Konvexe Hülle

„Vielzweckwaffe“

„Kürzester Zaun“
Approximation von Punktmengen
Simplex-Optimierung
Abstände und Kollisionen

- Eigenschaften s. weiter vorne
- Ziel: Design von Algorithmen üben,
  Verständnis der Algorithmen,
  nicht deren Nach-Implementierung (dazu gibt‘s schon zu viele und zu gute, z.B. qhull, LEDA, CGAL, …)

Konvexe Hülle (2D)

Naiver Algorithmus ???

Konvexe Hülle (2D)

Naiver Algorithmus

Teste für alle Kanten ($O(n^2)$),

ob alle anderen Punkte links der Geraden liegen ($O(n)$) :

$O(n^3)$

Und in R^n ???

Die übliche Frage: Geht das nicht besser?

Konvexe Hülle (2D)

Inkrementeller Algorithmus

Algorithmus 6.1 Inkrementelle Bestimmung der konvexen Hülle

$\textsc{ConvexHull}_{\textsc{Incremental}}(p_1, \ldots, p_n)$

Input: n Punkte $p_1, \ldots, p_n \in \mathbb{E}^2$

$CH \leftarrow (p_1, p_2, p_3)$ bzw. $CH \leftarrow (p_1, p_3, p_2)$ {die drei Punkte sollen gegen den Uhrzeigersinn orientiert sein}
for $i \leftarrow 4, \ldots, n$ **do**
    if p_i liegt nicht im Polygon CH **then**
        Bestimme die beiden Berührecken p_l bzw. p_r der Tangenten durch p_i an das Polygon CH
        Entferne alle Punkte zwischen p_l und p_r aus der Eckenliste CH
        Füge den Punkt p_i zwischen p_l und p_r in die Eckenliste CH ein
    end if
end for

Output: CH ist die konvexe Hülle der Punkte $p_1, \ldots, p_n$

Konvexe Hülle (2D)

Inkrementeller Algorithmus, Komplexität

O(n log n)

O(n)

O(log n)

max. n Punkte, da bis vor dem Ende alle Punkte zur k.H. gehören können (nicht k, wie am Ende)

Und in R^n ???

Algorithmus 6.1 Inkrementelle Bestimmung der konvexen Hülle

$\textsc{ConvexHull}_{\text{Incremental}}(p_1, \ldots, p_n)$

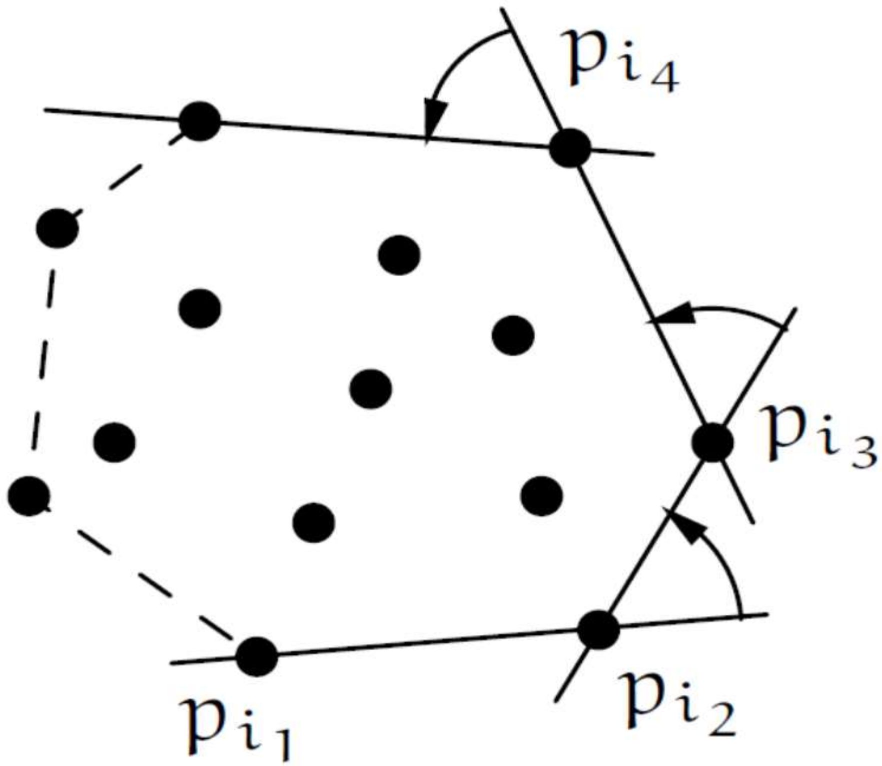
Input: n Punkte $p_1, \ldots, p_n \in \mathbb{E}^2$

```
CH ← (p1, p2, p3) bzw. CH ← (p1, p3, p2)   {die drei Punkte sollen
gegen den Uhrzeigersinn orientiert sein}
for i ← 4, . . . , n do
   if pi liegt nicht im Polygon CH then
      Bestimme die beiden Berührecken pl bzw. pr der Tangenten
      durch pi an das Polygon CH
      Entferne alle Punkte zwischen pl und pr aus der Eckenliste CH

      Füge den Punkt pi zwischen pl und pr in die Eckenliste CH ein
   end if
end for
```

