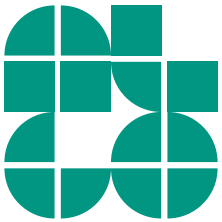
Karlsruhe Institute of Technology

# Übung Algorithmische Geometrie
## Streckensegmente

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

Andreas Gemsa
28.04.2011

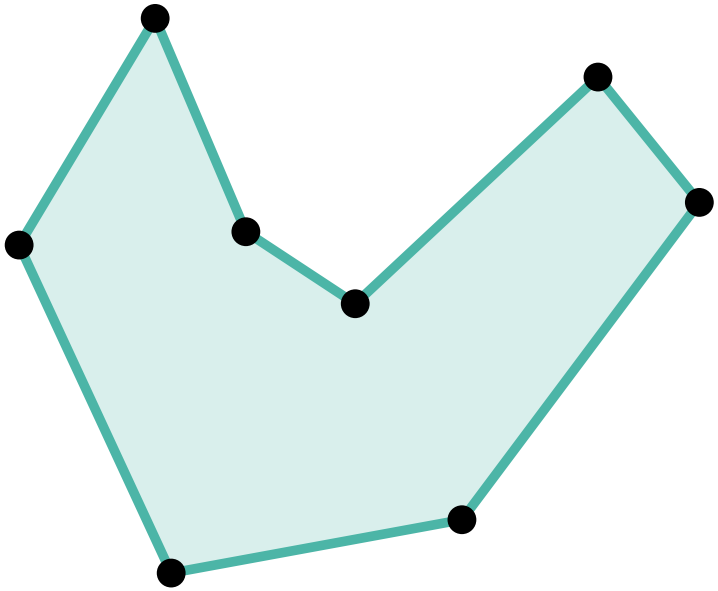


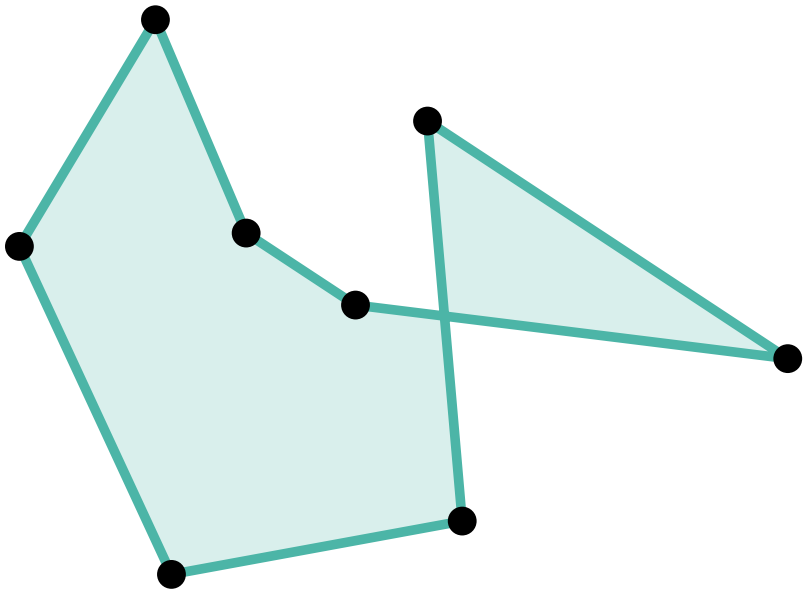
# Ablauf

- Besprechung ÜB2
  - Einfaches Polygon
  - Weniger Speicherplatz
  - Sweepen
- Doppelt-verkettete Kantenlisten

# Übungsblatt 2 - Aufgabe 1

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





# Übungsblatt 2 - Aufgabe 2

Algorithmus aus der Vorlesung:
Laufzeit: $\mathcal{O}((n+I)\log n)$
Speicherplatz: $\mathcal{O}(n+I)$

Reduziere den Speicherplatzverbrauch auf $\mathcal{O}(n)$.

# Übungsblatt 2 - Aufgabe 2

Karlsruhe Institute of Technology

Algorithmus aus der Vorlesung:
Laufzeit: $\mathcal{O}((n + I) \log n)$
Speicherplatz: $\mathcal{O}(n + I)$

Reduziere den Speicherplatzverbrauch auf $\mathcal{O}(n)$.

Welche Datenstruktur ist problematisch?

# Übungsblatt 2 - Aufgabe 2

Karlsruhe Institute of Technology

Algorithmus aus der Vorlesung:
Laufzeit: $\mathcal{O}((n + I) \log n)$
Speicherplatz: $\mathcal{O}(n + I)$

Reduziere den Speicherplatzverbrauch auf $\mathcal{O}(n)$.

Welche Datenstruktur ist problematisch?

# Übungsblatt 2 - Aufgabe 2

Algorithmus aus der Vorlesung:
Laufzeit: $\mathcal{O}((n + I) \log n)$
Speicherplatz: $\mathcal{O}(n + I)$

Reduziere den Speicherplatzverbrauch auf $\mathcal{O}(n)$.

Welche Datenstruktur ist problematisch?

