Karlsruhe Institute of Technology

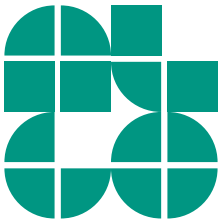
Übung Algorithmische Geometrie

Konvexe Hülle im $\mathbb{R}^3$ + WSPD

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

Andreas Gemsa

12.07.2012



Ablauf

Karlsruhe Institute of Technology

Konvexe Hülle im $\mathbb{R}^3$

WSPD

Neuer Algorithmus

Aufgabe 1

Worst Case Laufzeit

a) Zeige, dass die worst-case Laufzeit von ConvexHull in $\mathcal{O}(n^3)$ liegt.

```
3dConvexHull(P ⊂ ℝ³)
    wähle P' ⊆ P (4 nicht koplan. Punkte)
    C ← CH(P'); P ← P \ P'
    P = RandomPermutation(P)
    initializiere Konfliktgraph G
    while P ≠ ∅ do
        p ← P.remove_first()
        if F_conflict(p) ≠ ∅ then
            Lösche alle Facetten F_conflict(p) aus C
            ℒ ← Horizont-Kanten die von p aus sichtbar sind
            foreach e ∈ ℒ do
                f ← erzg. Facette_C(e, p); erzg. Knoten für f in G
                (f1, f2) ← vorher_inzident_C(e)
                P(e) ← P_conflict(f1) ∪ P_conflict(f2)
                foreach p ∈ P(e) do
                    if f sichtbar von p then add_edge(p, f) zu G
            lösche Knoten {p} ∪ F_conflict(p) von G
```

Aufgabe 1

Worst Case Laufzeit

a) Zeige, dass die worst-case Laufzeit von ConvexHull in $\mathcal{O}(n^3)$ liegt.

b) Zeige, dass es Punktmengen gibt für die ConvexHull tatsächlich eine Laufzeit von $\Theta(n^3)$ benötigt, wenn eine "schlechte" Permutation erzeugt wird.

Aufgabe 1

Worst Case Laufzeit

a) Zeige, dass die worst-case Laufzeit von ConvexHull in $\mathcal{O}(n^3)$ liegt.

b) Zeige, dass es Punktmengen gibt für die ConvexHull tatsächlich eine Laufzeit von $\Theta(n^3)$ benötigt, wenn eine "schlechte" Permutation erzeugt wird.

Aufgabe 2

Alternativer-Algorithmus

Idee: 'rotiere' Ebene um bereits bekannte Kante der konvexen Hülle. Laufzeit $\mathcal{O}(n^2)$. Beschreibung?

Aufgabe 2

Alternativer-Algorithmus

Idee: 'rotiere' Ebene um bereits bekannte Kante der konvexen Hülle. Laufzeit $\mathcal{O}(n^2)$. Beschreibung?

Woher kennen wir diese Idee... ?

a long long time ago...

Karlsruhe Institute of Technology

a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

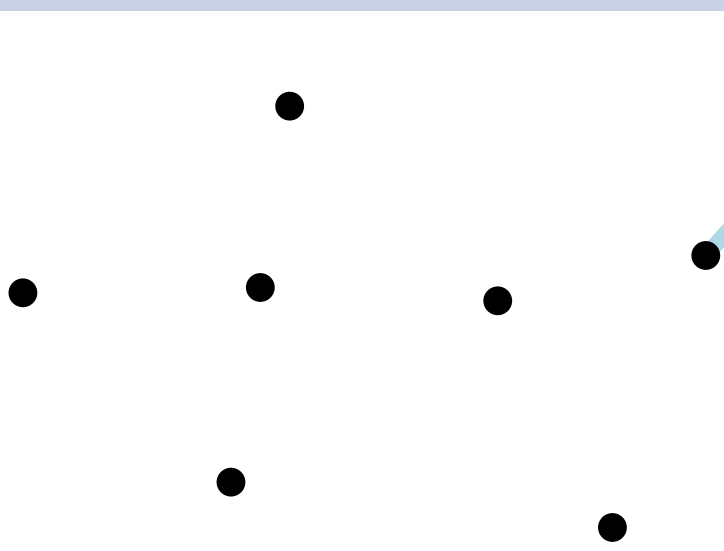
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

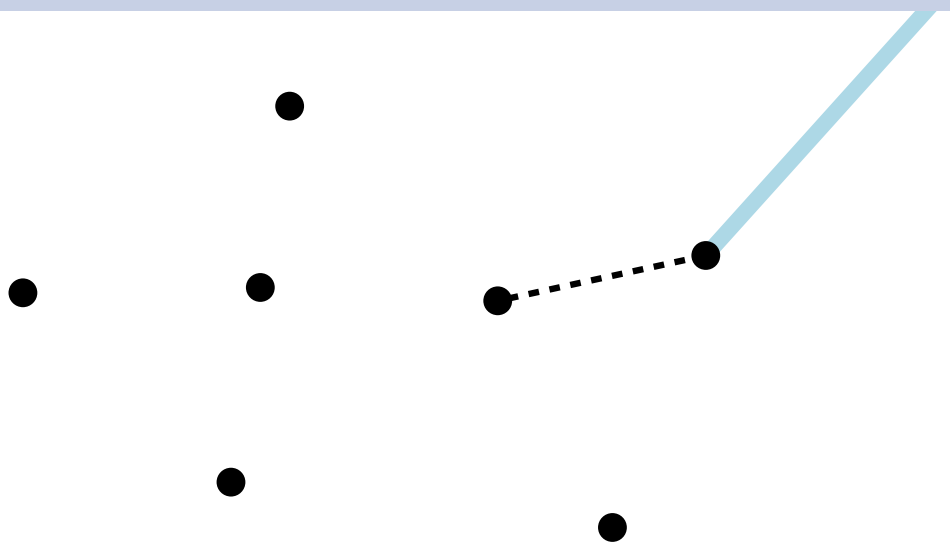
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

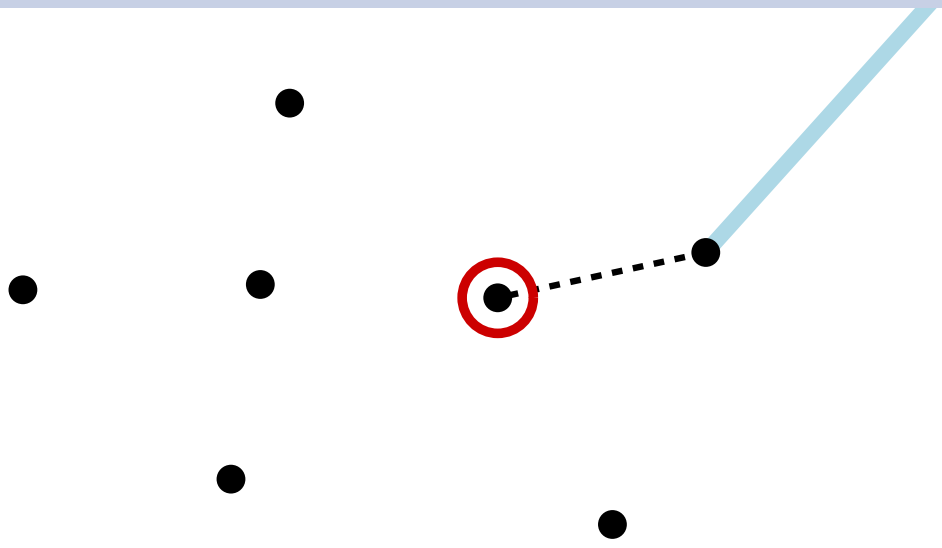
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

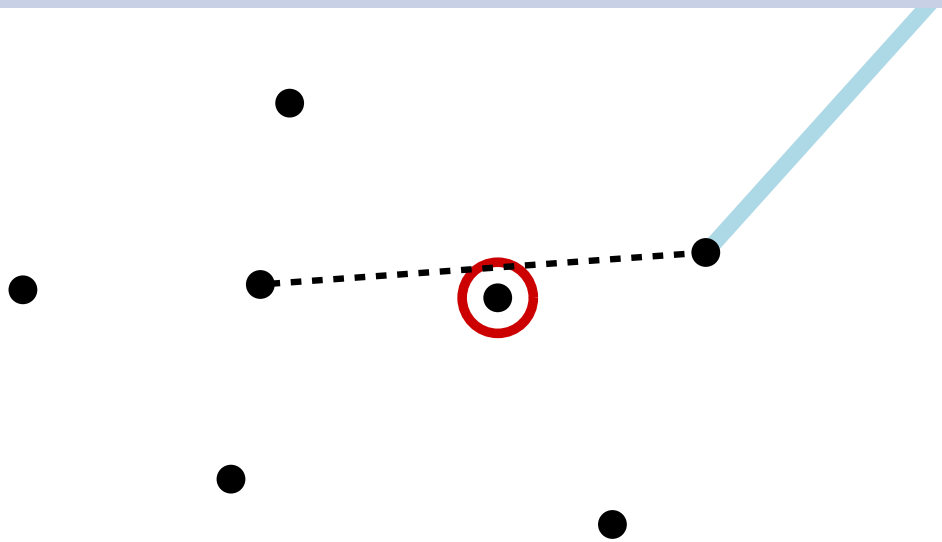
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

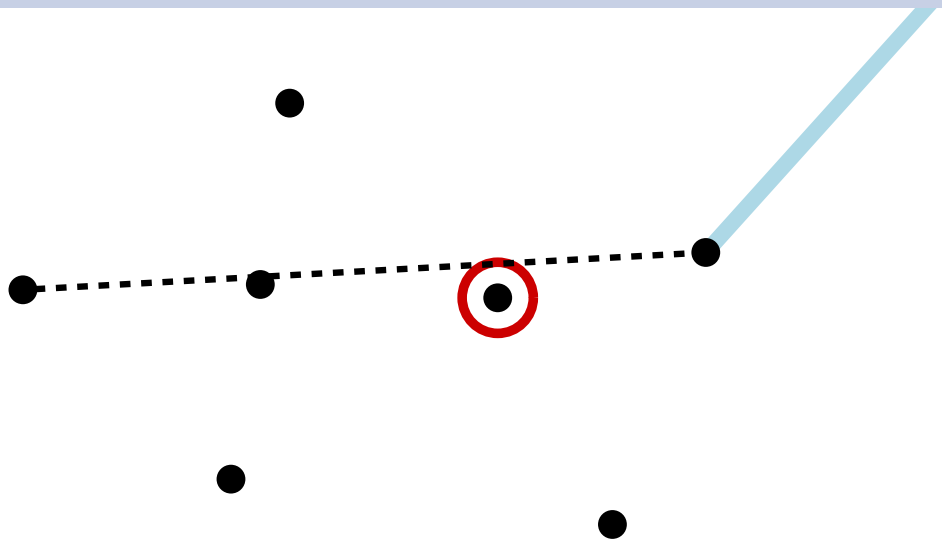
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

