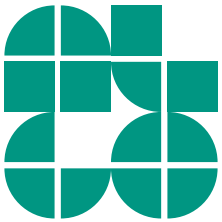
Karlsruhe Institute of Technology

Übung Algorithmische Geometrie
Streckensegmente + Polygontriangulierung

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

Andreas Gemsa

10.05.2012



Ablauf

Karlsruhe Institute of Technology

Übungsblatt 2

Übungsblatt 3

Nachtrag – Konvexe Hülle

Input: $P \subset \mathbb{R}^2$

- Berechne in $O(n)$ die Menge P' alle Punkte der von $CH(P)$
- Berechne die konvexe Hülle von P' in $O(h \log h)$

Gesamtlaufzeit: $O(n) + O(h \log h)$

Für $h = \log n$:

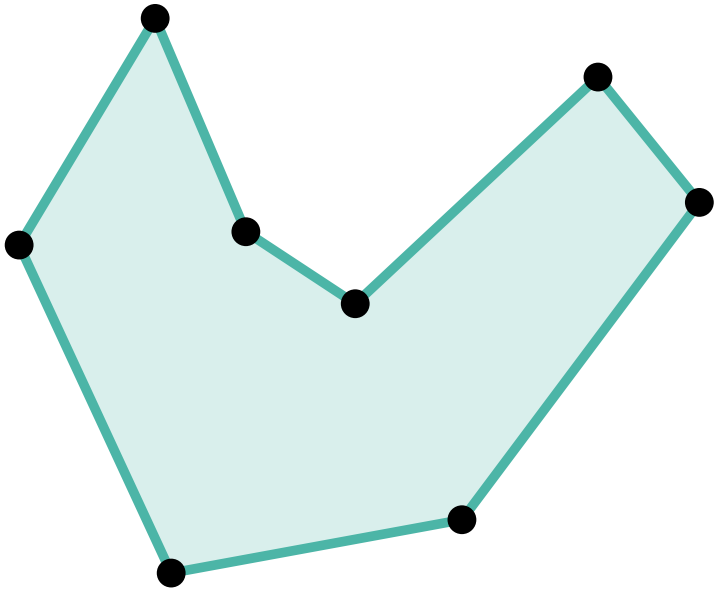
Alg	Chan
$O(n) + O(\log n \cdot \log \log n)$	$O(n \log \log n)$
$= O(n)$	

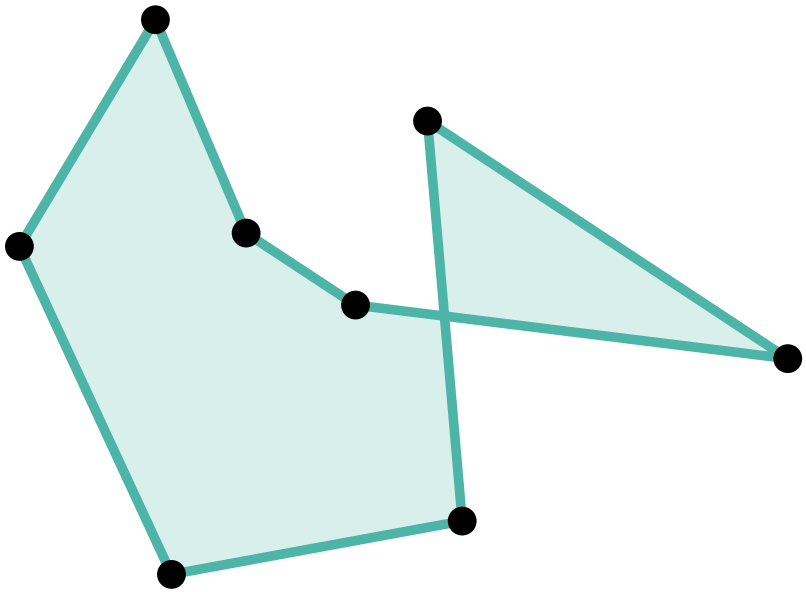
Ist so ein Verfahren denkbar?

Übungsblatt 2 - Aufgabe 1

Gesucht:

Algorithmus der in $\mathcal{O}(n \log n)$ bestimmt ob ein Polygon einfach (d.h. schnittfrei) ist.





Übungsblatt 2 - Aufgabe 2

Karlsruhe Institute of Technology

VL:

Laufzeit: $\mathcal{O}((n + I) \log n)$

Speicherplatz: $\mathcal{O}(n + I)$

Gesucht:

Algorithmus der nur linearen Speicher benötigt.

Übungsblatt 2 - Aufgabe 2

Karlsruhe Institute of Technology

VL:

Laufzeit: $\mathcal{O}((n + I) \log n)$

Speicherplatz: $\mathcal{O}(n + I)$

Gesucht:

Algorithmus der nur linearen Speicher benötigt.

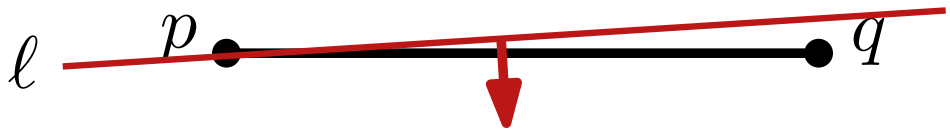
Frage:

Welche Datenstruktur ist problematisch?

Datenstrukturen

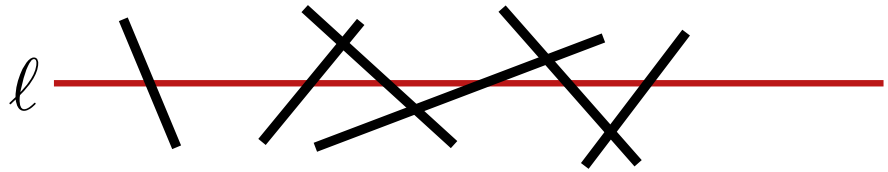
1.) Event Queue $\mathcal{Q}$

- definiere $p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \vee (y_p = y_q \wedge x_p < x_q)$



- speichere Events sortiert nach $\prec$ in **balanciertem binärem Suchbaum**
  $\rightarrow$ AVL-Baum, Rot-Schwarz-Baum, ...
- Operationen `insert`, `delete` und `nextEvent` in $O(\log |\mathcal{Q}|)$ Zeit

2.) Sweep-Line Status $\mathcal{T}$



- speichere von ℓ geschnittene Strecken geordnet von links nach rechts
- benötigte Operationen `insert`, `delete`, `findNeighbor`
- ebenfalls balancierter binärer Suchbaum mit Strecken in den Blättern!

Übungsblatt 2 - Aufgabe 2

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \emptyset$

foreach $s \in S$ **do**

    $\mathcal{Q}$.insert(upperEndPoint(s))

    $\mathcal{Q}$.insert(lowerEndPoint(s))

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

    $p \leftarrow \mathcal{Q}$.nextEvent()

    $\mathcal{Q}$.deleteEvent(p)

    handleEvent(p)

Übungsblatt 2 - Aufgabe 2

handleEvent(p)

$U(p) \leftarrow$ Strecken mit p oberer Endpunkt
$L(p) \leftarrow$ Strecken mit p unterer Endpunkt
$C(p) \leftarrow$ Strecken mit p innerer Punkt
if $|U(p) \cup L(p) \cup C(p)| \geq 2$ **then**
  gebe p und $U(p) \cup L(p) \cup C(p)$ aus
entferne $L(p) \cup C(p)$ aus $\mathcal{T}$
füge $U(p) \cup C(p)$ in $\mathcal{T}$ ein
if $U(p) \cup C(p) = \emptyset$ **then** //s_l und s_r Nachbarn von p in $\mathcal{T}$
  $\mathcal{Q} \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p
else //s' und s'' linkeste und rechteste Strecke in $U(p) \cup C(p)$
  $\mathcal{Q} \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p
  $\mathcal{Q} \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

Übungsblatt 2 - Aufgabe 2

Karlsruhe Institute of Technology

VL:

Laufzeit: $\mathcal{O}((n + I) \log n)$

Speicherplatz: $\mathcal{O}(n + I)$

Gesucht:

Algorithmus der nur linearen Speicher benötigt.

Frage:

Welche Datenstruktur ist problematisch?

Übungsblatt 2 - Aufgabe 2

Karlsruhe Institute of Technology

VL:

Laufzeit: $\mathcal{O}((n+I)\log n)$

Speicherplatz: $\mathcal{O}(n+I)$

Gesucht:

Algorithmus der nur linearen Speicher benötigt.

Frage:

Welche Datenstruktur ist problematisch?