Output: CH ist die konvexe Hülle der Punkte $p_1, \ldots, p_n$

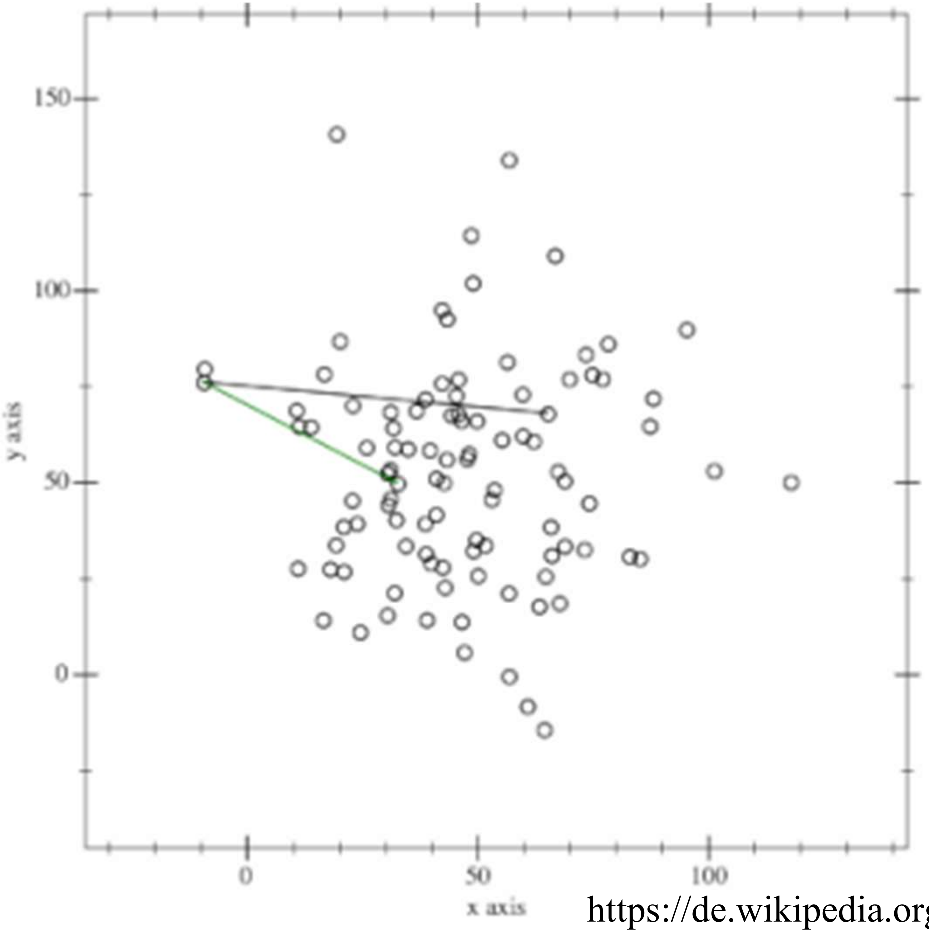
Konvexe Hülle (2D)

Jarvis March (Gift Wrap, Jarvis Wrap)



Konvexe Hülle (2D)

Jarvis March (Gift Wrap, Jarvis Wrap)



https://de.wikipedia.org/wiki/Gift-Wrapping-Algorithmus

Konvexe Hülle (2D)

Jarvis‘s March

Algorithmus 6.2 Jarvis’s march zur Bestimmung der konvexen Hülle

$\textsc{ConvexHull}_{\text{Jarvis}}(p_1, \ldots, p_n)$

Input: n Punkte $p_1, \ldots, p_n \in \mathbb{E}^2$

Bestimme den Punkt p_{i_1} als den Punkt mit minimaler y-Koordinate (bei Punkten gleicher y-Koordinate, denjenigen mit minimaler x-Koordinate)
$j \leftarrow 1$ $\{j$ zählt die Eckpunkte p_{i_j} der konvexen Hülle$\}$
repeat
  $j \leftarrow j + 1$
  $\mathit{rechts} \leftarrow 1$
  for $k \leftarrow 2, \ldots, n$ **do**
    if (p_k liegt echt rechts von $[p_{i_{j-1}}\, p_{\mathit{rechts}}]$) oder ($p_k$ liegt auf $\overline{p_{i_{j-1}}\, p_{\mathit{rechts}}}$ und näher an $p_{i_{j-1}}$ als p_{rechts}) **then**
      $\mathit{rechts} \leftarrow k$
    end if
  end for
  $p_{i_j} \leftarrow p_{\mathit{rechts}}$
until $p_{i_j} = p_{i_1}$