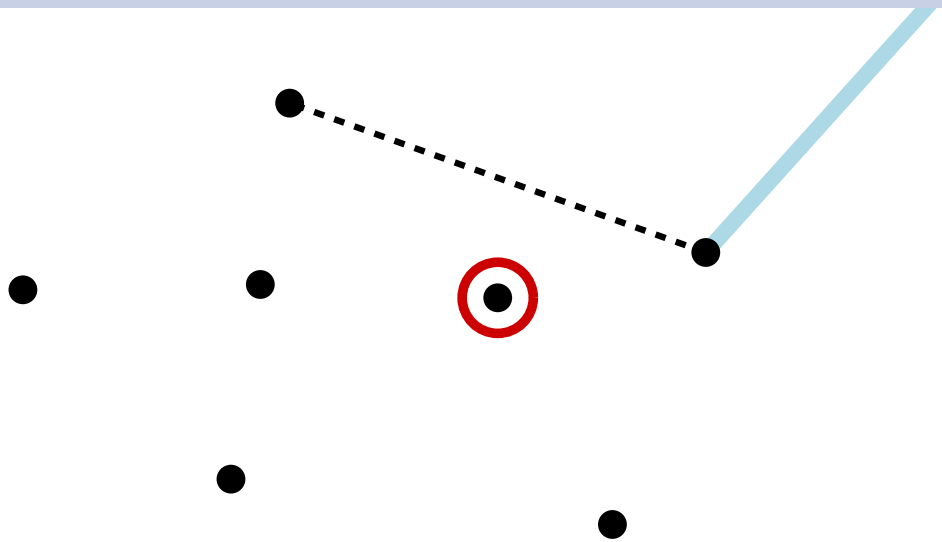
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

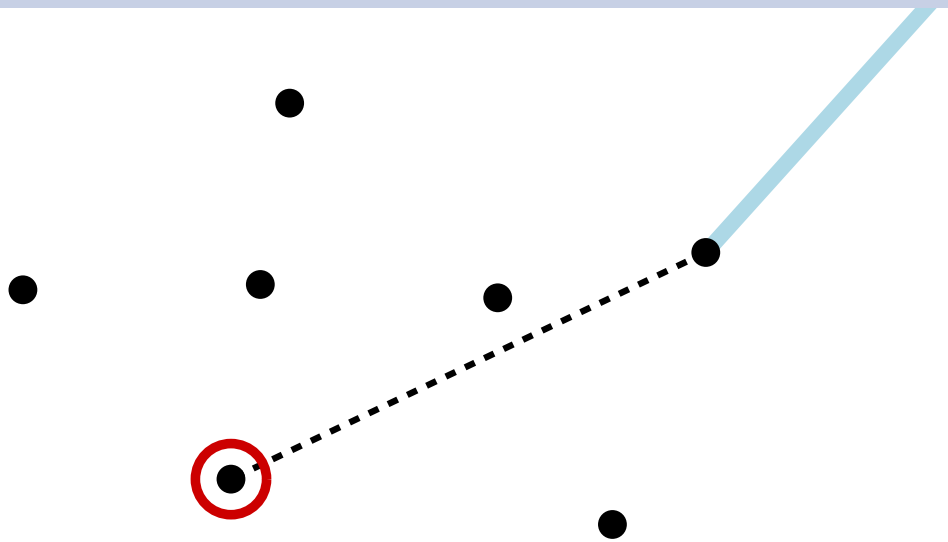
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

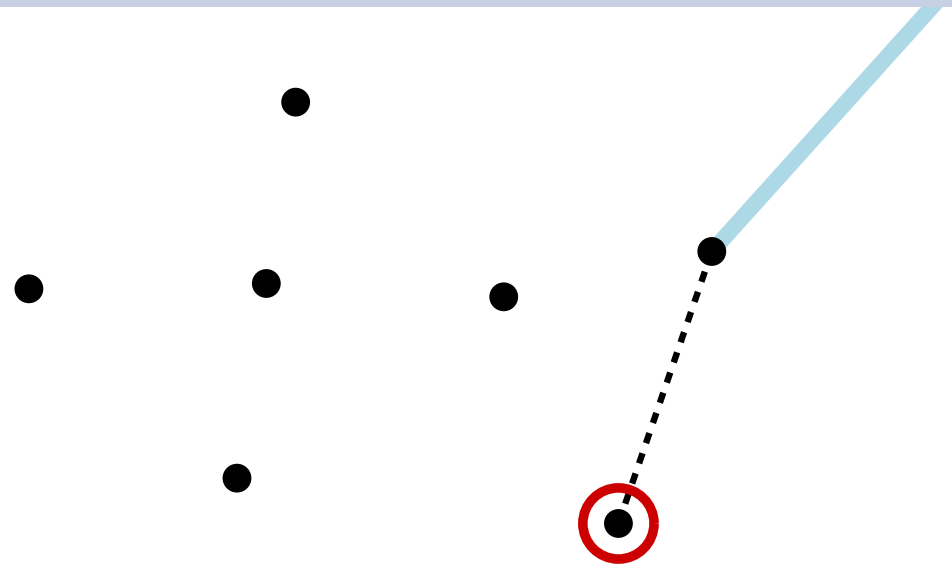
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

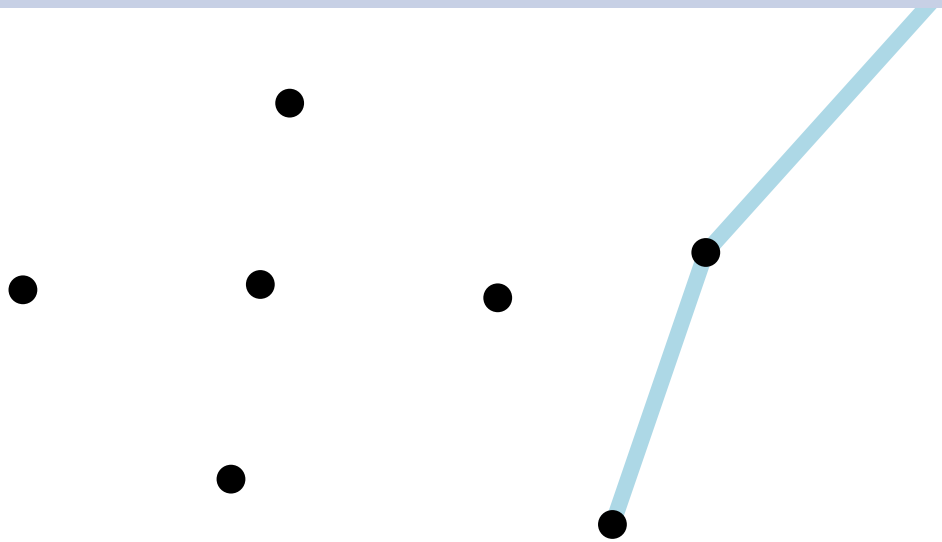
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

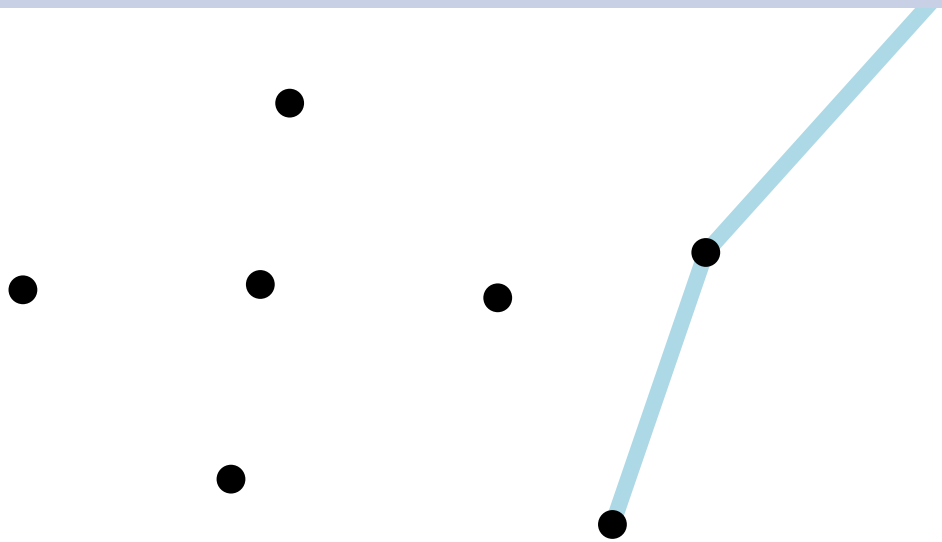
while true **do**

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

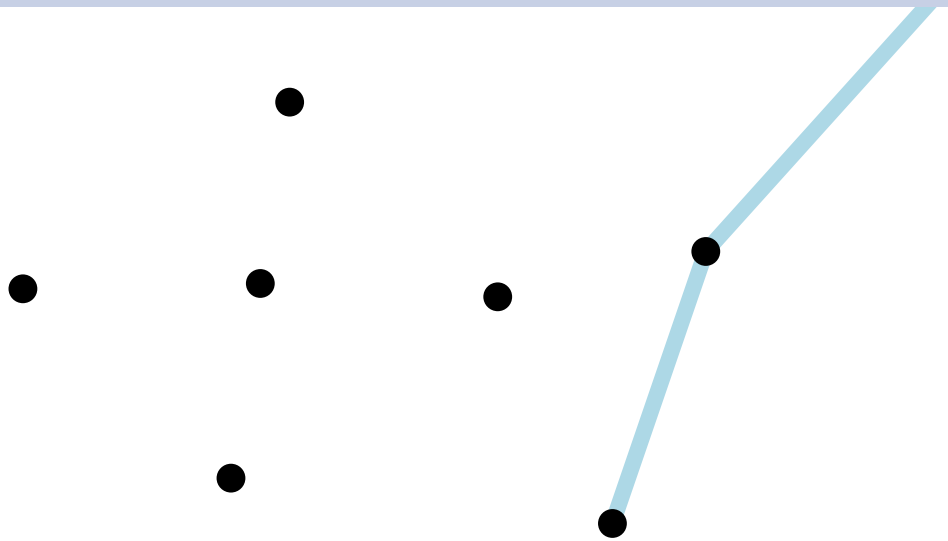
while true **do** $\mathcal{O}(n)$

  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

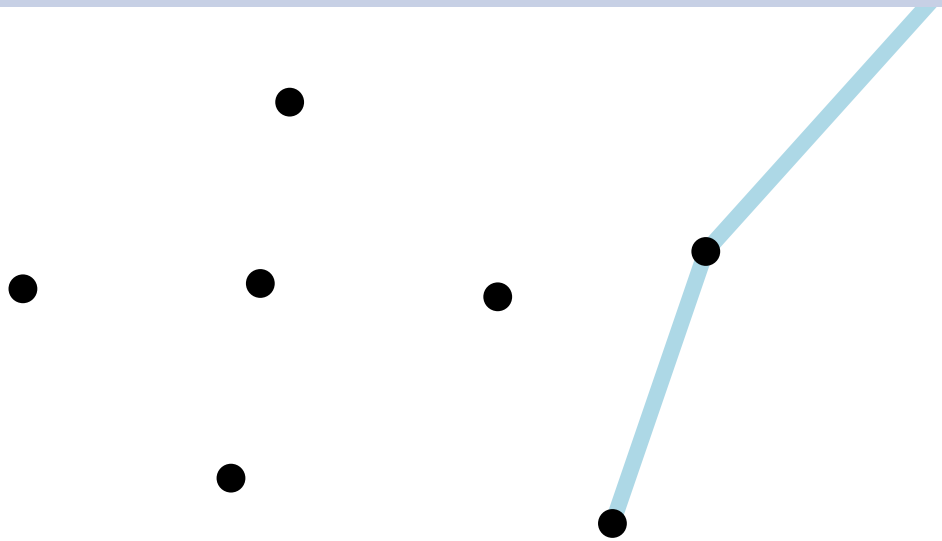
while true **do** $\mathcal{O}(h)$ $\mathcal{O}(n)$

    Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

    if $p_{j+1} == p_1$ **then**

        break

return $(p_1, \ldots, p_h)$



a long long time ago...