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

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

# 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 3

Karlsruhe Institute of Technology

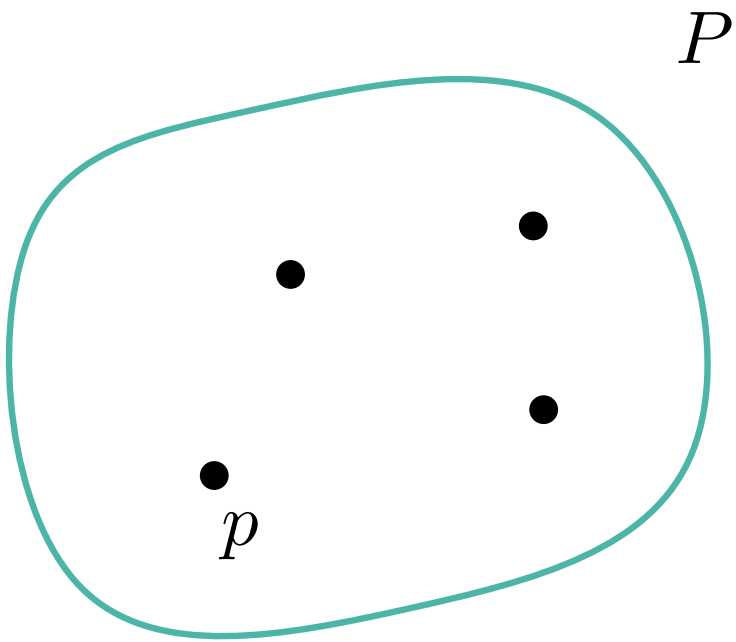
Gegeben: Endliche Punktmenge $P$

*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 $P$

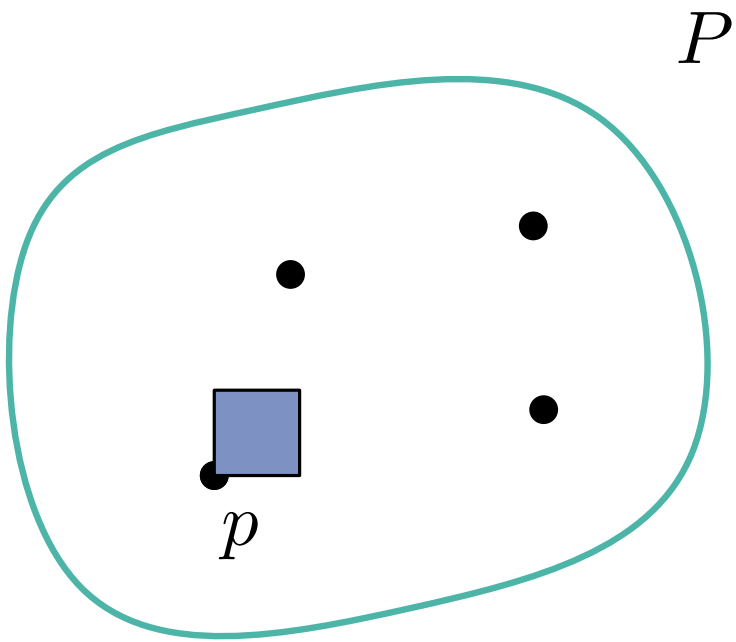
*größter rechter obere Bereich (groB)* von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.



# Übungsblatt 2 - Aufgabe 3

Gegeben: Endliche Punktmenge $P$

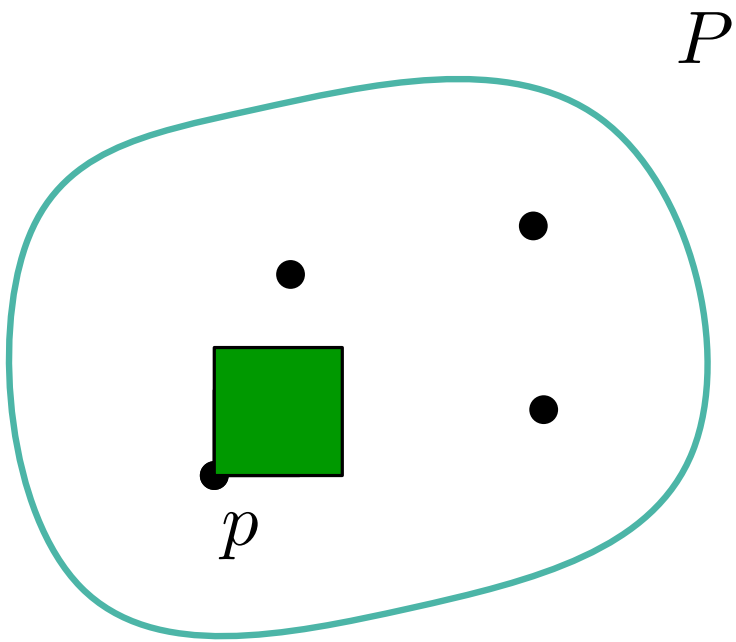
*größter rechter obere Bereich (groB)* von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.



# Übungsblatt 2 - Aufgabe 3

Gegeben: Endliche Punktmenge $P$

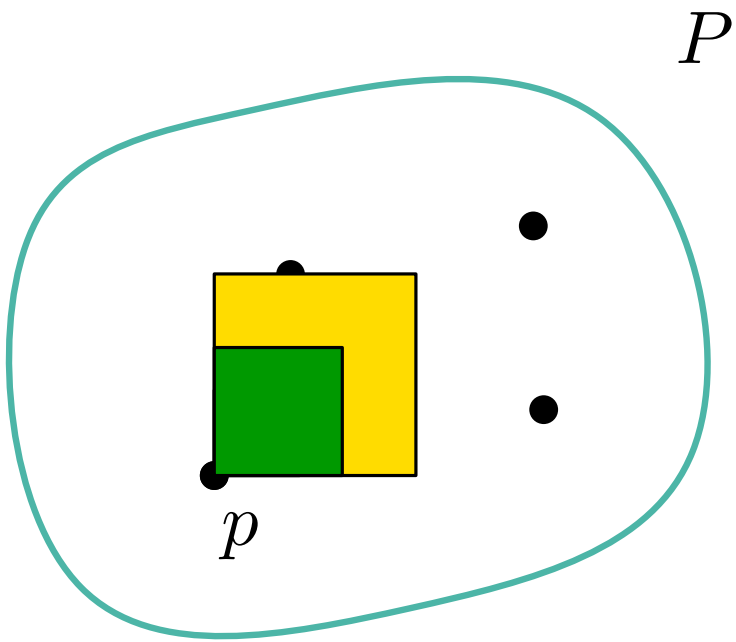
*größter rechter obere Bereich (groB)* von $p \in P$: Vereinigung aller offenen achsenparallelen Quadrate, die p mit ihrer linken unteren Ecke ber ühren und keinen Punkt aus P in ihrem Inneren enthalten.



