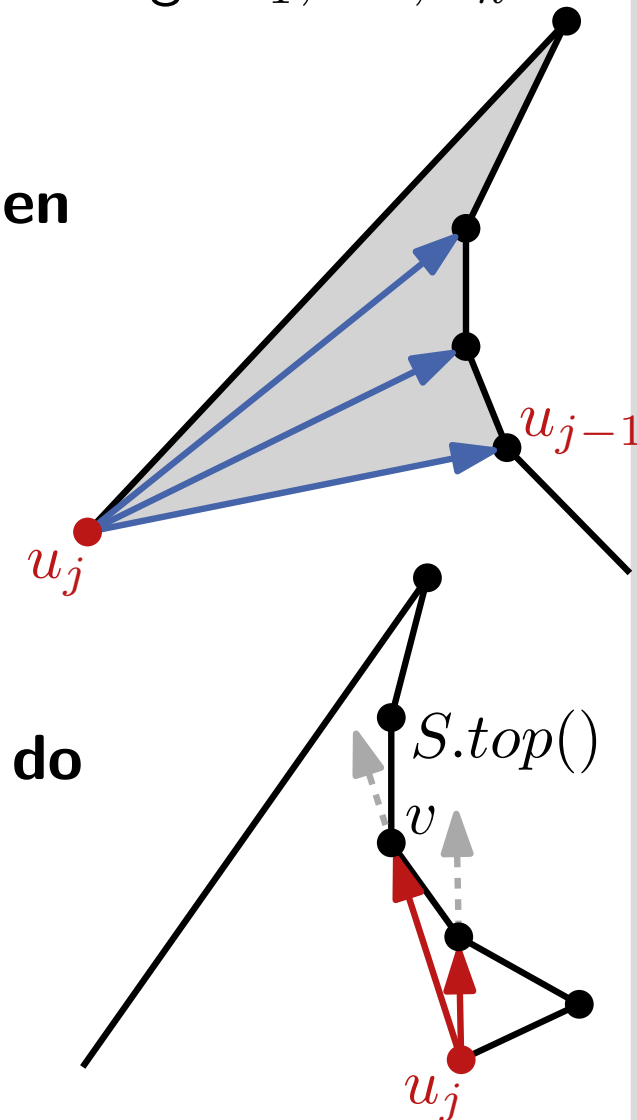
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

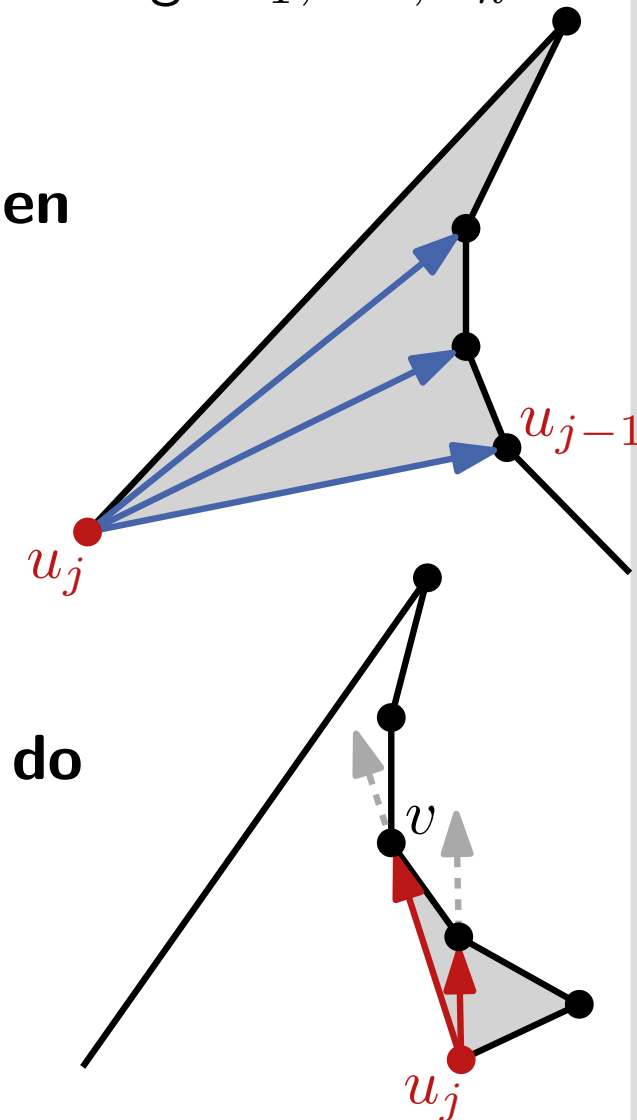
$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