Maximale Größe von $\mathcal{Q}$: $\mathcal{O}(n\log^2 n)$

[Pach, Sharir, "On Vertical Visibility in Arrangments of Segments and the Queue Size in the Bentley-Ottmann Line Sweeping Algorithm", (1991)]

Übungsblatt 2 - Aufgabe 3

Karlsruhe Institute of Technology

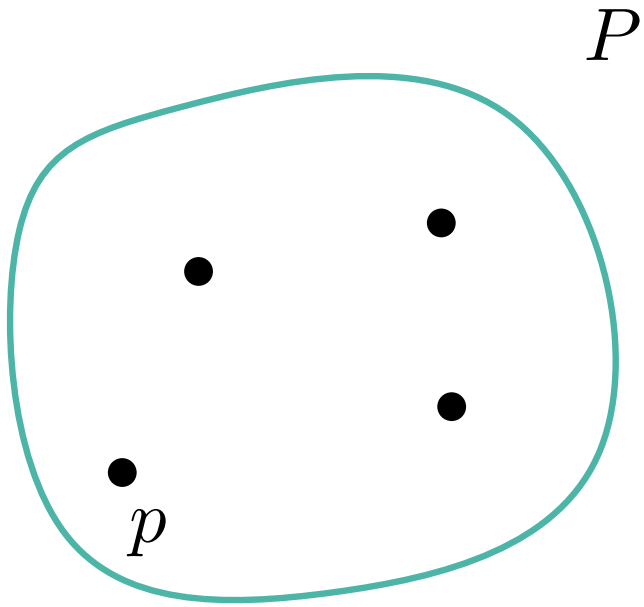
Gegeben:
Endliche Punktmenge P

Def.:
größter rechter obere Bereich (groB) von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.

Übungsblatt 2 - Aufgabe 3

Gegeben:
Endliche Punktmenge P

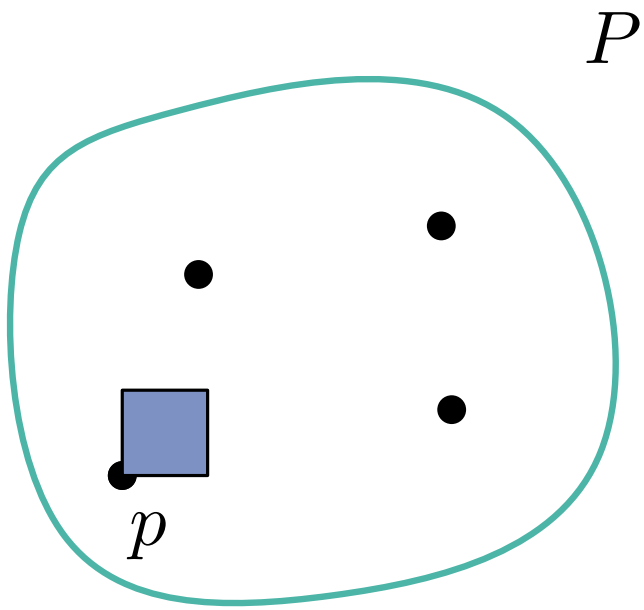
Def.:
größter rechter obere Bereich (groB) von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.



Übungsblatt 2 - Aufgabe 3

Gegeben:
Endliche Punktmenge P

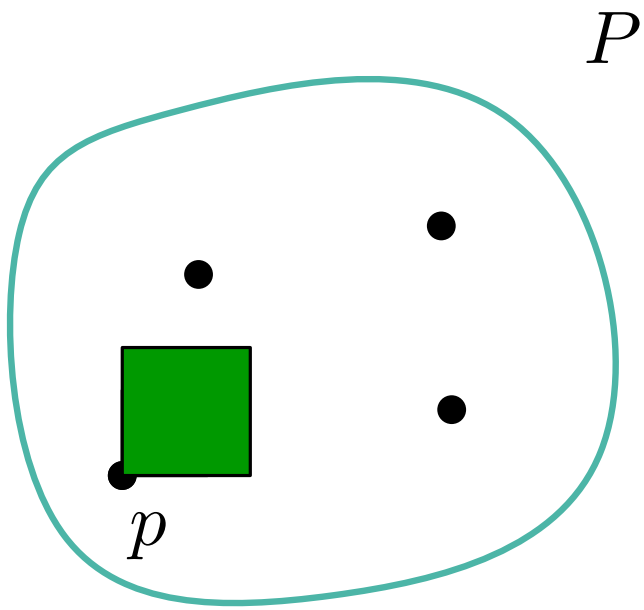
Def.:
größter rechter obere Bereich (groB) von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.



Übungsblatt 2 - Aufgabe 3

Gegeben:
Endliche Punktmenge P

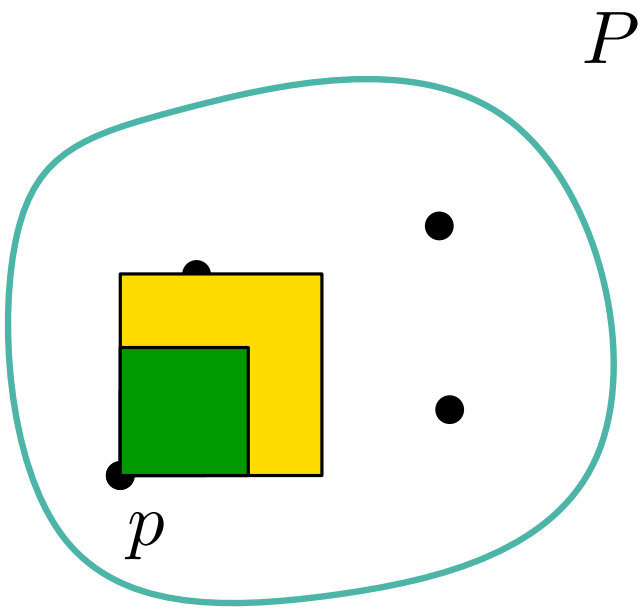
Def.:
größter rechter obere Bereich (groB) von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.



Übungsblatt 2 - Aufgabe 3

Gegeben:
Endliche Punktmenge P

Def.:
größter rechter obere Bereich (groB) von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.

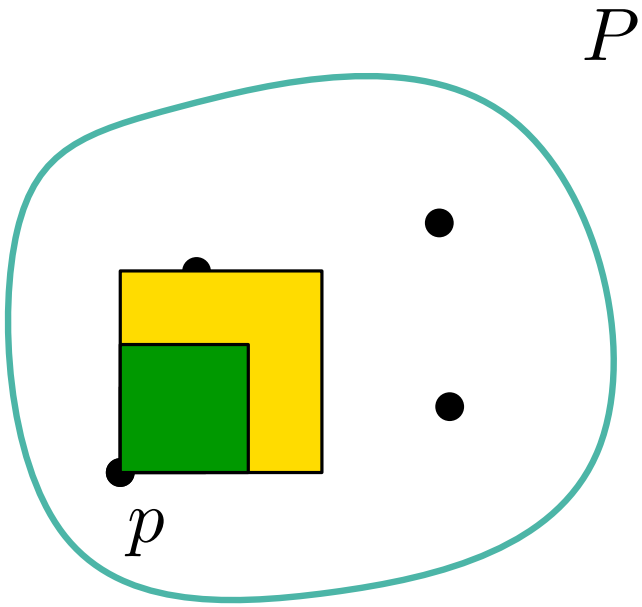


Übungsblatt 2 - Aufgabe 3

Gegeben:
Endliche Punktmenge P

Def.:
größter rechter obere Bereich (groB) von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.

a) Zeige, dass *groB* entweder Quadrat oder Schnitt zweier offener Halbebenen



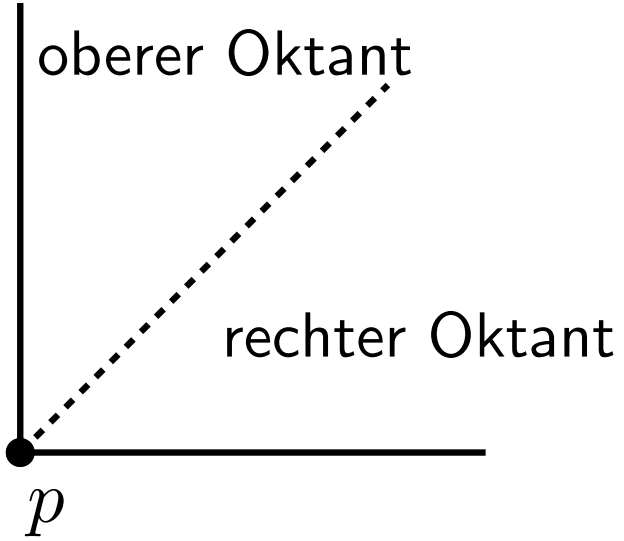
Übungsblatt 2 - Aufgabe 3

Gegeben:
Endliche Punktmenge P

Def.:
größter rechter obere Bereich (groB) von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.