# Übungsblatt 2 - Aufgabe 3

Gegeben: Endliche Punktmenge $P$

*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.

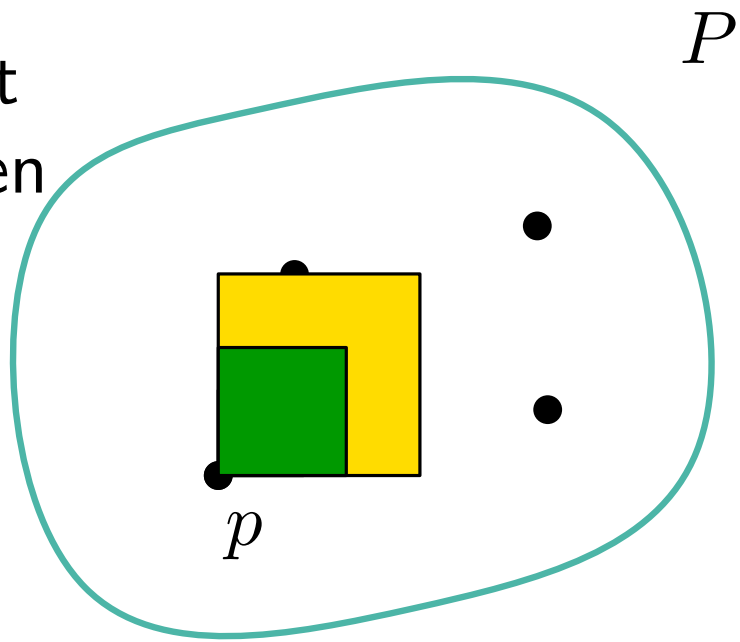


# Übungsblatt 2 - Aufgabe 3

Gegeben: Endliche Punktmenge $P$

*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



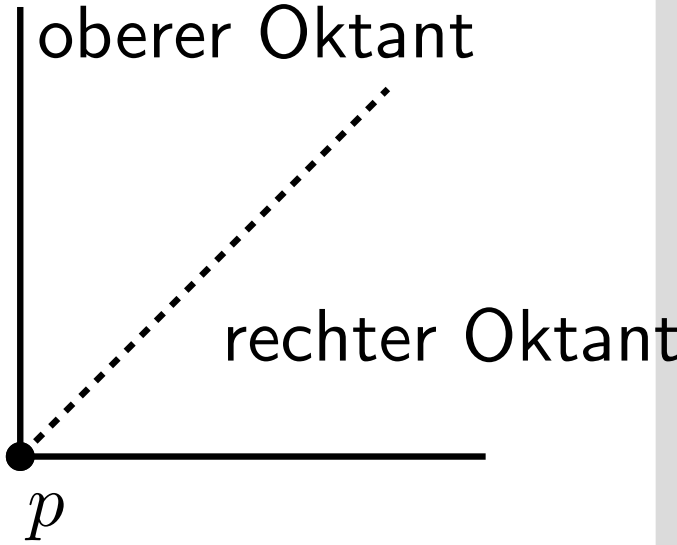
# Übungsblatt 2 - Aufgabe 3

Gegeben: Endliche Punktmenge $P$

*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?