$v \leftarrow S.pop()$

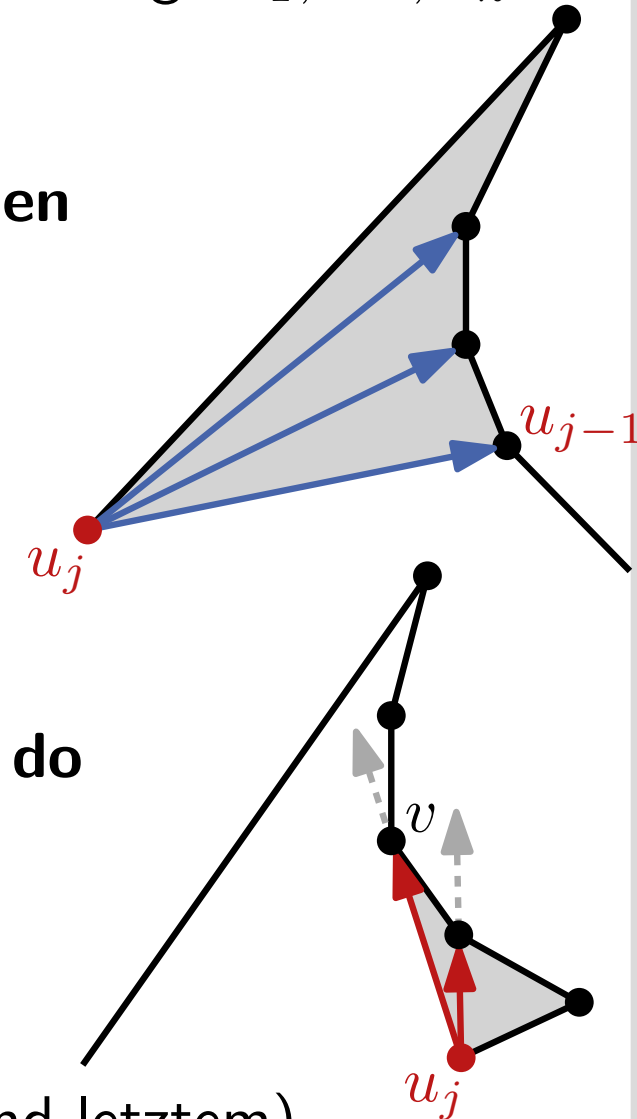
while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$

verbinde u_n zu allen Knoten in S (außer erstem und letztem)



Algorithmus TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P als doppelt-verk. Kantenliste)

verschmelze linken und rechten Pfad \rightarrow absteigende Folge u_1, \dots, u_n

Stack $S \leftarrow \emptyset$; $S.push(u_1)$; $S.push(u_2)$

for $j \leftarrow 3$ **to** $n - 1$ **do**

if u_j und $S.top()$ auf verschiedenen Pfaden **then**

while not $S.empty()$ **do**

$v \leftarrow S.pop()$

if not $S.empty()$ **then** zeichne (u_j, v)

$S.push(u_{j-1})$; $S.push(u_j)$

else

$v \leftarrow S.pop()$

while not $S.empty()$ **and** u_j sieht $S.top()$ **do**

$v \leftarrow S.pop()$

 zeichne Diagonale (u_j, v)

$S.push(v)$; $S.push(u_j)$

verbinde u_n zu allen Knoten in S (außer erstem und letztem)

Aufgabe:

Welche Laufzeit hat der Algorithmus?

Zusammenfassung

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

Zusammenfassung

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

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

(alt)

Zusammenfassung

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

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

(alt)



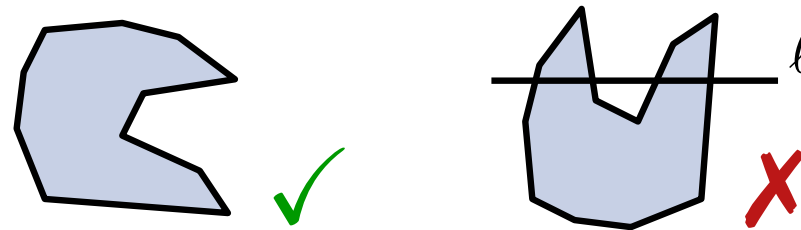
Satz: Ein einfaches Polygon mit n Knoten kann in $O(n \log n)$ Zeit und $O(n)$ Platz trianguliert werden.

Triangulierung: Überblick

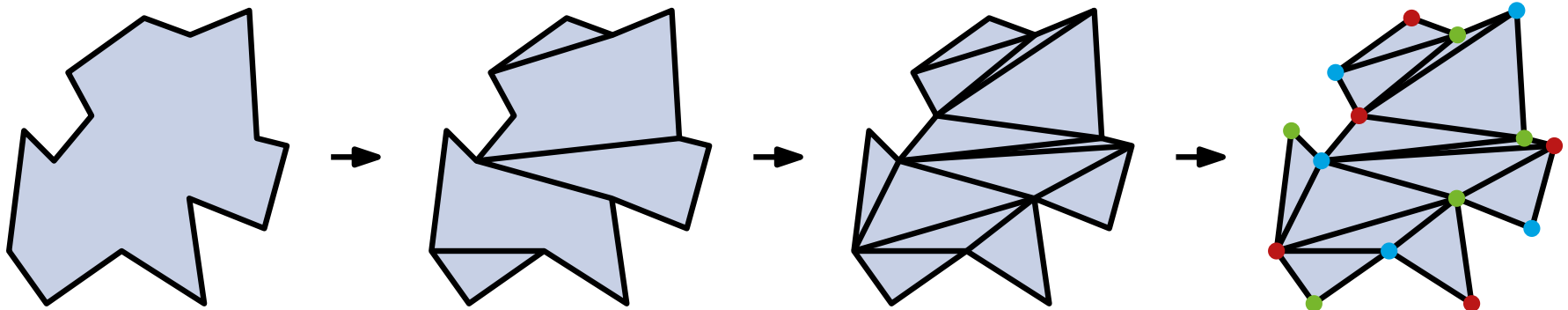
Dreistufiges Verfahren:

- 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 ✓



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

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

Ja, der Algorithmus zum Zerlegen in y -montone Teilpolygone funktioniert weiterhin.

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

Ja, der Algorithmus zum Zerlegen in y -montone Teilpolygone funktioniert weiterhin.

Geht es für allgemeine einfache Polygone noch schneller?

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

Ja, der Algorithmus zum Zerlegen in y -montone Teilpolygone funktioniert weiterhin.

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 vorgestellt wurden, beschrieb Chazelle [1990] einen (komplizierten) deterministischen Linearzeit-Algorithmus.