a) Zeige, dass *groB* entweder Quadrat oder Schnitt zweier offener Halbebenen

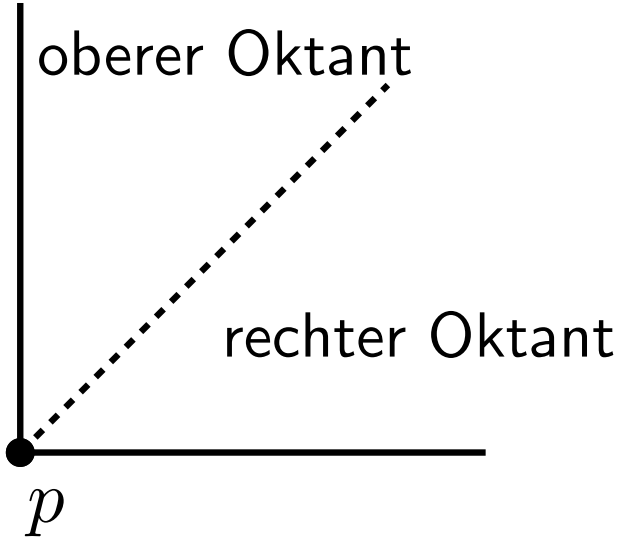
b) Welche Punkte im rechten und im oberen Oktanten schränken den *groB* am stärksten ein?



Übungsblatt 2 - Aufgabe 3

Gegeben:
Endliche Punktmenge P

Def.:
größter rechter obere Bereich (groB) von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.



a) Zeige, dass *groB* entweder Quadrat oder Schnitt zweier offener Halbebenen

b) Welche Punkte im rechten und im oberen Oktanten schränken den *groB* am stärksten ein?

c) Algorithmus der für alle Punkte in P den *groB* in $\mathcal{O}(n \log n)$ berechnet.

Übungsblatt 2 - Aufgabe 3

c) Algorithmus der für alle Punkte in P den *groB* in $\mathcal{O}(n \log n)$ berechnet.

Vorschlag:

Sweep(P)

Input: Punktmenge P

Output: groB für alle Punkte aus P

ysort = sort_y(P) //aufsteigend

xsort = sort_x(P) //aufsteigend

foreach $p \in P$ **do**

    next_x = erster Knoten rechts von p

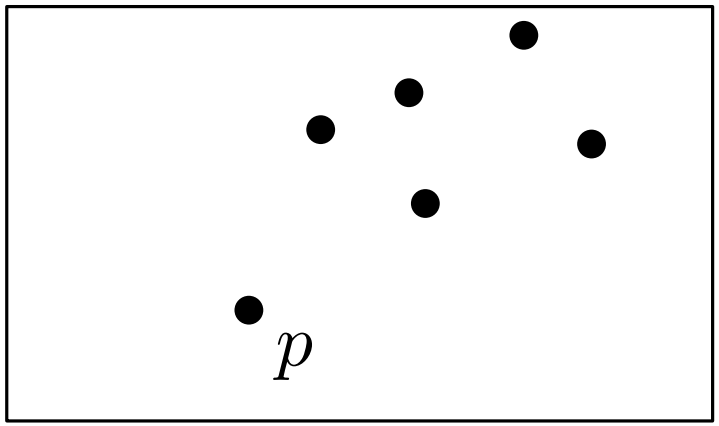
    next_y = erster Knoten oberhalb von p

    groB[p] = $\min\{x(\text{next\_x}) - x(p), y(\text{next\_y}) - y(p)\}$

return groB

Übungsblatt 2 - Aufgabe 3

c) Algorithmus der für alle Punkte in P den *groB* in $\mathcal{O}(n \log n)$ berechnet.



Vorschlag:

Sweep(P)

Input: Punktmenge P

Output: groB für alle Punkte aus P

ysort = sort_y(P) //aufsteigend

xsort = sort_x(P) //aufsteigend

foreach $p \in P$ **do**

  next_x = erster Knoten rechts von p

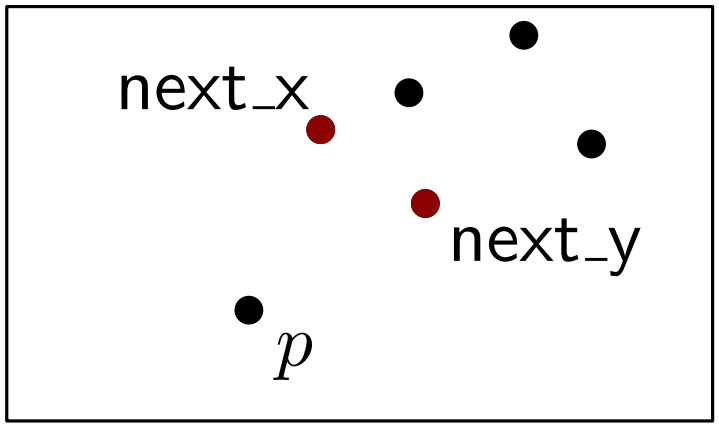
  next_y = erster Knoten oberhalb von p

  groB[p] = $\min\{x(\text{next\_x}) - x(p), y(\text{next\_y}) - y(p)\}$

return groB

Übungsblatt 2 - Aufgabe 3

c) Algorithmus der für alle Punkte in P den *groB* in $\mathcal{O}(n \log n)$ berechnet.



Vorschlag:

Sweep(P)

Input: Punktmenge P

Output: groB für alle Punkte aus P

ysort = sort_y(P) //aufsteigend

xsort = sort_x(P) //aufsteigend

foreach $p \in P$ **do**

    next_x = erster Knoten rechts von p

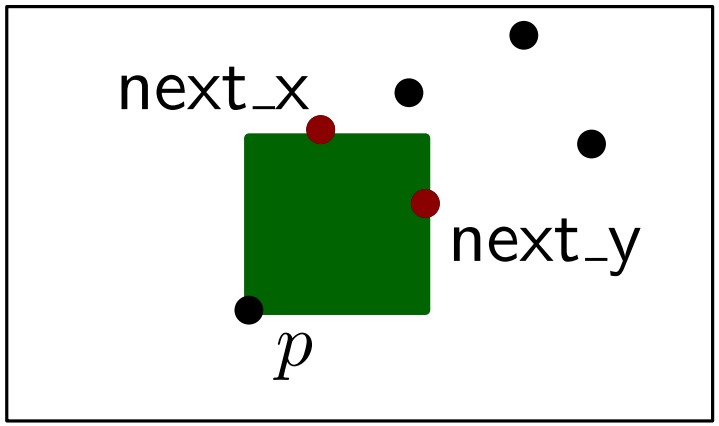
    next_y = erster Knoten oberhalb von p

    groB[p] = $\min\{x(\text{next\_x}) - x(p), y(\text{next\_y}) - y(p)\}$

return groB

Übungsblatt 2 - Aufgabe 3

c) Algorithmus der für alle Punkte in P den *groB* in $\mathcal{O}(n \log n)$ berechnet.



Vorschlag:

Sweep(P)

Input: Punktmenge P

Output: groB für alle Punkte aus P

ysort = sort_y(P) //aufsteigend

xsort = sort_x(P) //aufsteigend

foreach $p \in P$ **do**

    next_x = erster Knoten rechts von p

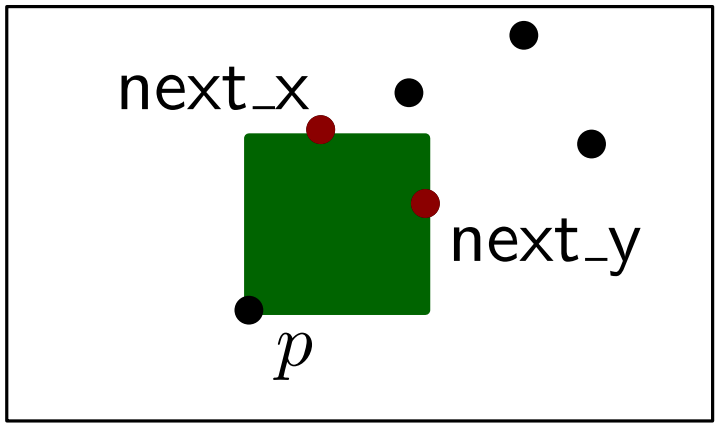
    next_y = erster Knoten oberhalb von p

    groB[p] = $\min\{x(\text{next\_x}) - x(p), y(\text{next\_y}) - y(p)\}$

return groB

Übungsblatt 2 - Aufgabe 3

c) Algorithmus der für alle Punkte in P den *groB* in $\mathcal{O}(n \log n)$ berechnet.