GiftWrapping(P)

$p_0 = (\infty, \infty)$, $p_1 =$ rechtester Knoten in P

while true **do** $\mathcal{O}(h)$ $\mathcal{O}(n)$

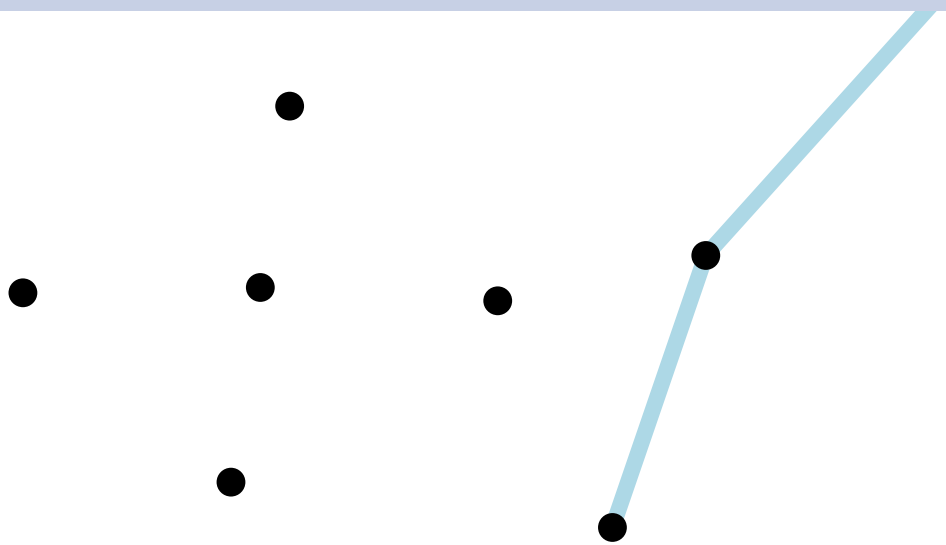
  Wähle $p_{j+1} \in P$ der den Winkel $p_{j-1}p_jp_{j+1}$ maximiert.

  if $p_{j+1} == p_1$ **then**

    break

return $(p_1, \ldots, p_h)$

Laufzeit: $\mathcal{O}(nh)$



Aufgabe 3

Zufallszahlen

Zufallszahlengenerator kann in $O(1)$ nur ein zufälliges Bit erzeugen.

a) Gebe Verfahren an um damit eine zufällige Permutation von n Zahlen zu bestimmen.

b) Was is die Laufzeit von dem in a) beschriebenem Verfahren?

Aufgabe 3

Karlsruhe Institute of Technology

$\textsc{RandomPermutation}(A)$

Input: Array $A[1 \ldots n]$
Output: Array A, zufällig gleichverteilt permutiert
for $k \leftarrow n$ **to** 2 **do**
    $r \leftarrow \textsc{RandomInt}(k)$
    tausche $A[r]$ und $A[k]$

Aufgabe 3

Karlsruhe Institute of Technology

$\textsc{RandomInt}(k)$

$\mathcal{T} = k$ Zufallsbits

return Anzahl 1en in $\mathcal{T}$

$\textsc{RandomPermutation}(A)$

Input: Array $A[1 \ldots n]$

Output: Array A, zufällig gleichverteilt permutiert

for $k \leftarrow n$ **to** 2 **do**

  $r \leftarrow \textsc{RandomInt}(k)$

  tausche $A[r]$ und $A[k]$

Aufgabe 3

Karlsruhe Institute of Technology

$\textsc{RandomInt}(k)$

$d = \lceil \log_2 k \rceil$
result $= 0$
for $i \leftarrow 1$ to d **do**
    result $=$ result $+ 2^i \cdot$ zufälliges Bit
return result

$\textsc{RandomPermutation}(A)$

Input: Array $A[1 \ldots n]$
Output: Array A, zufällig gleichverteilt permutiert
for $k \leftarrow n$ **to** 2 **do**
    $r \leftarrow \textsc{RandomInt}(k)$
    tausche $A[r]$ und $A[k]$

Aufgabe 3

Karlsruhe Institute of Technology

$\textsc{RandomInt}(k)$

$d = \lceil \log_2 k \rceil$
result $= 0$
for $i \leftarrow 1$ to d **do**
    result = result + $2^i \cdot$ zufälliges Bit
if result $> k$ **then**
    return $k-$result
return result

$\textsc{RandomPermutation}(A)$

Input: Array $A[1 \ldots n]$
Output: Array A, zufällig gleichverteilt permutiert
for $k \leftarrow n$ **to** 2 **do**
    $r \leftarrow \textsc{RandomInt}(k)$
    tausche $A[r]$ und $A[k]$

Aufgabe 3

Karlsruhe Institute of Technology

$\textsc{RandomInt}(k)$

$d = \lceil \log_2 k \rceil$
result $= 0$
for $i \leftarrow 1$ to d **do**
  result = result + $2^i \cdot$ zufälliges Bit
if result $> k$ **then**
  return $\textsc{RandomInt}$(k);
return result

$\textsc{RandomPermutation}(A)$

Input: Array $A[1 \ldots n]$
Output: Array A, zufällig gleichverteilt permutiert
for $k \leftarrow n$ **to** 2 **do**
  $r \leftarrow \textsc{RandomInt}(k)$
  tausche $A[r]$ und $A[k]$

Ablauf

Karlsruhe Institute of Technology

Konvexe Hülle im $\mathbb{R}^3$

WSPD

Neuer Algorithmus

Aufgabe 1

Karlsruhe Institute of Technology

WSPD

- $x := 2/s + 1$
- $S := \{x^i \mid 0 \leq i \leq n-1\}$

$\mathcal{W} = \{A_j, B_j\}$ beliebige s-WSPD für S $(s > 0)$
$1 \leq j \leq m$

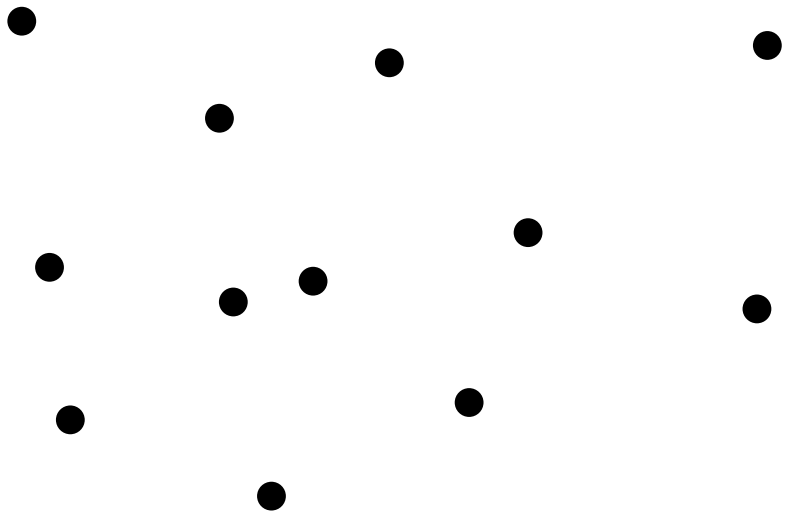
Zeige:

$$\sum_{j=1}^{m}(|A_j| + |B_j|) = \binom{n}{2} + m$$

Hinweis: Für jedes j ist wenigstens eine der Mengen A_j oder B_j ein Singleton.

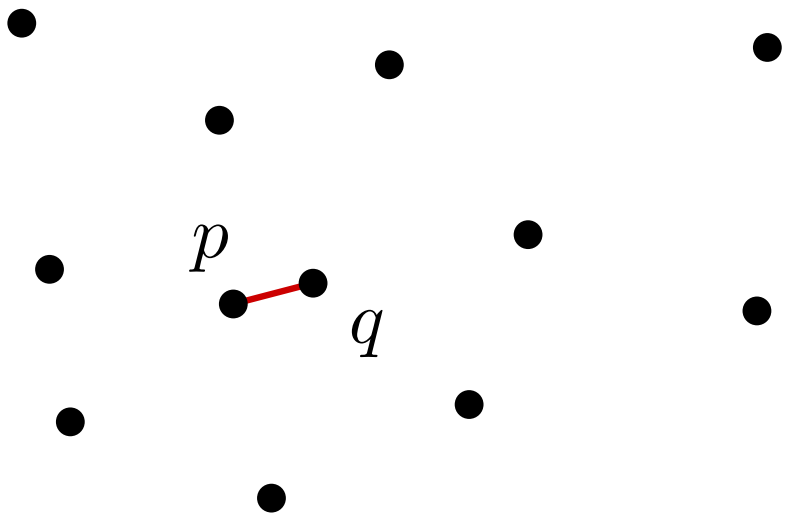
Aufgabe 2/3

- P: n Punkte aus dem $\mathbb{R}^d$



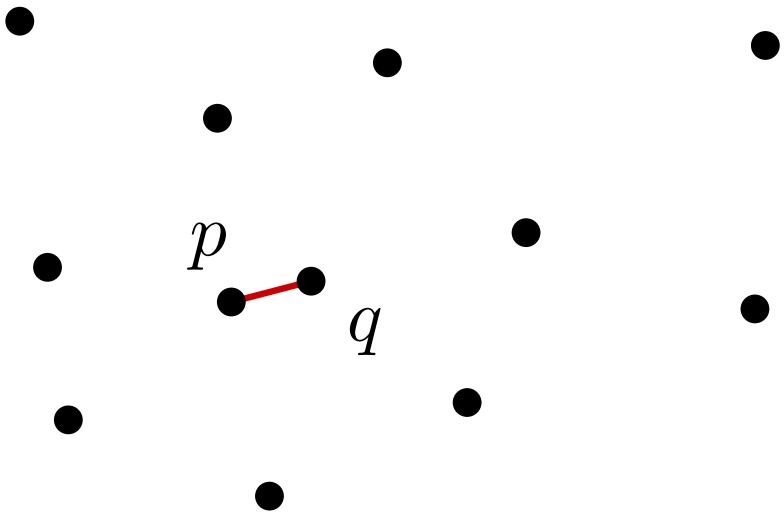
Aufgabe 2/3

- P: n Punkte aus dem $\mathbb{R}^d$
- $p, q \in P$ und Abstand zwischen p und q ist minimal



Aufgabe 2/3

- P: n Punkte aus dem $\mathbb{R}^d$
- $p, q \in P$ und Abstand zwischen p und q ist minimal



Gegeben: s-WSPD für P mit $s > 2$
Für ein Paar $\{A, B\}$ in $\mathcal{W}$ liegt $p \in A$ und $q \in B$

- Zeige, dass dann A ein Singleton ist.
- Zeige, dass die Größe von $\mathcal{W}$ mindestens $n/2$ ist.
- Zeige, dass $\{\{p\}, \{q\}\}$ in $\mathcal{W}$ vorkommt.