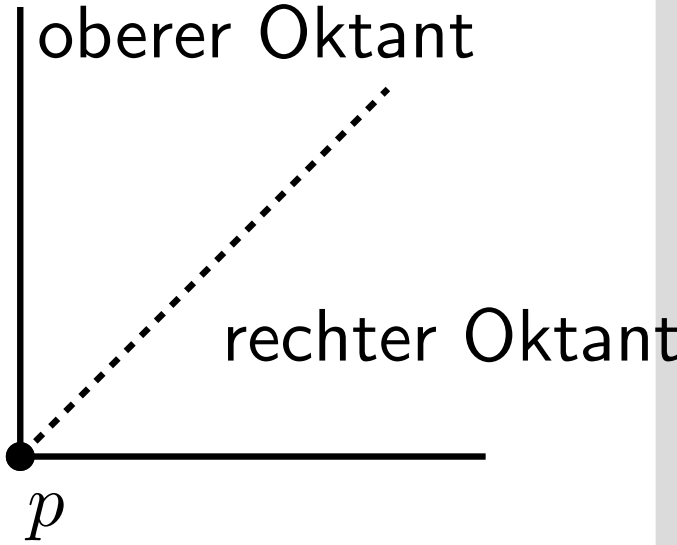
# Übungsblatt 2 - Aufgabe 3

Gegeben: Endliche Punktmenge $P$

*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 $p$

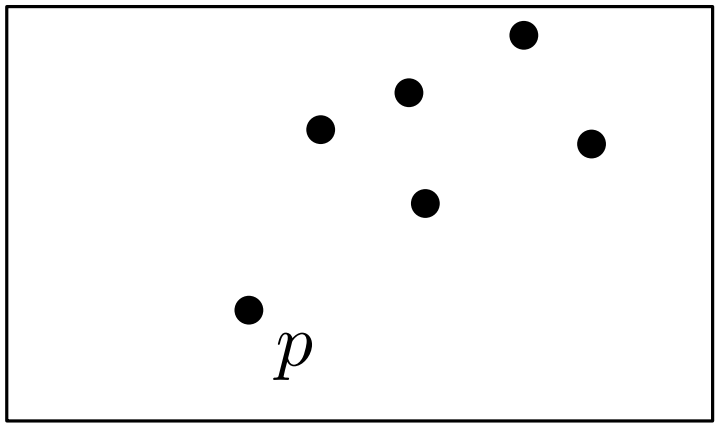
next_y = erster Knoten oberhalb von $p$

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 $p$

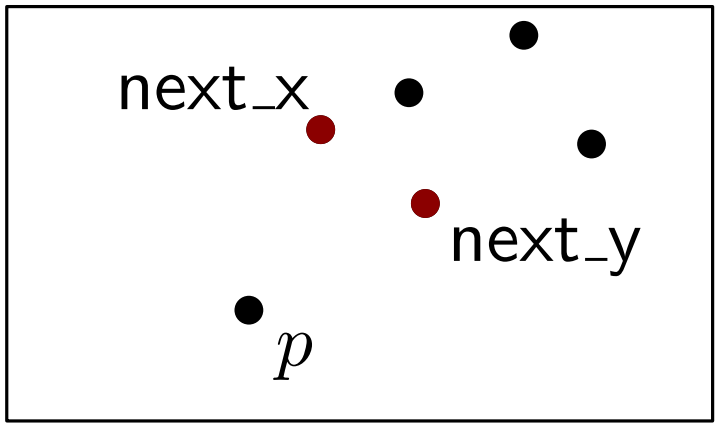
  next_y = erster Knoten oberhalb von $p$

  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

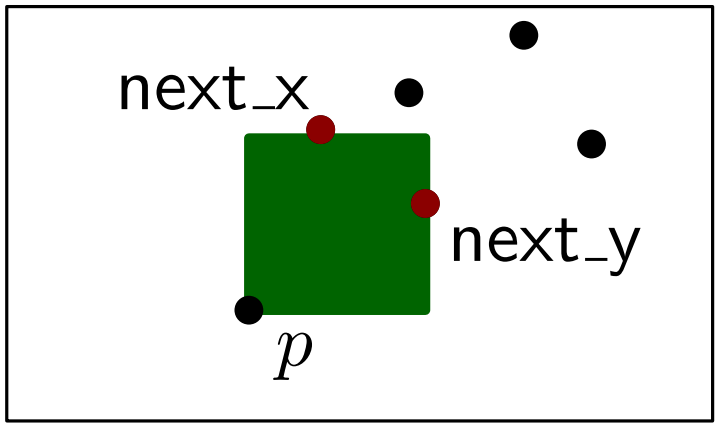
**foreach** $p \in P$ **do**

- 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

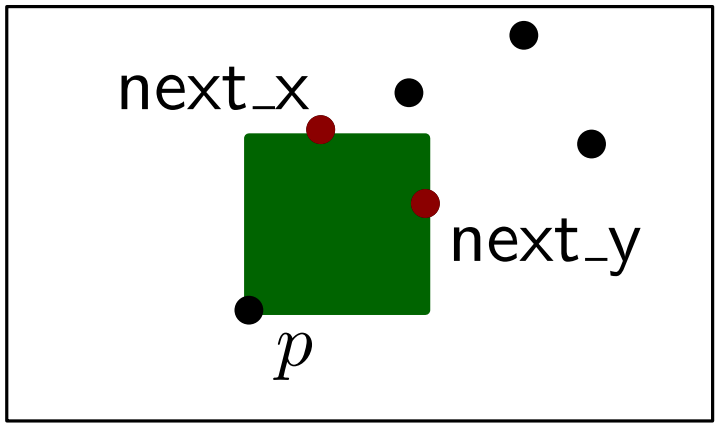
**foreach** $p \in P$ **do**

- 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 $p$ — $\mathcal{O}(1)$

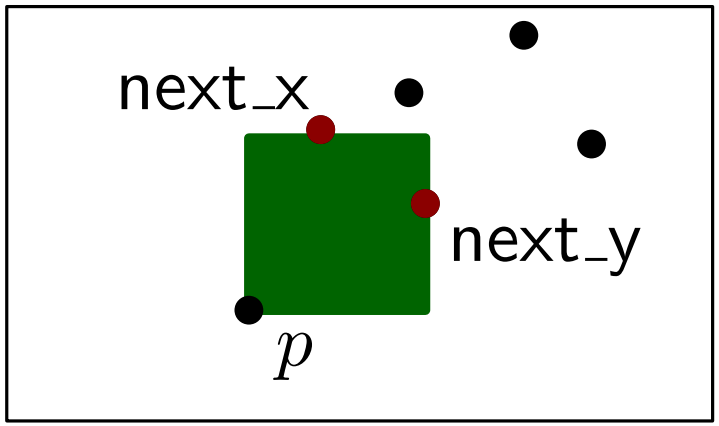
  next_y = erster Knoten oberhalb von $p$