Vorschlag:

Sweep(P)

Input: Punktmenge P

Output: groB für alle Punkte aus P

ysort = sort_y(P) //aufsteigend — $\mathcal{O}(n \log n)$

xsort = sort_x(P) //aufsteigend

foreach $p \in P$ **do** — $\mathcal{O}(n)$

  next_x = erster Knoten rechts von p — $\mathcal{O}(1)$

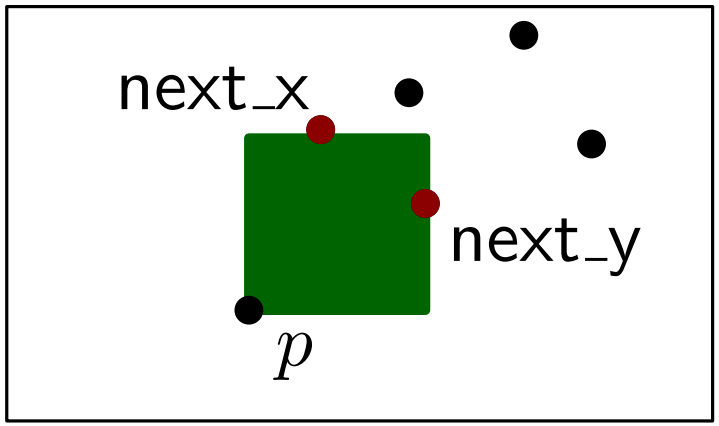
  next_y = erster Knoten oberhalb von p

  groB[p] = $\min\{x(\text{next\_x}) - x(p), y(\text{next\_y}) - y(p)\}$

return groB

Übungsblatt 2 - Aufgabe 3

c) Algorithmus der für alle Punkte in P den *groB* in $\mathcal{O}(n \log n)$ berechnet.



Vorschlag:

Sweep(P)

Input: Punktmenge P

Output: groB für alle Punkte aus P

ysort = sort_y(P) //aufsteigend — $\mathcal{O}(n \log n)$

xsort = sort_x(P) //aufsteigend

foreach $p \in P$ **do** — $\mathcal{O}(n)$

  next_x = erster Knoten rechts von p — $\mathcal{O}(1)$

  next_y = erster Knoten oberhalb von p

  groB[p] = $\min\{x(\text{next\_x}) - x(p), y(\text{next\_y}) - y(p)\}$

return groB

Korrekt?

Ablauf

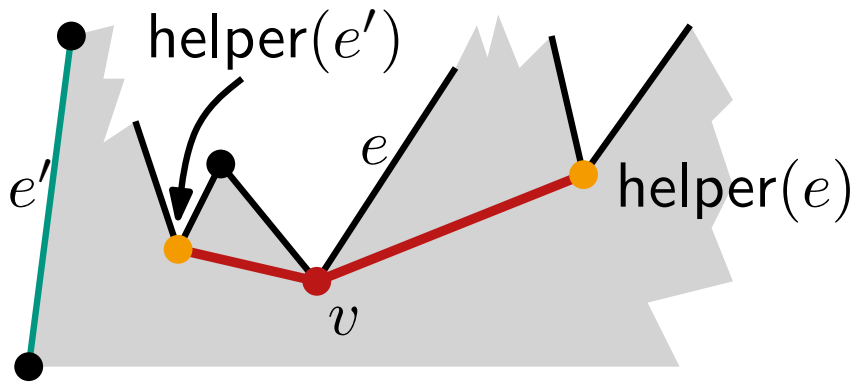
Karlsruhe Institute of Technology

Übungsblatt 2

Übungsblatt 3

Aufgabe 1

Beweise die Korrektheit von **handleMergeVertex**.



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') $\leftarrow v$

Aufgabe 2

VL: Algorithmus zur Partitionierung eines Polygons in y-monotone Teilpolygone in $\mathcal{O}(n \log n)$.

Angenommen die Anzahl der Wendeknoten ist in $O(1)$.
Reduziere die Laufzeit zu $\mathcal{O}(n)$.

Aufgabe 2

VL: Algorithmus zur Partitionierung eines Polygons in y-monotone Teilpolygone in $\mathcal{O}(n \log n)$.

Angenommen die Anzahl der Wendeknoten ist in $O(1)$.
Reduziere die Laufzeit zu $\mathcal{O}(n)$.

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 $\mathcal{D}$

Algorithmus MakeMonotone(P)

Karlsruhe Institute of Technology

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 $\mathcal{D}$

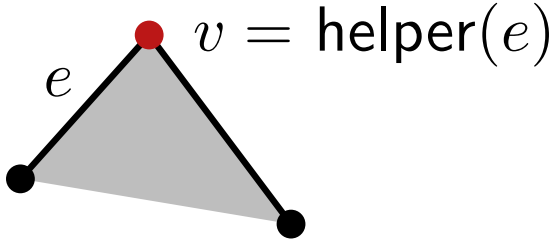
Algorithmus MakeMonotone(P)

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)

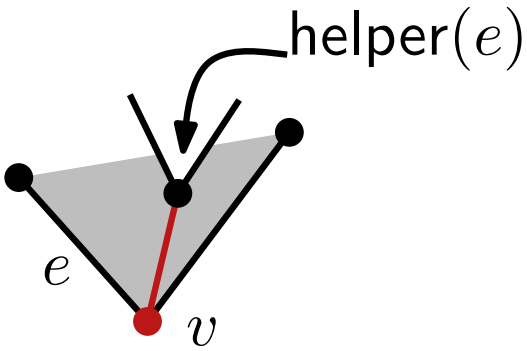
$\mathcal{T} \leftarrow$ füge linke Kante e ein
helper(e) $\leftarrow v$



Algorithmus MakeMonotone(P)

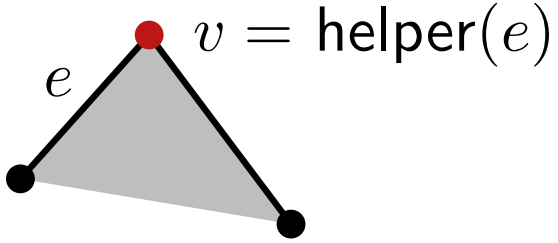
MakeMonotone(Polygon P)

$\mathcal{D} \leftarrow$ 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 $\mathcal{T}$

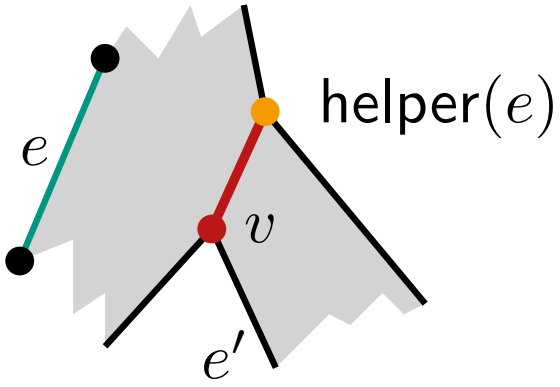
Algorithmus MakeMonotone(P)

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)

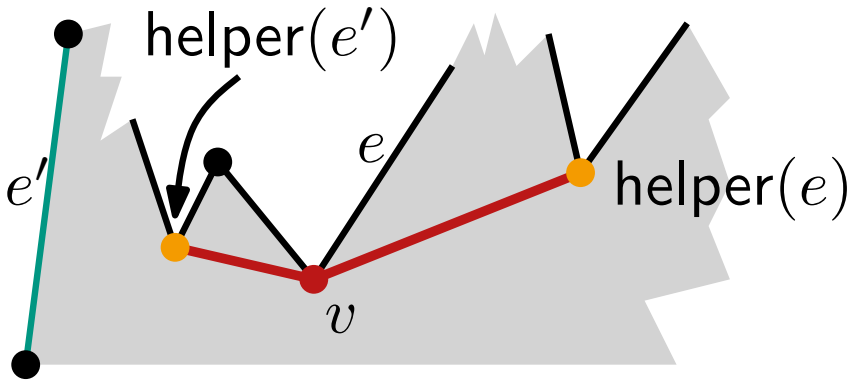
$e \leftarrow$ Kante links von v in $\mathcal{T}$
$\mathcal{D} \leftarrow$ füge (helper(e), v) ein
helper(e) $\leftarrow v$
$\mathcal{T} \leftarrow$ füge rechte Kante e' von v ein
helper(e') $\leftarrow v$