Ablauf

Karlsruhe Institute of Technology

Konvexe Hülle im $\mathbb{R}^3$

WSPD

Neuer Algorithmus

Neuer Algorithmus

Karlsruhe Institute of Technology

$\textsc{DivideConquerConvexHull}(P \subset \mathbb{R}^2)$

Neuer Algorithmus

Karlsruhe Institute of Technology

DivideConquerConvexHull($P \subset \mathbb{R}^2$)

if $|P| \leq 3$ **then**

  return $CH(P)$

Halbiere P in linke Teilmenge P_ℓ und rechte Teilmenge P_r

$CH(P_\ell) =$ DivideConquerConvexHull(P_ℓ)

$CH(P_r) =$ DivideConquerConvexHull(P_r)

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$

return $CH(P)$

Neuer Algorithmus

DivideConquerConvexHull($P \subset \mathbb{R}^2$)

if $|P| \leq 3$ **then**

  return $CH(P)$

Halbiere P in linke Teilmenge P_ℓ und rechte Teilmenge P_r

$CH(P_\ell) =$ DivideConquerConvexHull(P_ℓ)

$CH(P_r) =$ DivideConquerConvexHull(P_r)

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$

return $CH(P)$

Neuer Algorithmus

DivideConquerConvexHull($P \subset \mathbb{R}^2$)

if $|P| \leq 3$ **then**

  return $CH(P)$

Halbiere P in linke Teilmenge P_ℓ und rechte Teilmenge P_r

$CH(P_\ell) =$ DivideConquerConvexHull(P_ℓ)

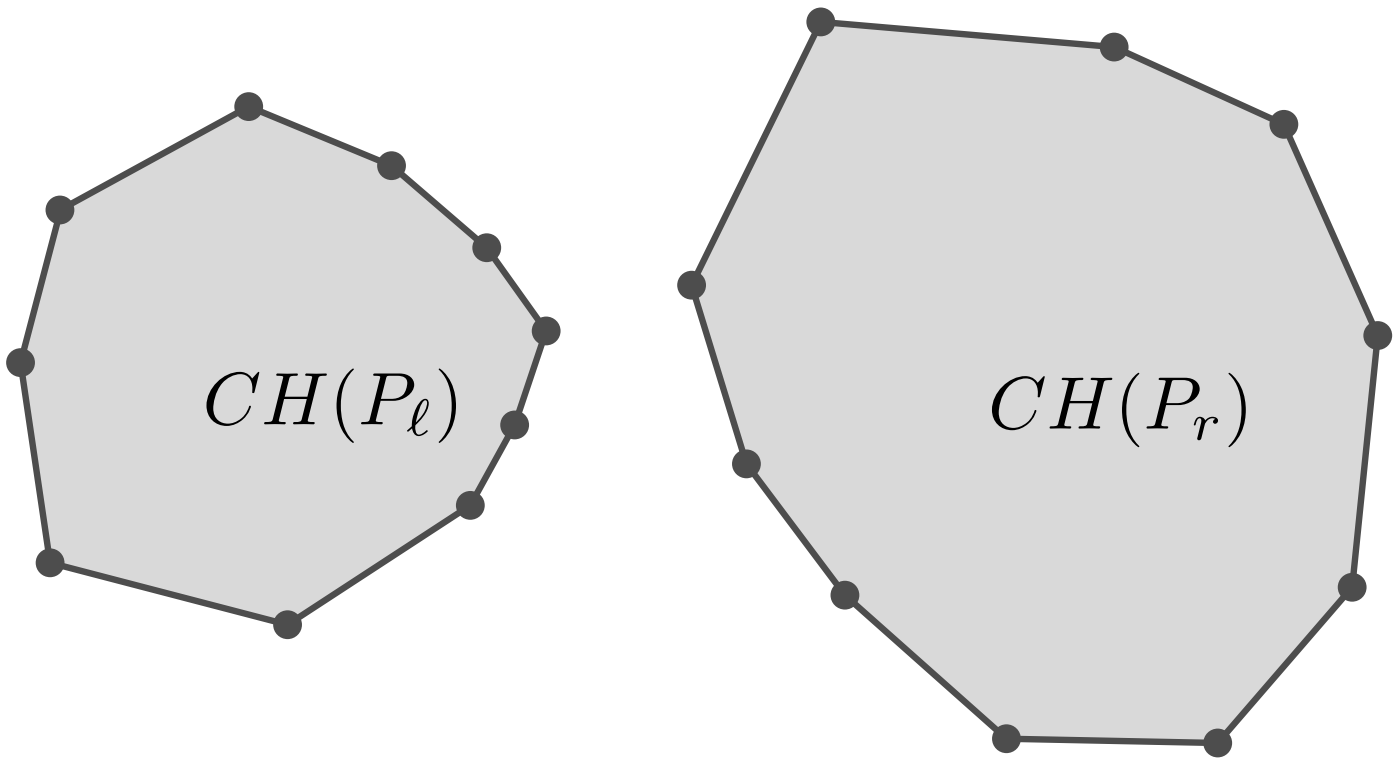
$CH(P_r) =$ DivideConquerConvexHull(P_r)

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$

return $CH(P)$

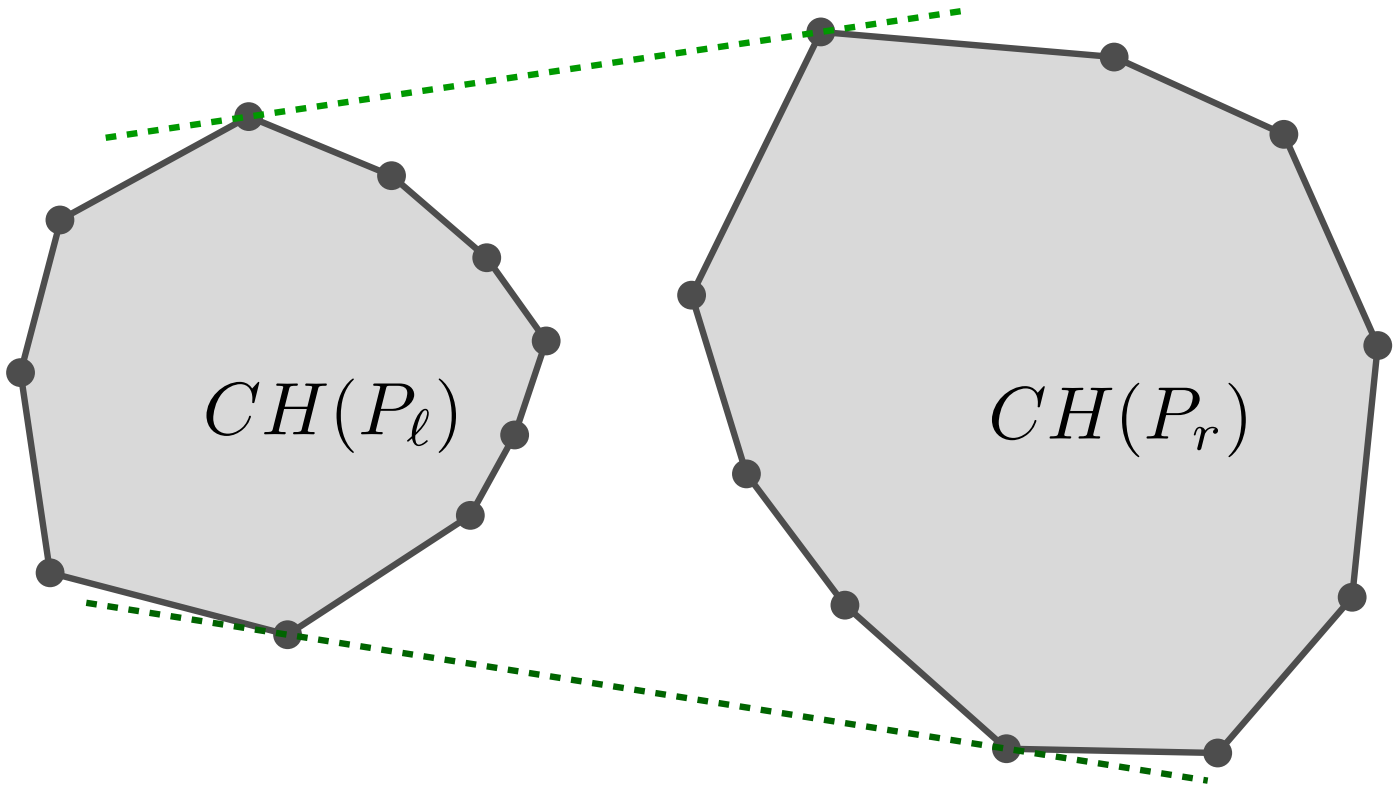
Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