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

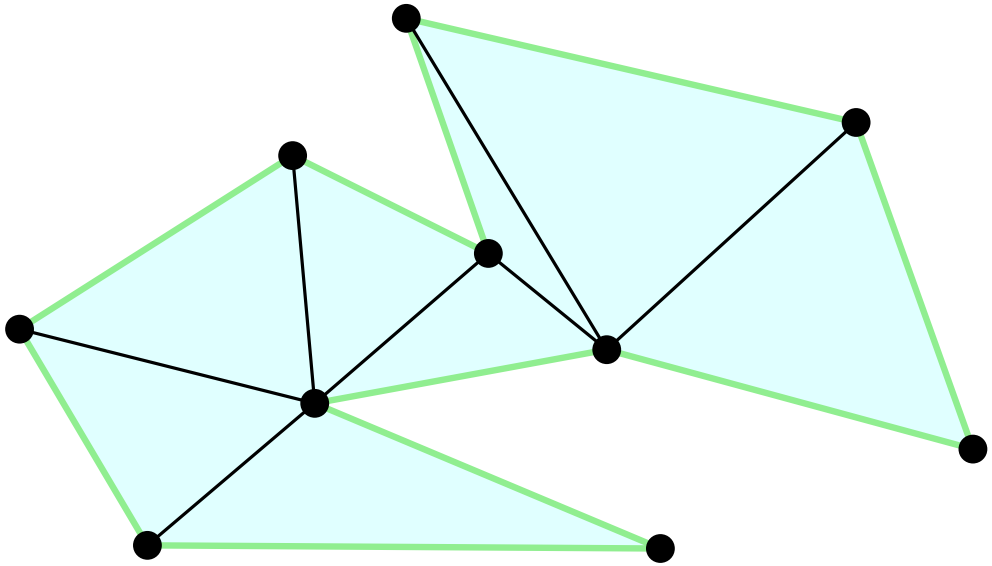
return groB

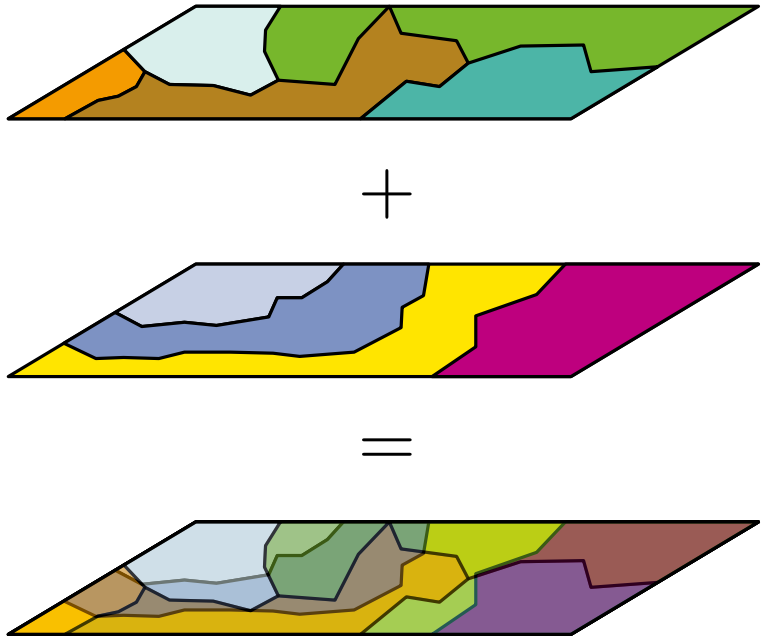
Korrekt?

# Doppelt verkettete Kantenliste

Wofür?