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 $\mathcal{D}$



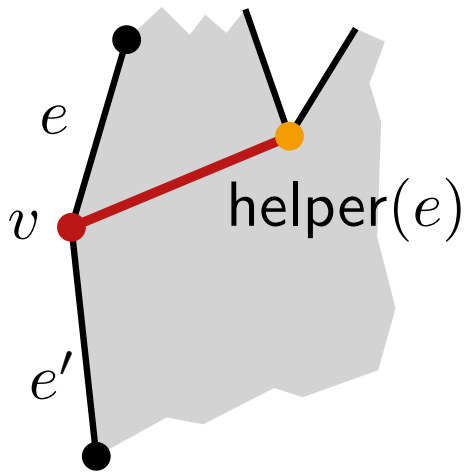
handleMergeVertex(vertex v)

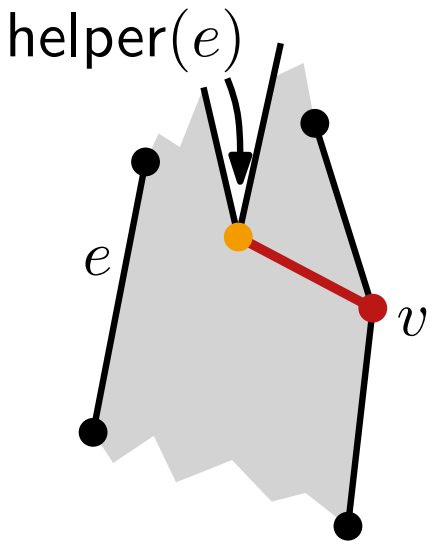
$e \leftarrow$ rechte Kante
if isMergeVertex(helper(e)) **then**
  $\mathcal{D} \leftarrow$ füge (helper(e), v) ein
lösche e aus $\mathcal{T}$
$e' \leftarrow$ Kante links von v in $\mathcal{T}$
if isMergeVertex(helper(e')) **then**
  $\mathcal{D} \leftarrow$ füge (helper(e'), v) ein
helper(e') $\leftarrow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

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)

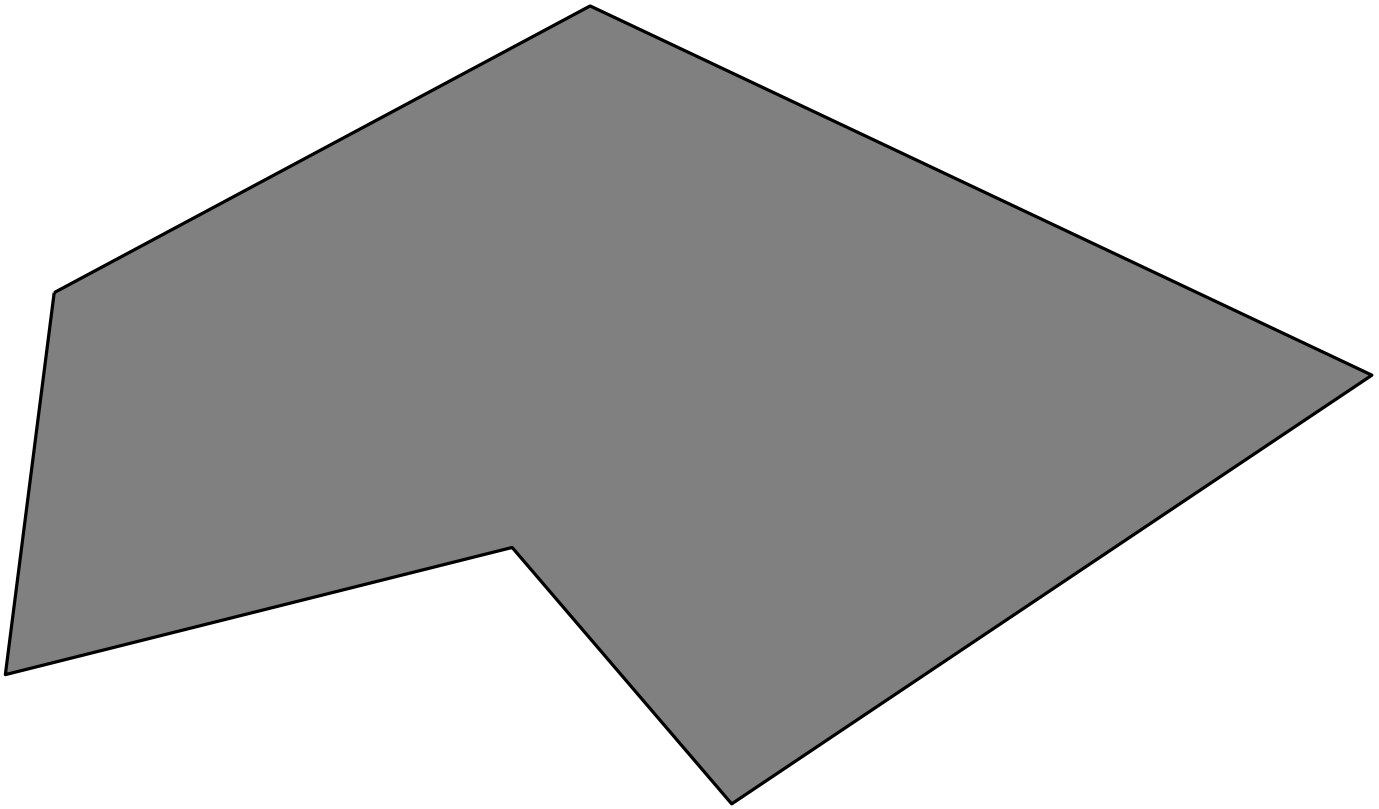
if P liegt lokal rechts von v **then**
  $e, e' \leftarrow$ obere, untere Kante
  if isMergeVertex(helper(e)) **then**
    $\mathcal{D} \leftarrow$ füge (helper(e), v) ein
  lösche e aus $\mathcal{T}$
  $\mathcal{T} \leftarrow$ füge e' ein; helper(e') $\leftarrow v$
else
  $e \leftarrow$ Kante links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

Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?

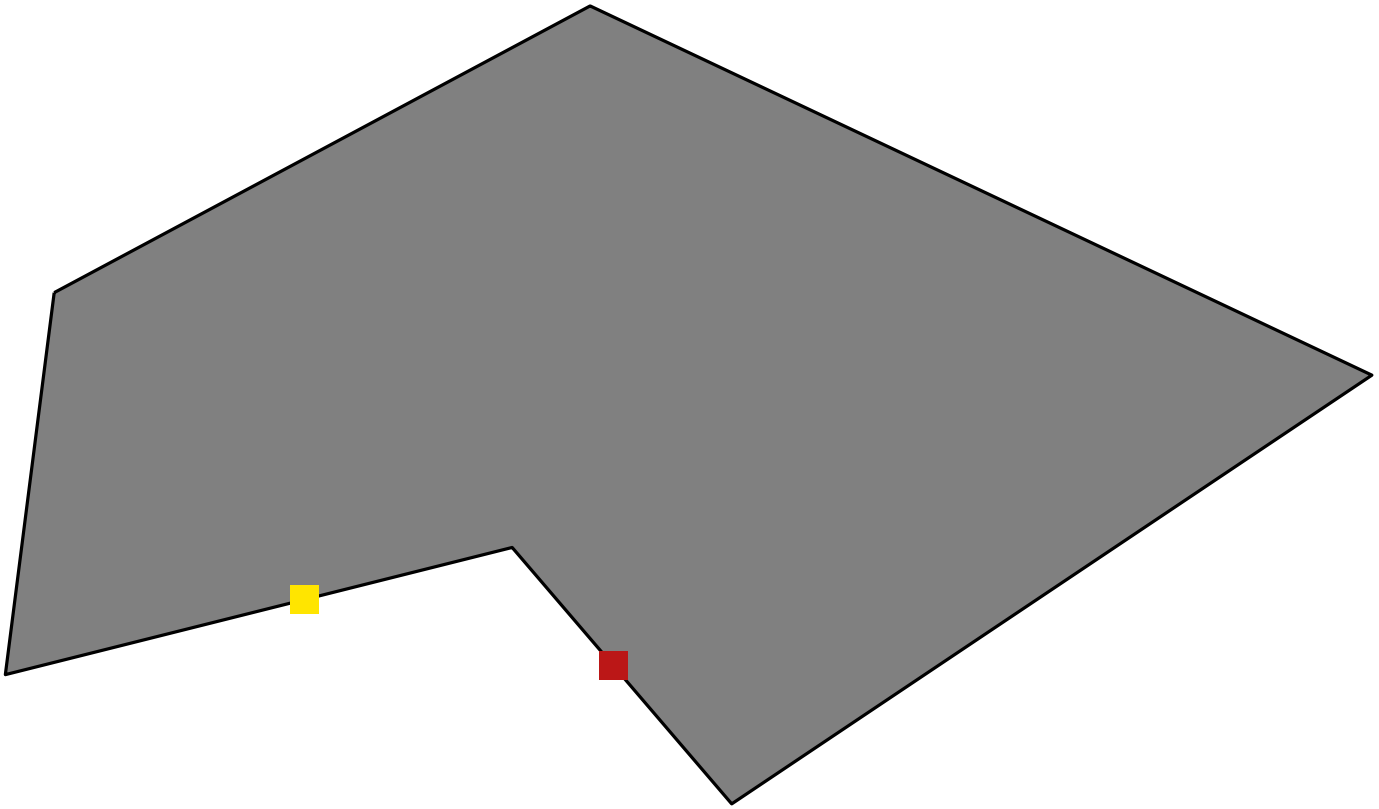
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