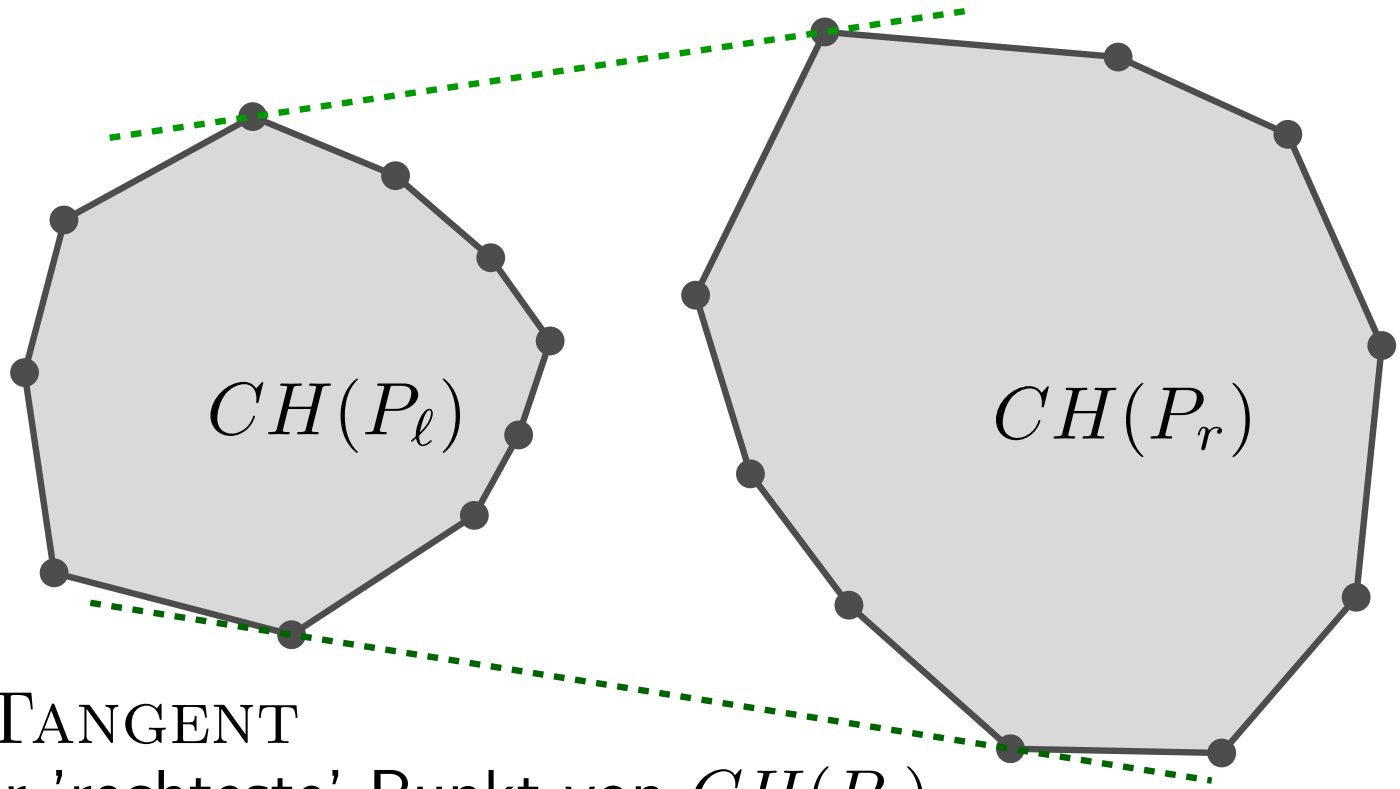
Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

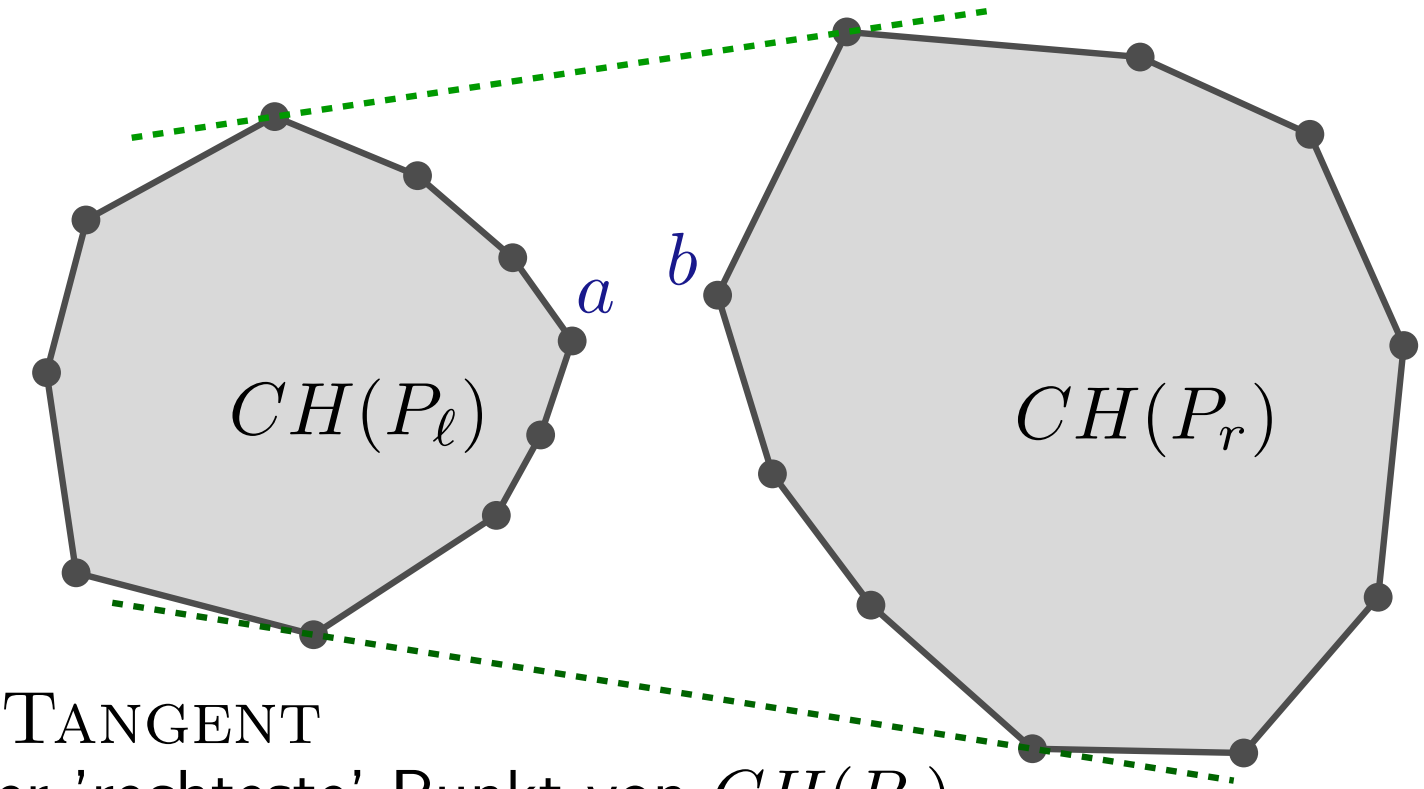
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

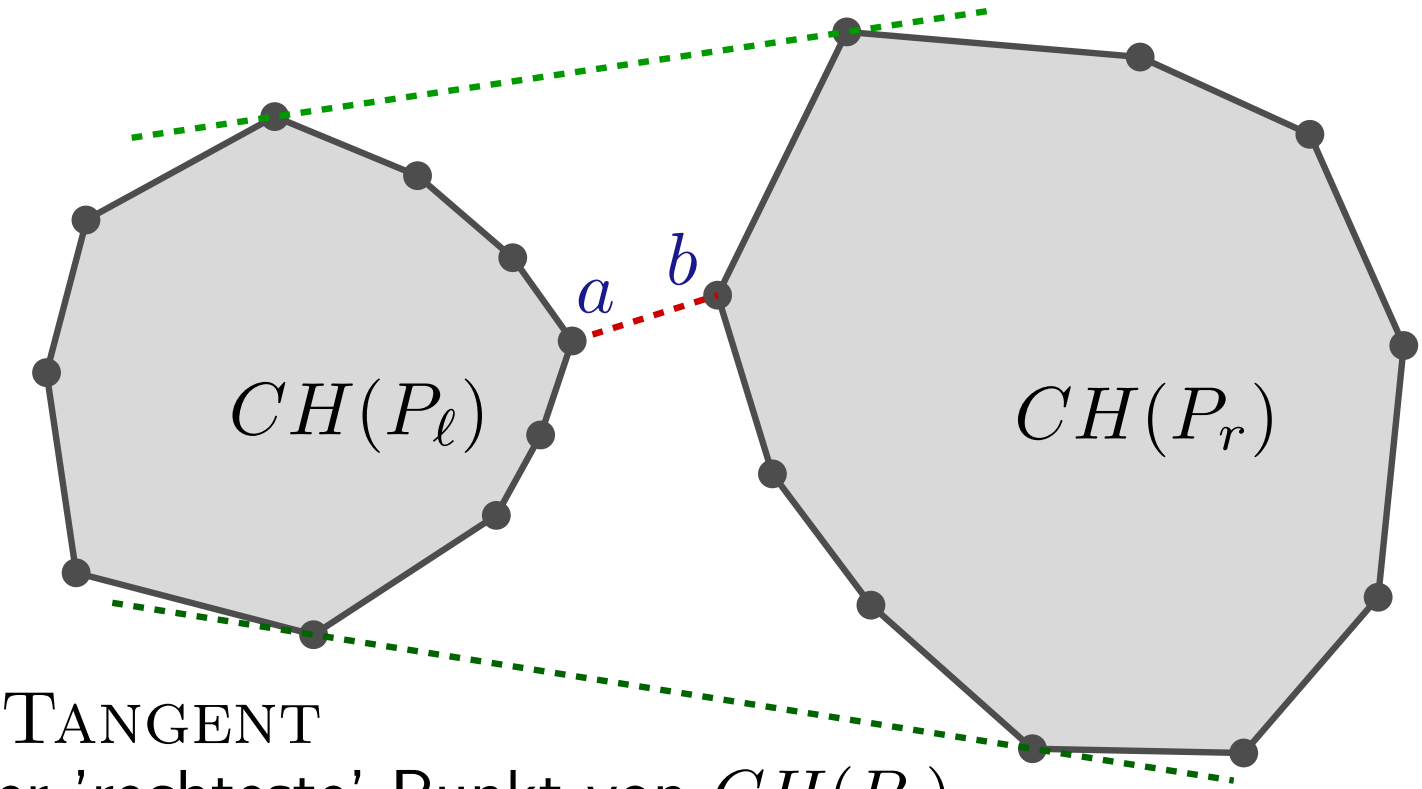
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