- Overlay-Konstruktion
- Polygontriangulierung

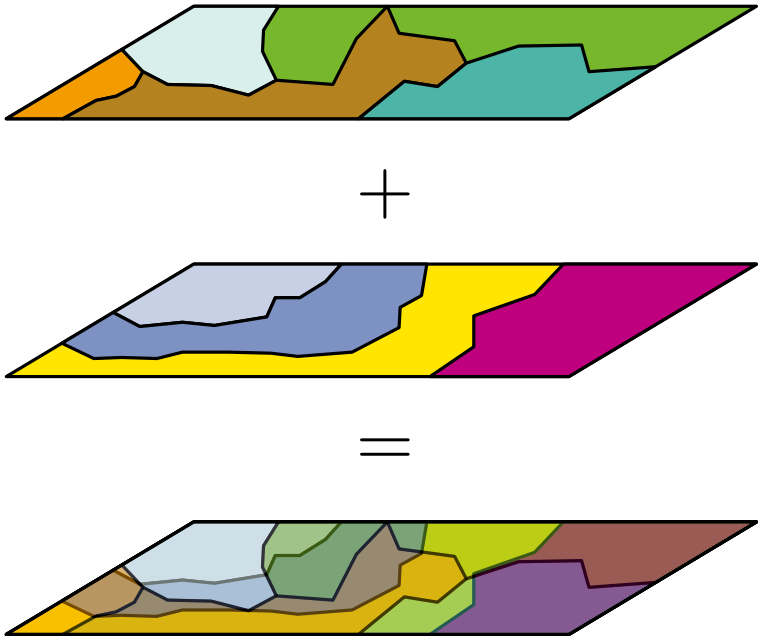


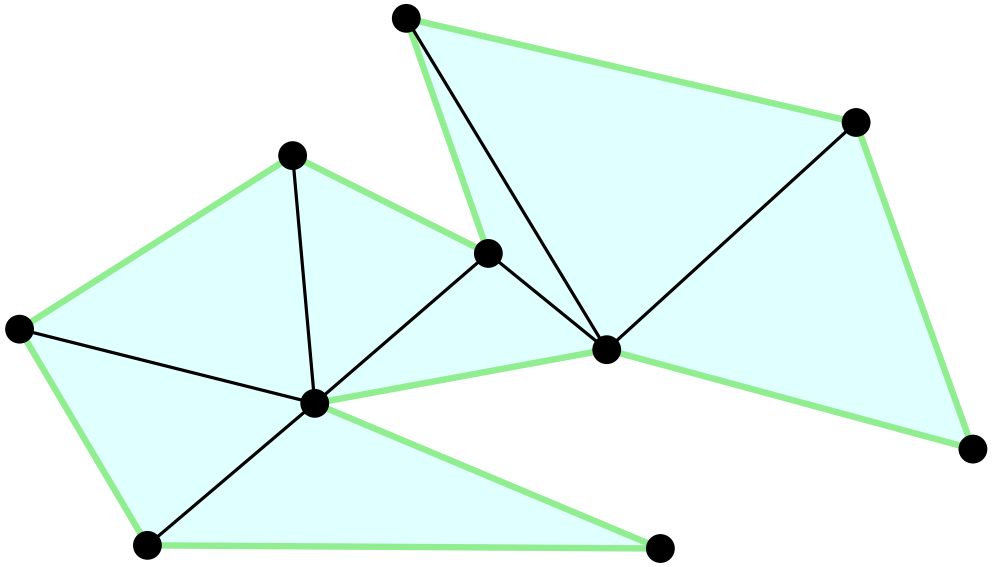


# Doppelt verkettete Kantenliste

Wofür?

- Overlay-Konstruktion
- Polygontriangulierung



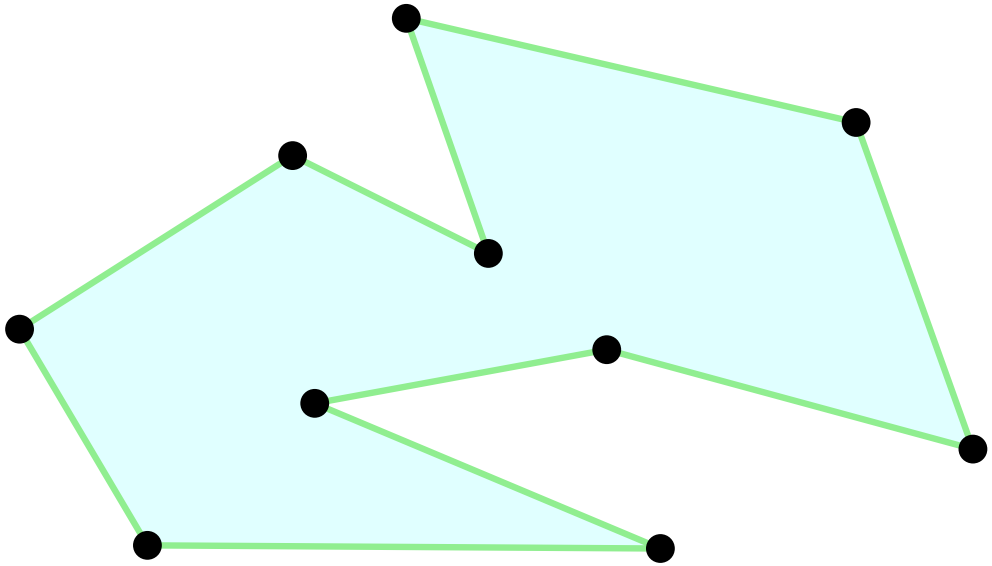


Einschränkung auf geradlining, planar eingebettete Graphen

# Doppelt verkettete Kantenliste

Anforderungen:

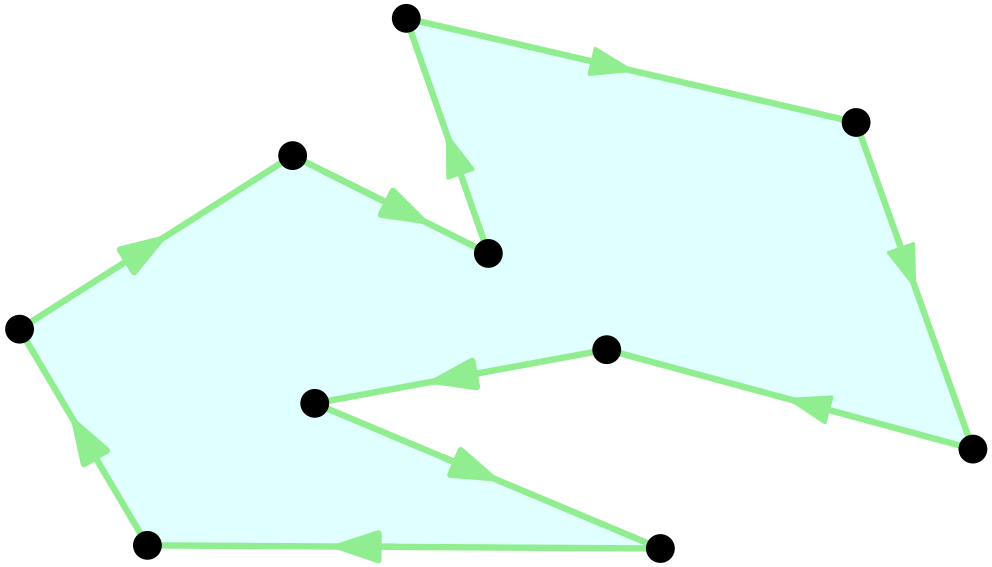
- Auf dem Rand eines Polygons entlanglaufen
- Benachbarte Facette ausgeben



# Doppelt verkettete Kantenliste

Anforderungen:

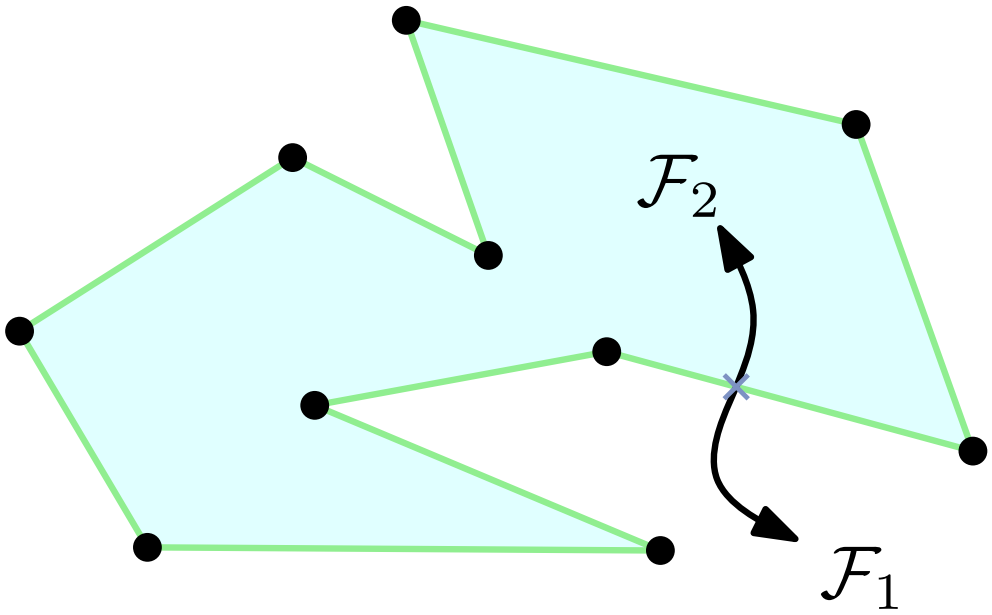
- Auf dem Rand eines Polygons entlanglaufen
- Benachbarte Facette ausgeben



# Doppelt verkettete Kantenliste

Anforderungen:

- Auf dem Rand eines Polygons entlanglaufen
- Benachbarte Facette ausgeben

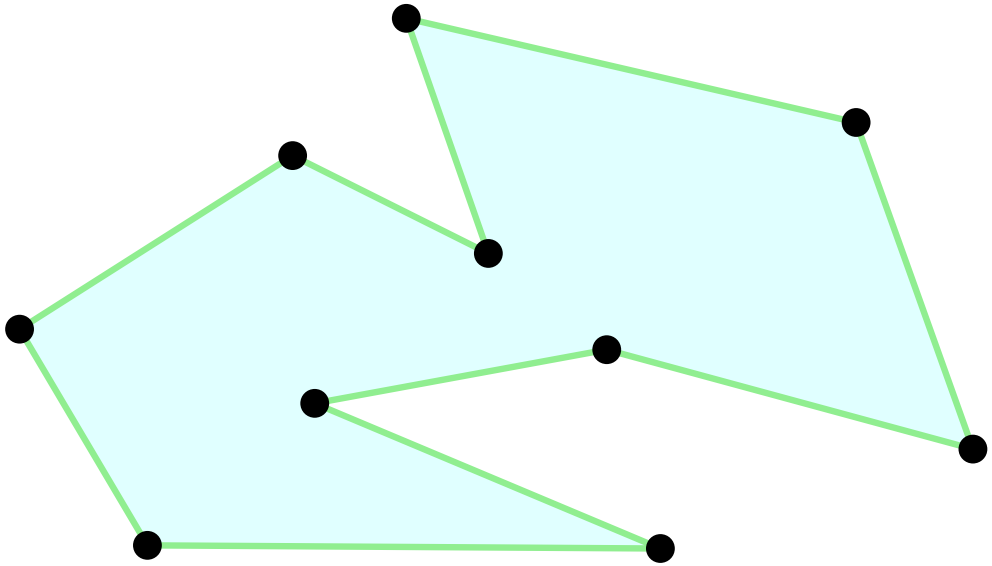


# Doppelt verkettete Kantenliste

Anforderungen:

- Auf dem Rand eines Polygons entlanglaufen
- Benachbarte Facette ausgeben

Eintrag pro Facette, Kante und Knoten



# Doppelt verkettete Kantenliste

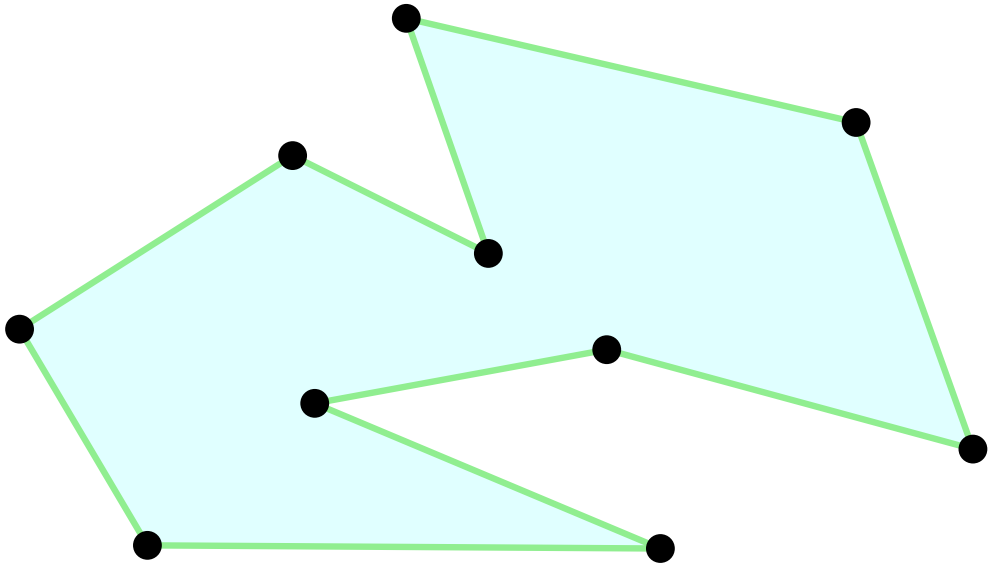
Anforderungen:

- Auf dem Rand eines Polygons entlanglaufen
- Benachbarte Facette ausgeben

---

Eintrag pro Facette, Kante und Knoten

Speichere pro Kanten einen Pointer auf seinen Vorgänger und Nachfolger



# Doppelt verkettete Kantenliste

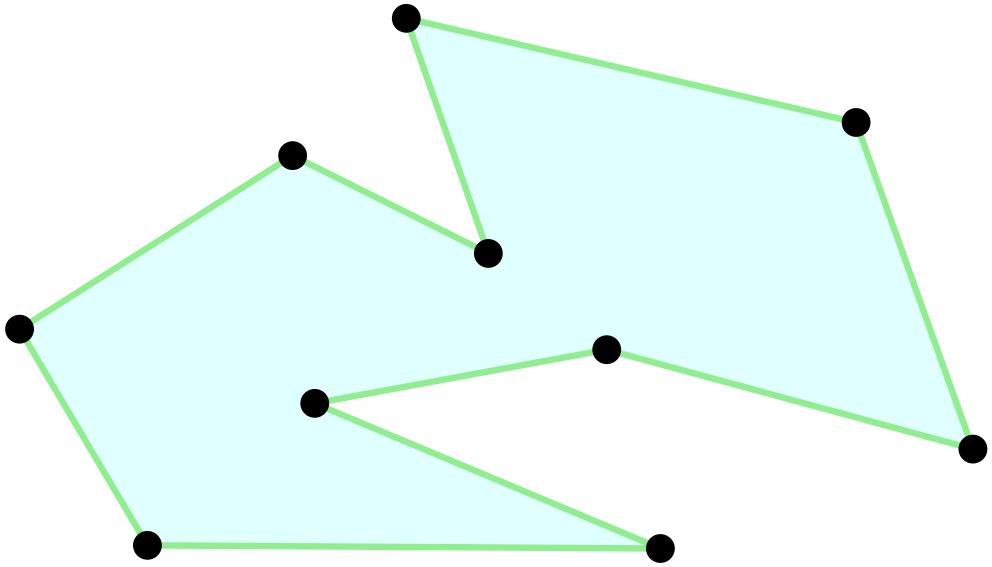
Anforderungen:

- Auf dem Rand eines Polygons entlanglaufen
- Benachbarte Facette ausgeben

---

Eintrag pro Facette, Kante und Knoten

Speichere pro Kanten einen Pointer auf seinen Vorgänger und Nachfolger



Kanten
- *twin*
- *origin/destination*
- *next/prev*
- *incidentFace*

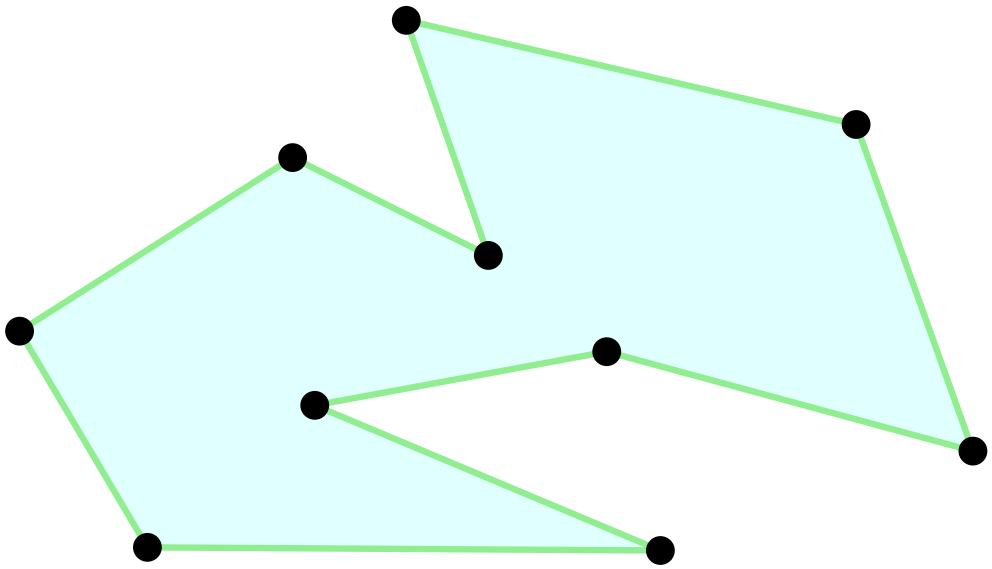
Knoten
- *incidentEdge*
- *coordinates*

Facetten
- *innerComponent*
- *outerComponent*

# Doppelt verkettete Kantenliste

Kante $e$:

- twin(twin($e$)) $= e$?
- next(prev($e$)) $= e$?
- twin(prev(twin($e$))) $=$ next($e$)?
- incidentFace($e$) $=$ incidentFace(next($e$))?



Kanten
- *twin*
- *origin/destination*
- *next/prev*
- *incidentFace*

Knoten
- *incidentEdge*
- *coordinates*

Facetten
- *innerComponent*
- *outerComponent*

# Das war's!

Karlsruhe Institute of Technology

## Fragen?

Nächster Termin:

**Donnerstag, 05.05, 10:15 Uhr**

Raum 131, Gebäude 50.34