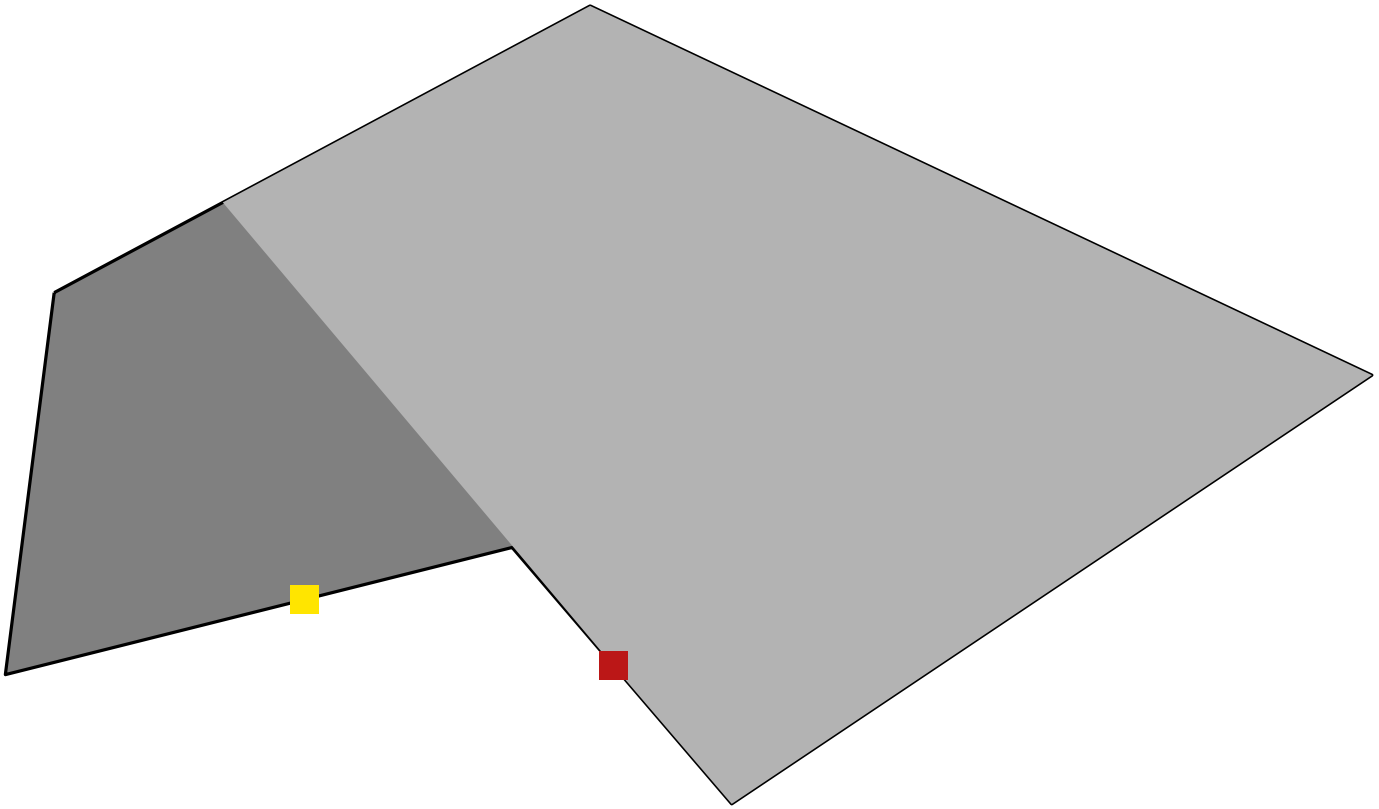
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



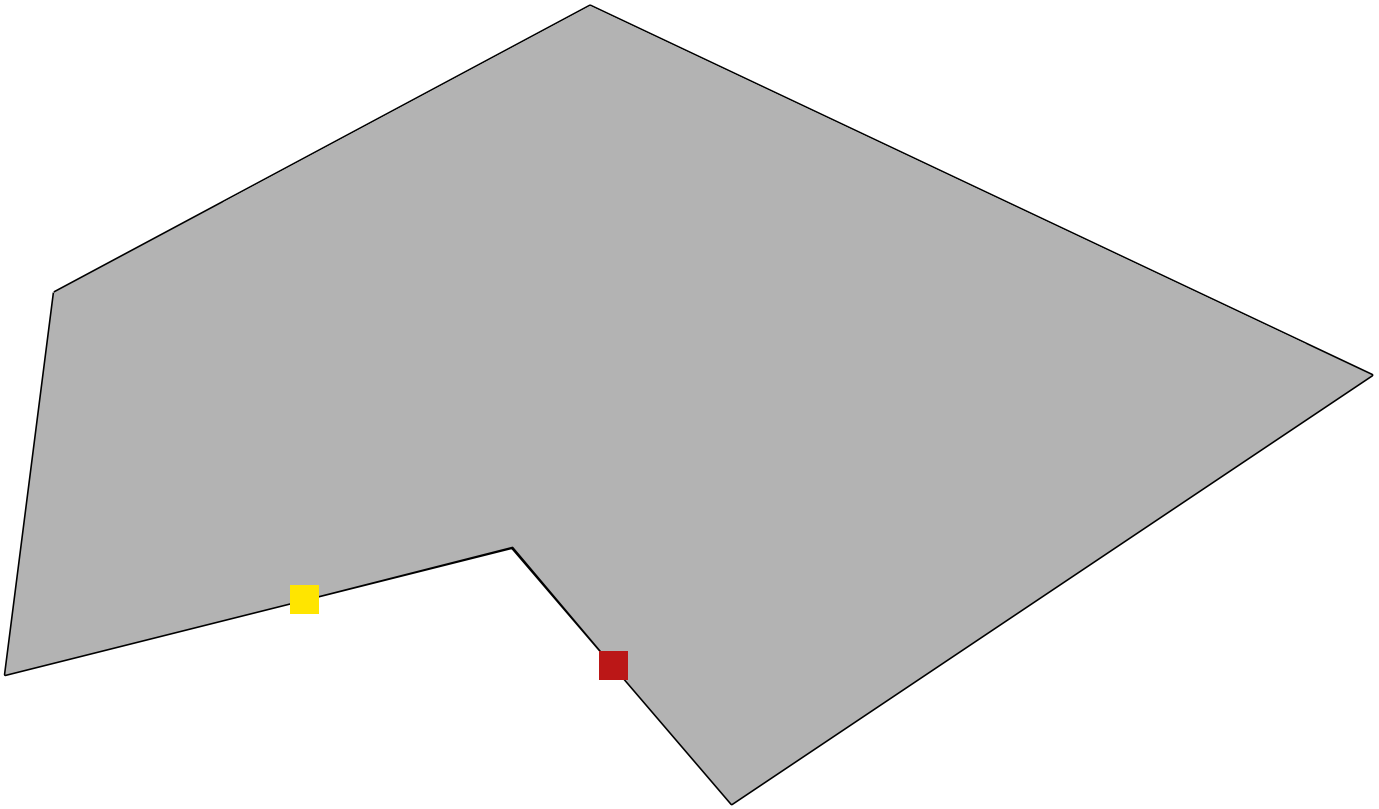
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