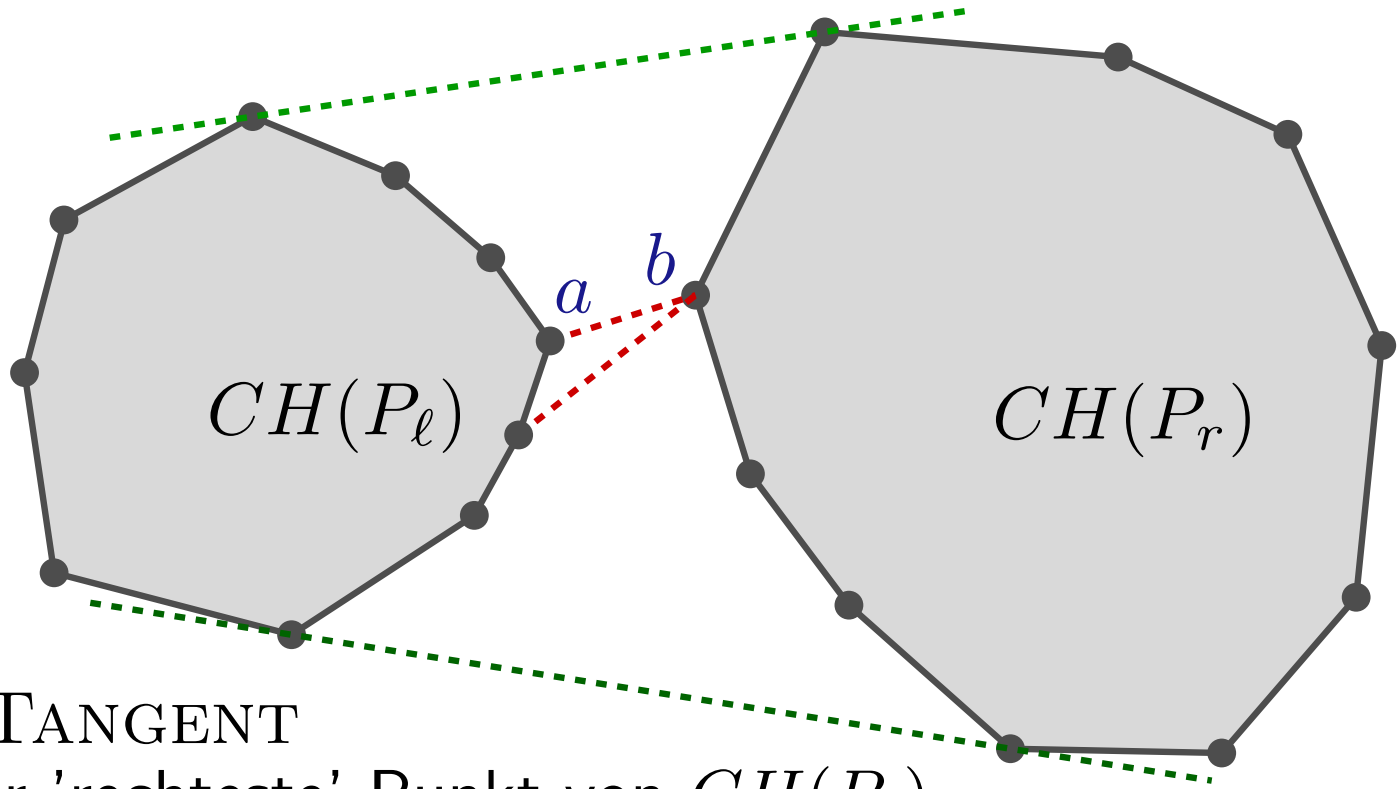
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

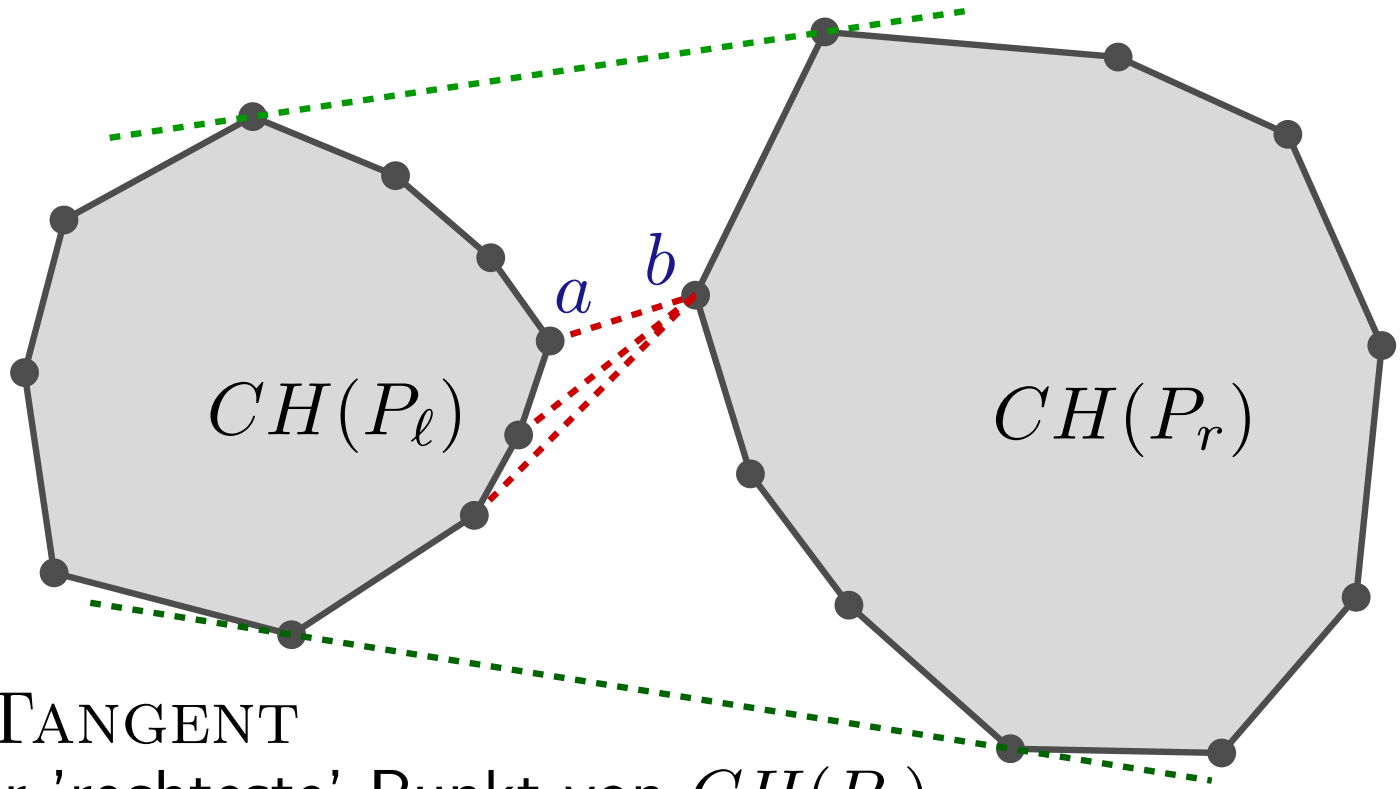
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

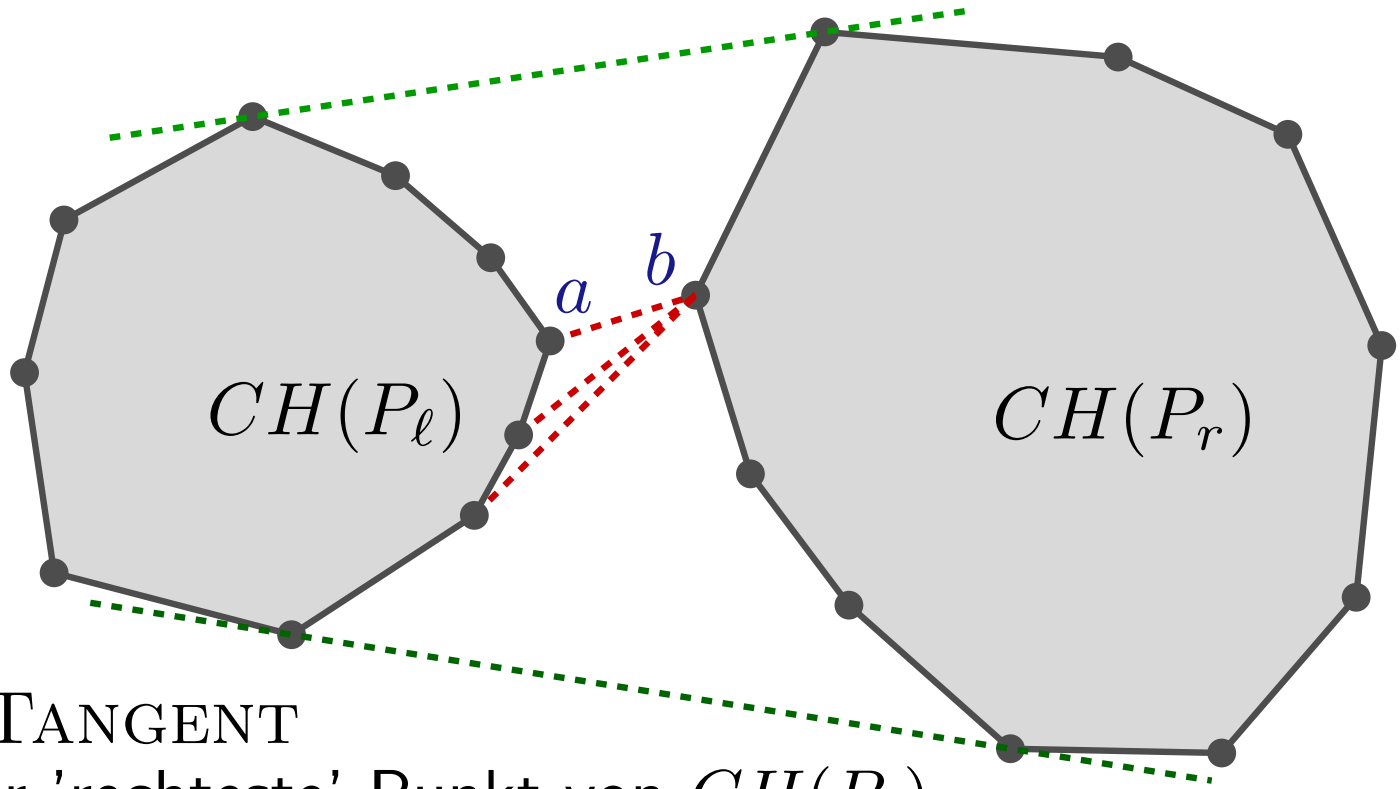
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