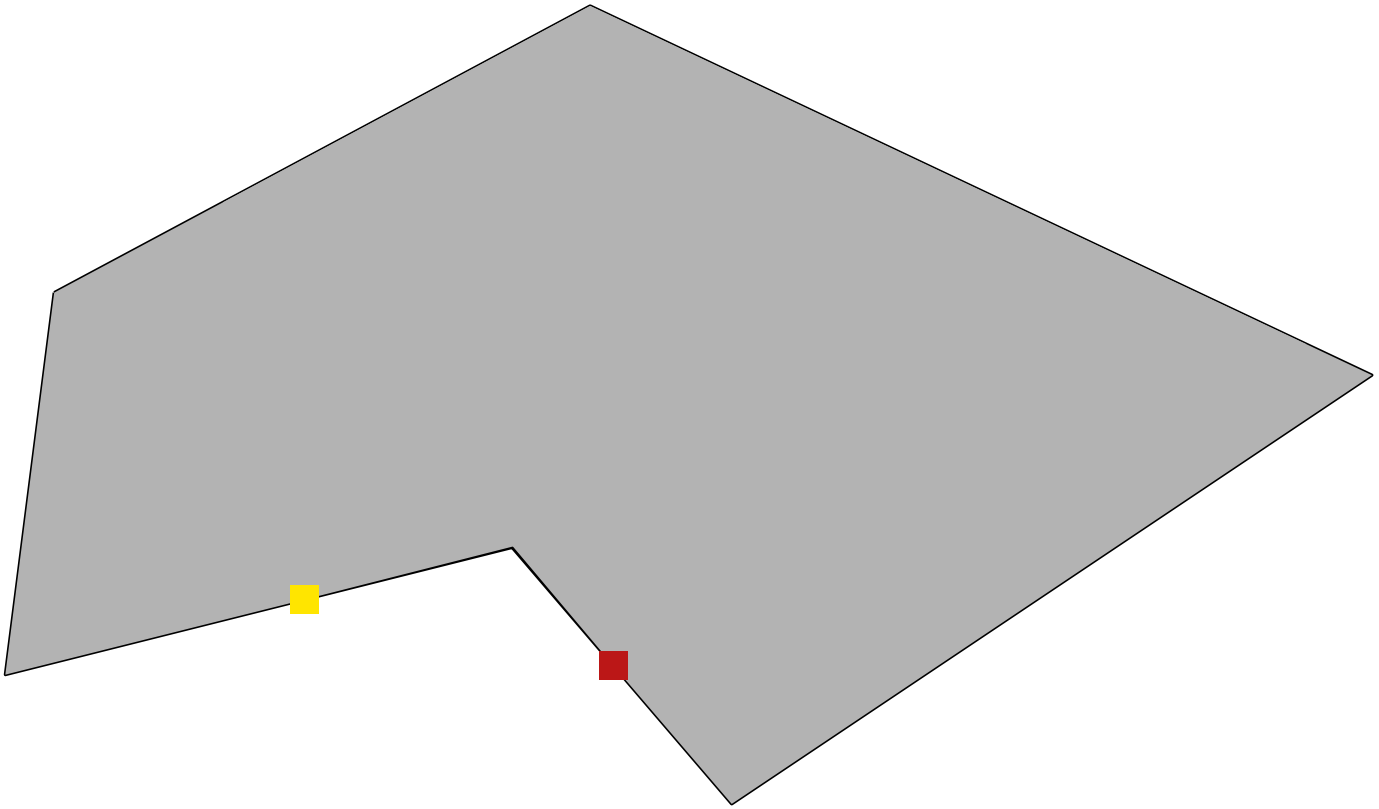
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



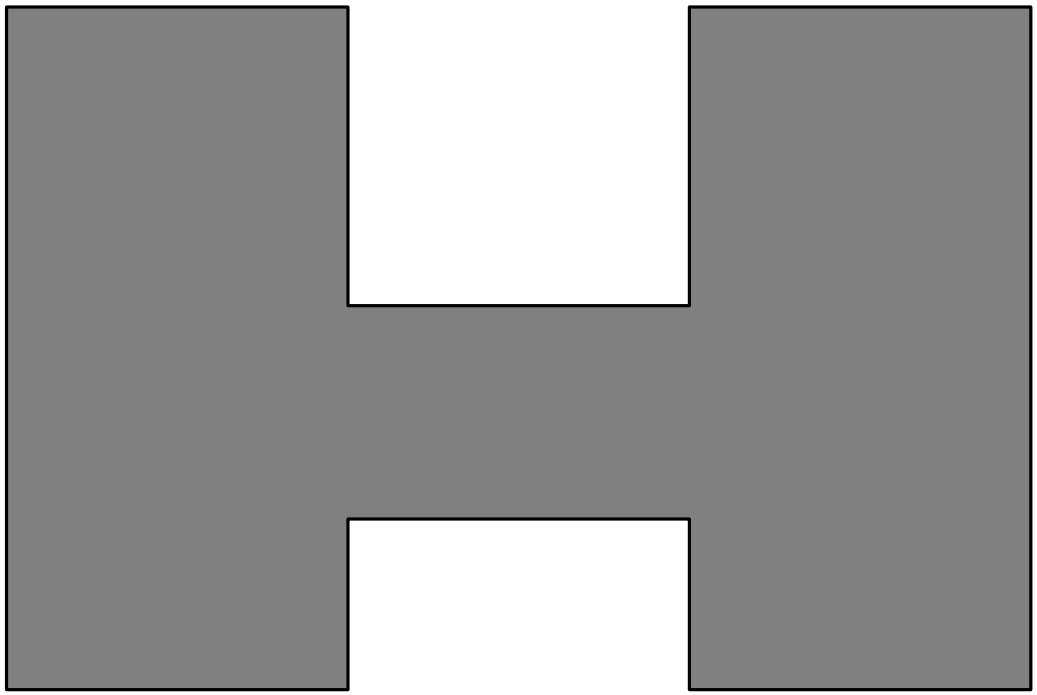
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



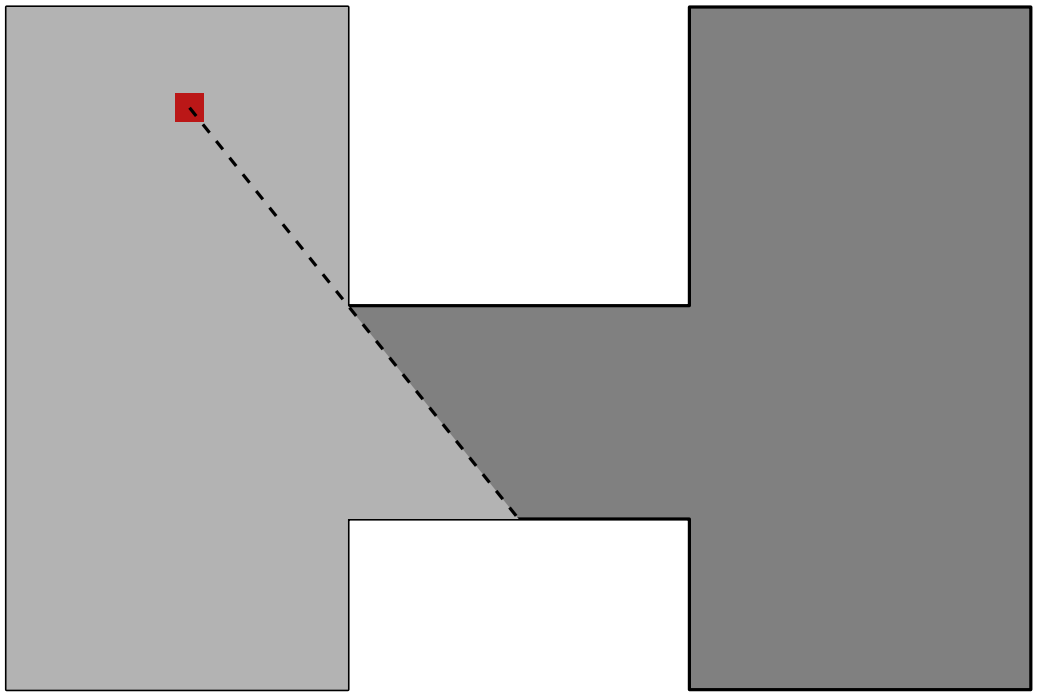
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