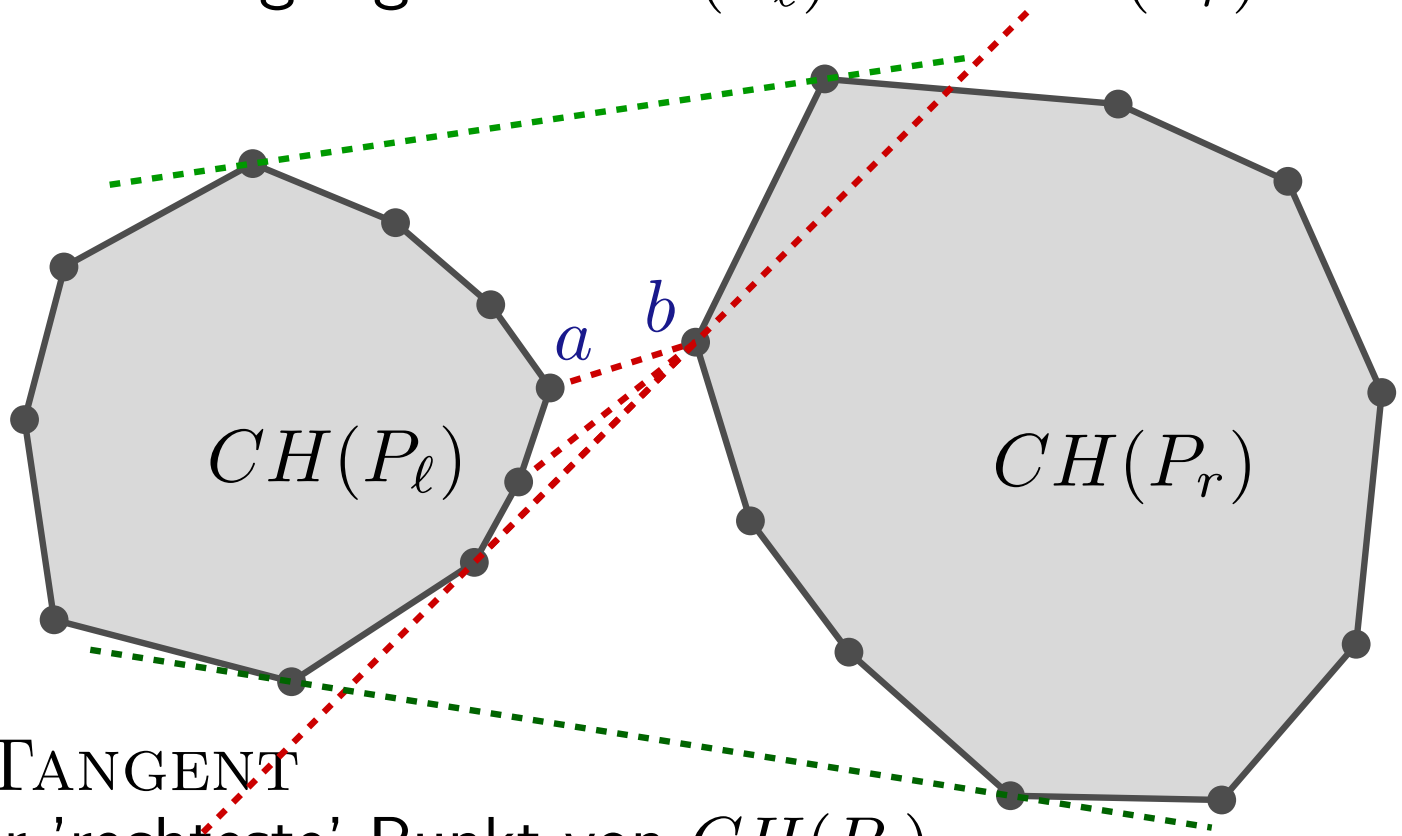
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

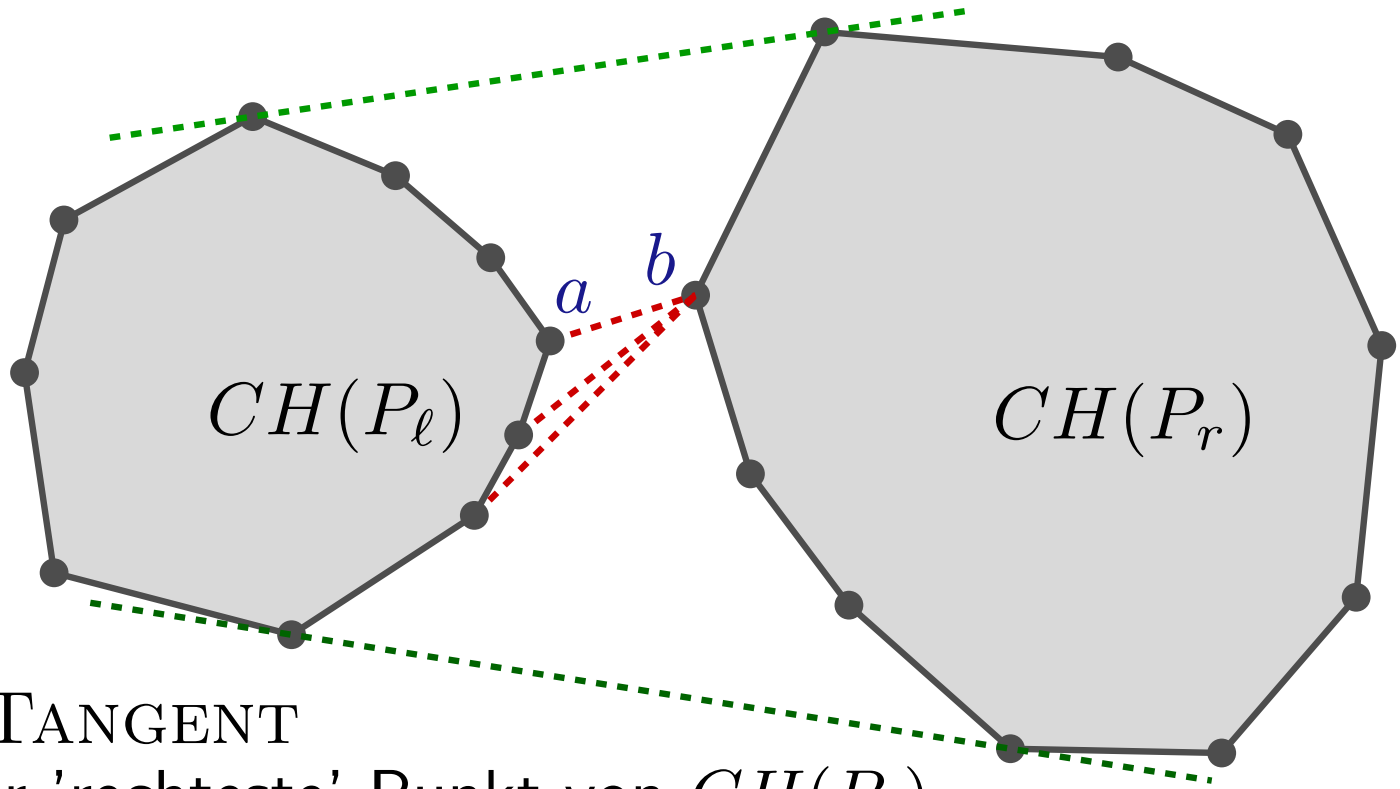
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

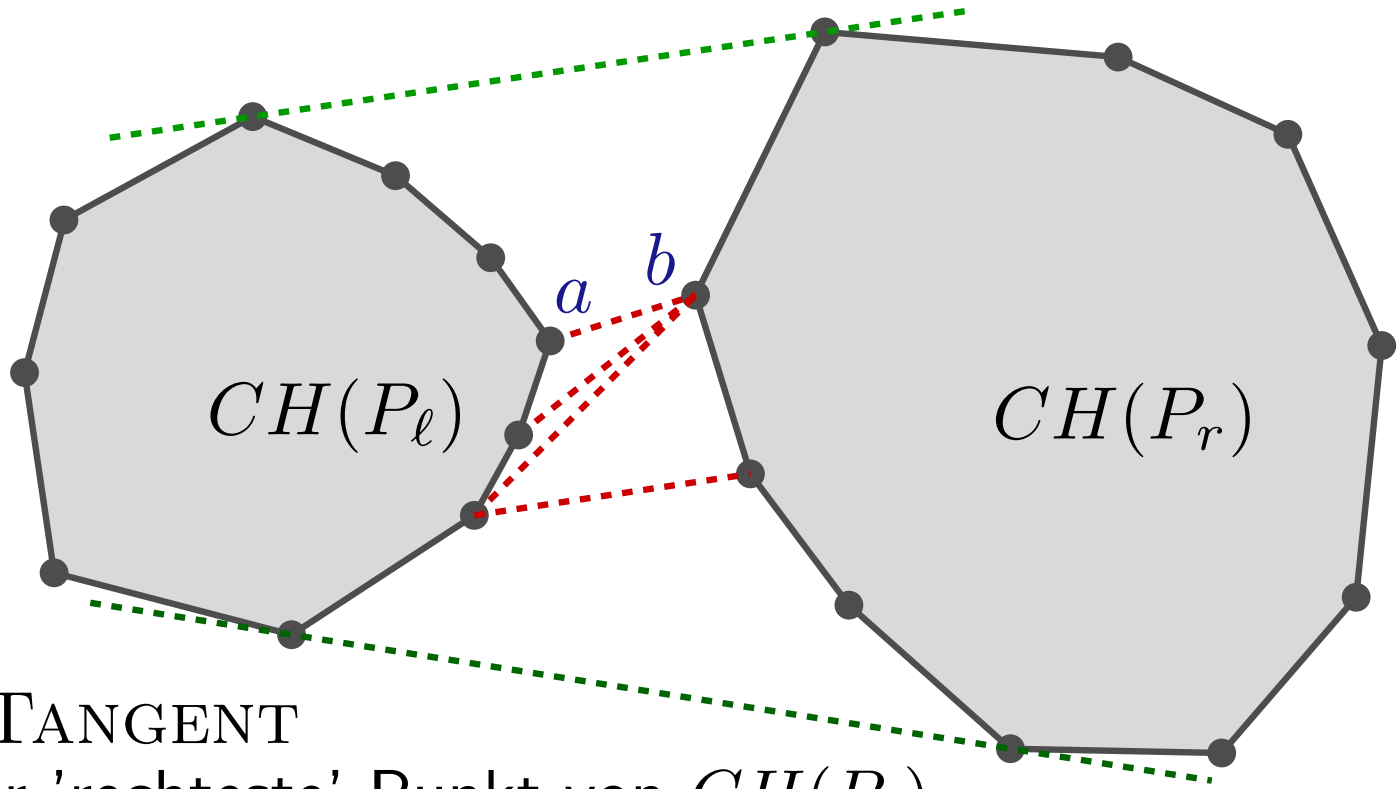
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