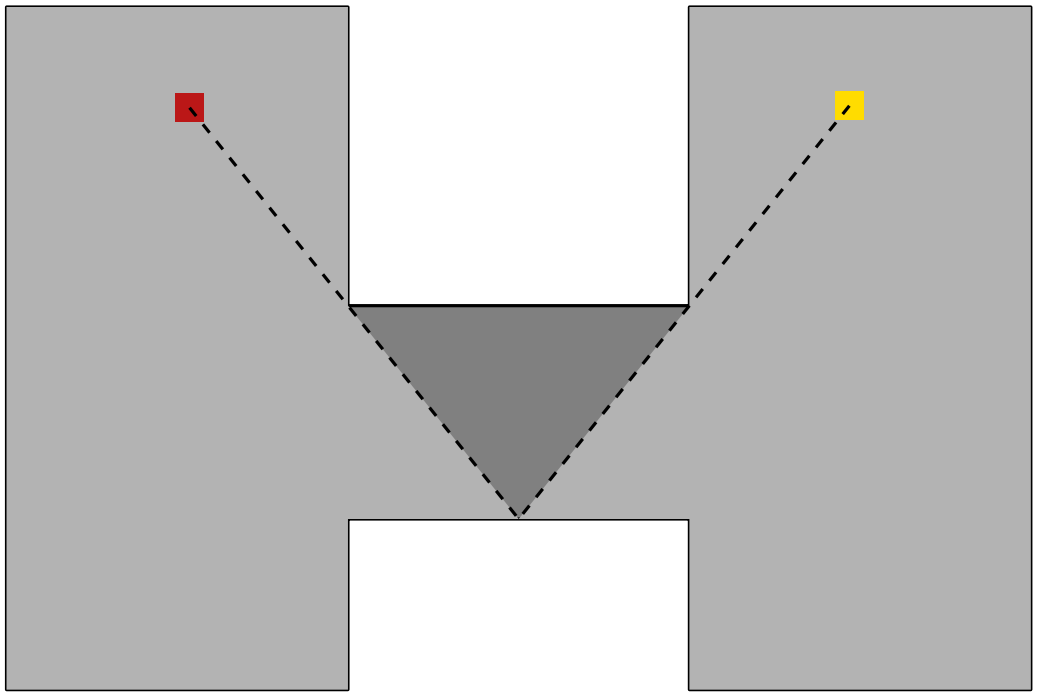
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



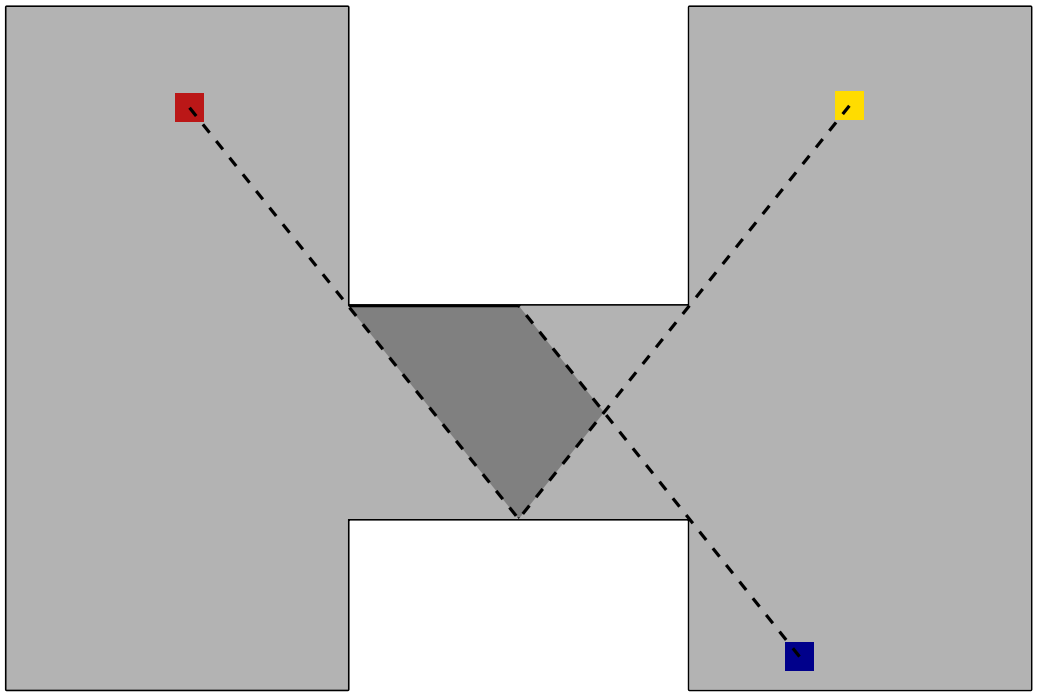
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



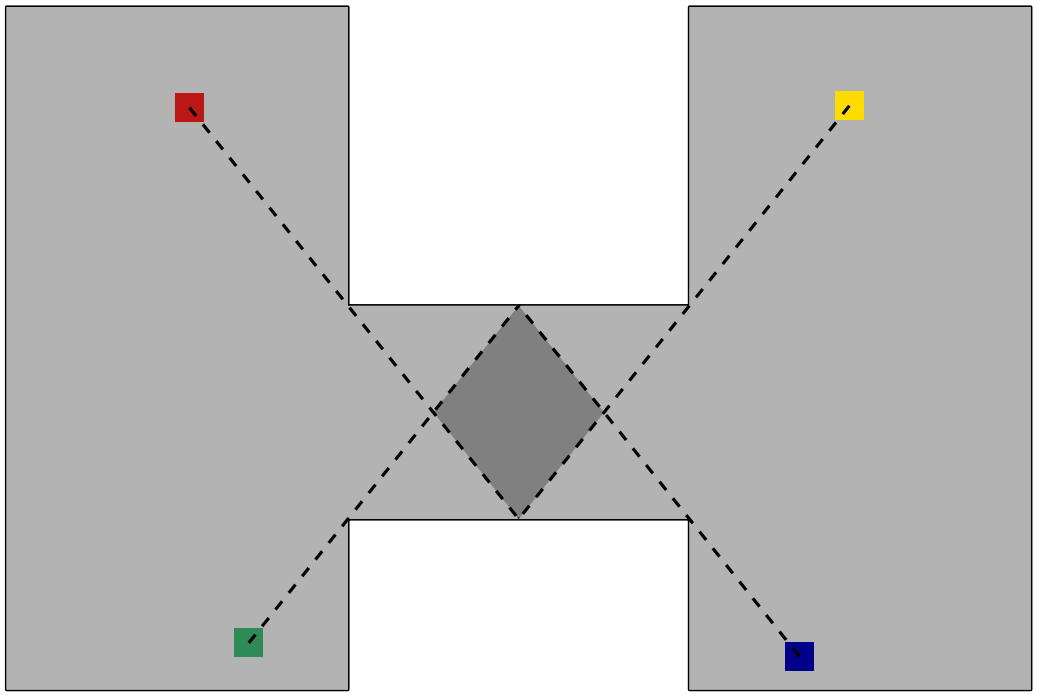
Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



Aufgabe 3

$\mathcal{P}$ sei Polygon.
Kameras sind so platziert, dass der vollständige Rand von $\mathcal{P}$ abgedeckt wird. Wird automatisch auch das Innere vollständig abgedeckt?



Aufgabe 4

Gebe einen $\mathcal{O}(n \log n)$ Algorithmus an der ein einfaches Polygon aus n Knoten in zwei einfache Polygone auftrennt. Jedes der beiden soll aus höchstens $\lfloor 2n/3 \rfloor + 2$ Knoten bestehen.

Aufgabe 4

Gebe einen $\mathcal{O}(n \log n)$ Algorithmus an der ein einfaches Polygon aus n Knoten in zwei einfache Polygone auftrennt. Jedes der beiden soll aus höchstens $\lfloor 2n/3 \rfloor + 2$ Knoten bestehen.

- Trianguliere das Polygon
- Berechne Dualgraph zum Polygon
- Bestimme alle Blätter im Dualgraphen (Dualgraph ist ein Baum)

Aufgabe 4

Gebe einen $\mathcal{O}(n \log n)$ Algorithmus an der ein einfaches Polygon aus n Knoten in zwei einfache Polygone auftrennt. Jedes der beiden soll aus höchstens $\lfloor 2n/3 \rfloor + 2$ Knoten bestehen.

- Trianguliere das Polygon
- Berechne Dualgraph zum Polygon
- Bestimme alle Blätter im Dualgraphen (Dualgraph ist ein Baum)

Das war's!

Karlsruhe Institute of Technology

Nächster Übung:

Donnertag, 17.05, 10:15 Uhr

Raum 131, Gebäude 50.34