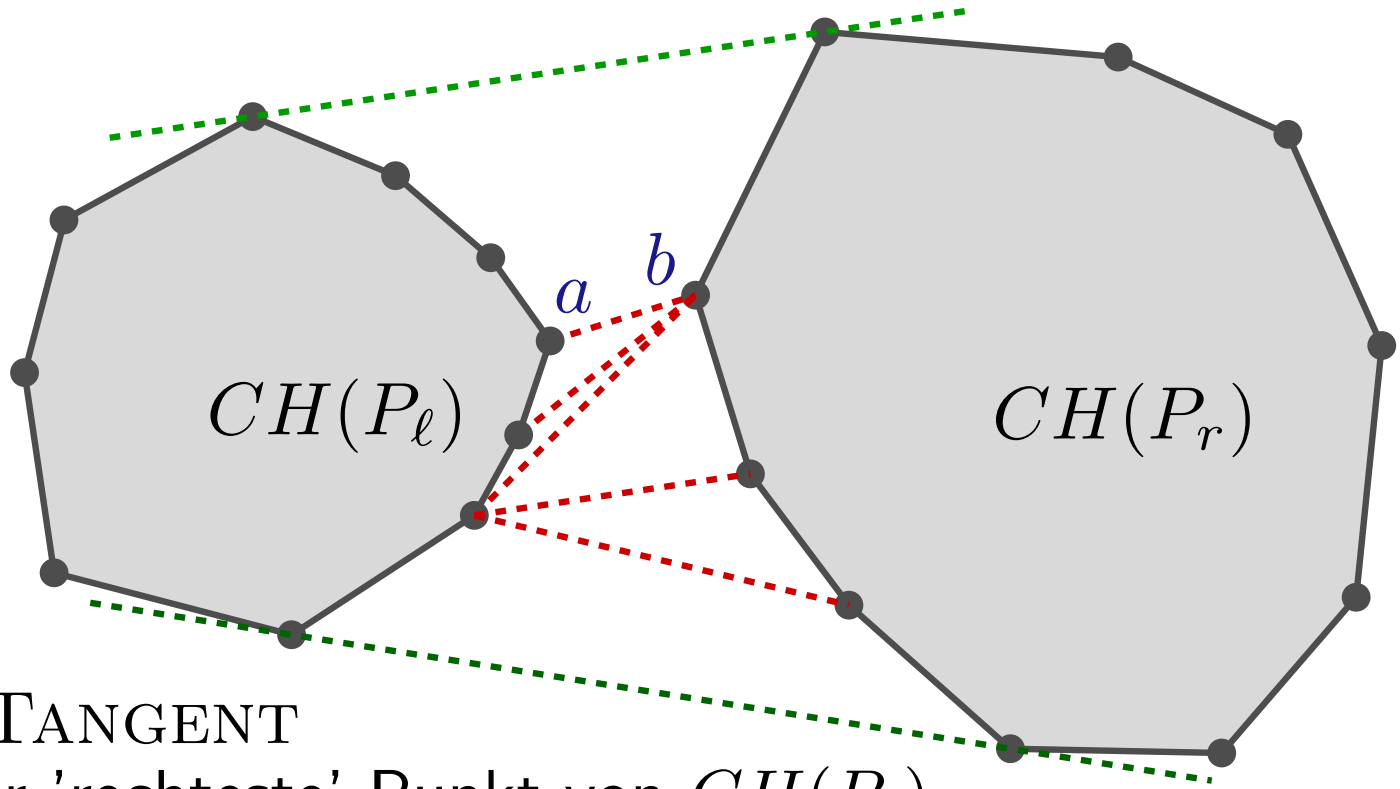
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

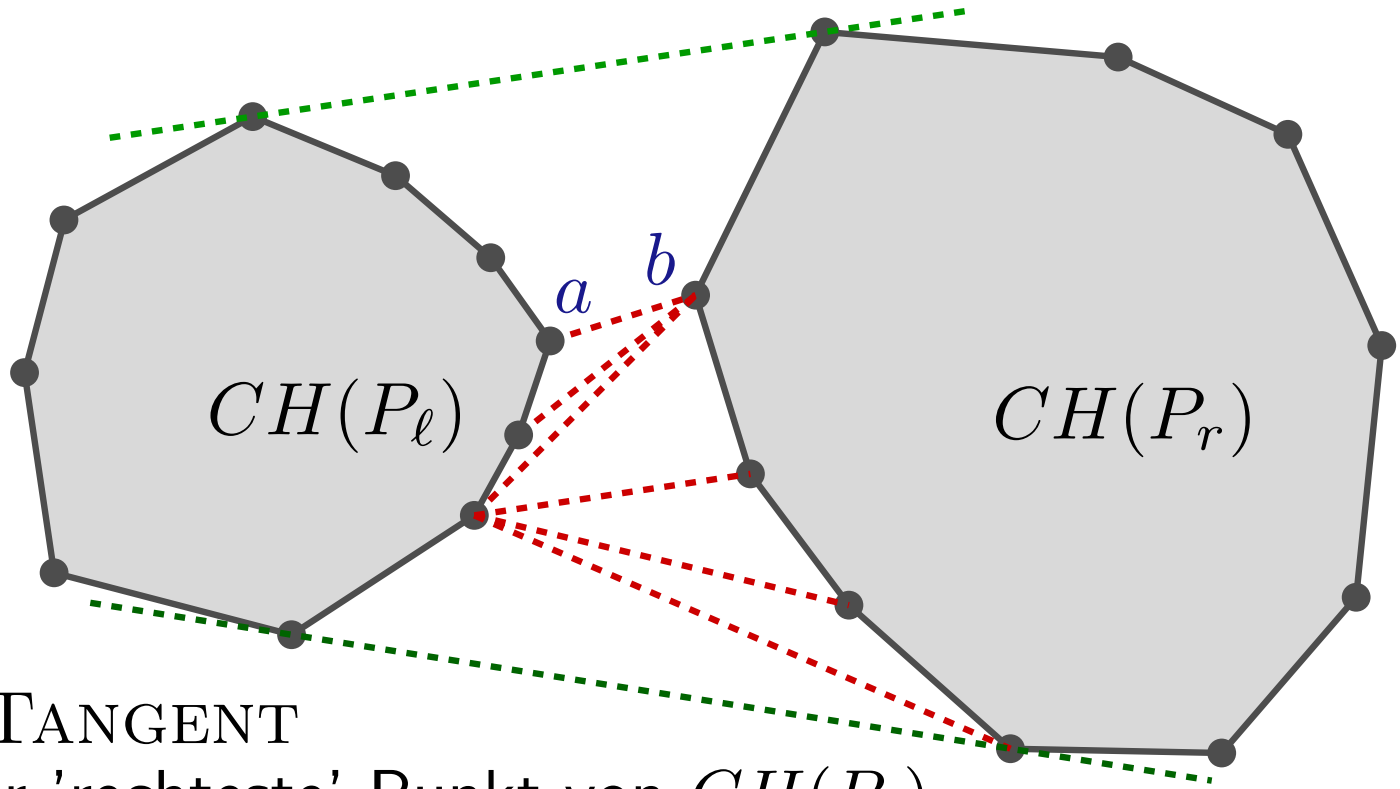
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

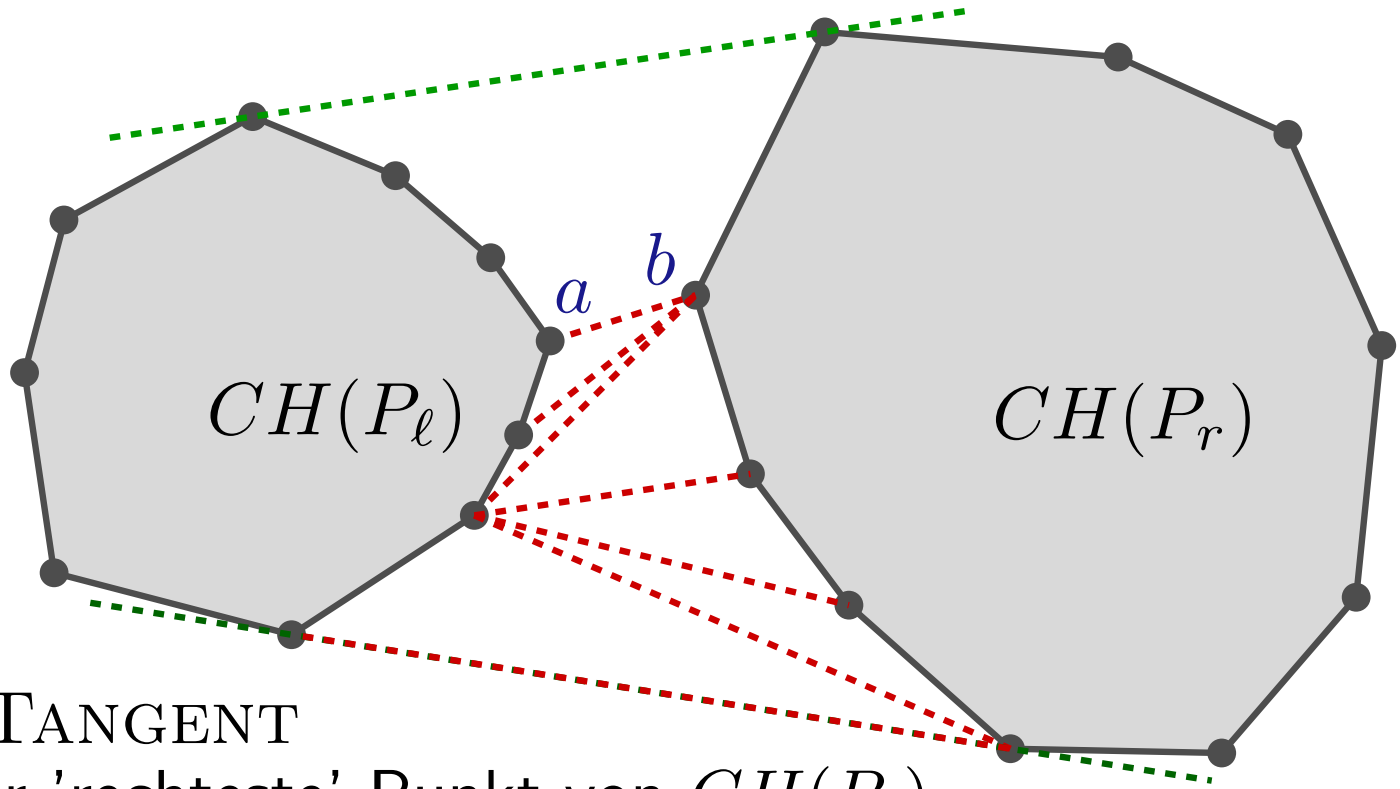
Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

- **while** ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred
- **while** ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$



LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

  while ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred

  while ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Vereinigung

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$

LOWER TANGENT

Sei a der 'rechteste' Punkt von $CH(P_\ell)$

Sei b der 'linkeste' Punkt von $CH(P_r)$

while ab nicht untere Tangente **do**

  while ab nicht untere Tangente von $CH(P_\ell)$ **do** $a = a$.pred

  while ab nicht untere Tangente von $CH(P_r)$ **do** $b = b$.succ

Laufzeit?

Karlsruhe Institute of Technology

DivideConquerConvexHull($P \subset \mathbb{R}^2$)

if $|P| \leq 3$ **then**

  return $CH(P)$

Halbiere P in linke Teilmenge P_ℓ und rechte Teilmenge P_r

$CH(P_\ell) =$ DivideConquerConvexHull(P_ℓ)

$CH(P_r) =$ DivideConquerConvexHull(P_r)

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$

return $CH(P)$

Laufzeit?

DivideConquerConvexHull($P \subset \mathbb{R}^2$)

if $|P| \leq 3$ **then**

  return $CH(P)$

Halbiere P in linke Teilmenge P_ℓ und rechte Teilmenge P_r

$CH(P_\ell) =$ DivideConquerConvexHull(P_ℓ)

$CH(P_r) =$ DivideConquerConvexHull(P_r)

$CH(P) =$ Vereinigung von $CH(P_\ell)$ und $CH(P_r)$

return $CH(P)$

$$T(n) = \begin{cases} \mathcal{O}(1) & , |P| \leq 3 \\ 2T(n/2) + \mathcal{O}(n) & , \text{sonst} \end{cases}$$

Das war's!

Karlsruhe Institute of Technology

Nächster & letzter Termin:

Donnertag, 19.07, 10:15 Uhr

Raum 131, Gebäude 50.34