Output: Die konvexe Hülle der Punkte $p_1, \ldots, p_n$ ist das Polygon mit der Eckenfolge $(p_{i_1}, \ldots, p_{i_{j-1}})$

Konvexe Hülle (2D)

Jarvis‘s March, Komplexität

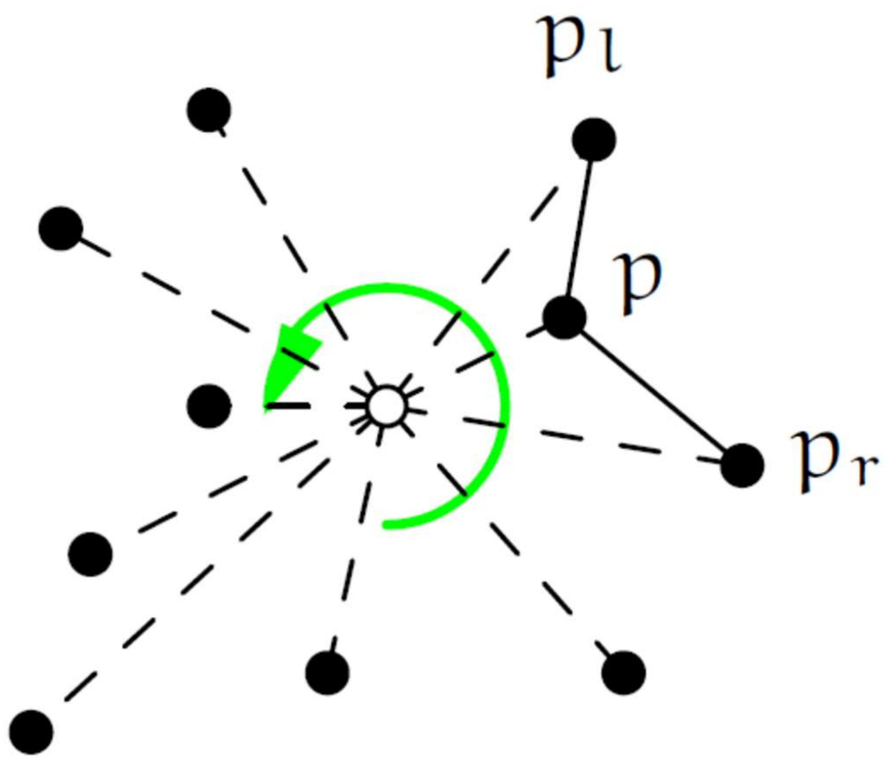
Algorithmus ist output-sensitiv: Für jede Kante der konvexen Hülle (k) alle Punkte (n) durchlaufen:

O(kn)

Und im R^n ???

Konvexe Hülle (2D)

Graham‘s Scan



Konvexe Hülle (2D)

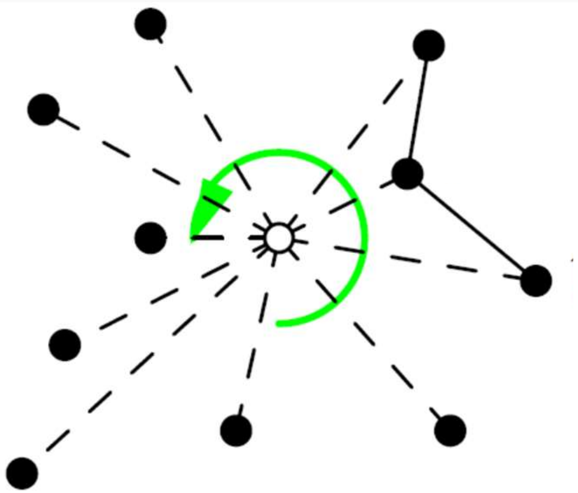
Graham‘s Scan

```
Funktion GrahamScan
   Eingabe: Punktemenge S = {P}
   Ausgabe: konvexe Hülle von S
   Beginn Funktion
      Sei S die nach dem Winkel zu P0 sortierte Punktemenge
      PUSH(P0)
      PUSH(P1)
      i := 2
      n := Anzahl der Punkte in S

      Solange i < n, führe aus:
         Sei Pt1 der oberste Punkt auf dem Stack
         Sei Pt2 der zweitoberste Punkt auf dem Stack
         Wenn Si links des Vektors Pt2→Pt1 liegt, dann führe aus:
            PUSH(Si)
            i := i + 1
         Ansonsten führe aus:
            POP(Pt1)
         Ende Bedingung

      Ende Schleife

   Ende Funktion
```

Noch zwei Anforderungen:

- P_0 gehört zur konvexen Hülle (z.B. linkester, unterster Punkt)
- $P_n = P_0$ (sonst wird Hülle nicht geschlossen)

(oder einem beliebigen Punkt innerhalb der konvexen Hülle, s. Skizze)



Konvexe Hülle (2D)

Graham‘s Scan, Komplexität

- Sortieren nach dem Winkel : O(n log n)
- Durchlaufen der sortierten Punktfolge O(n)

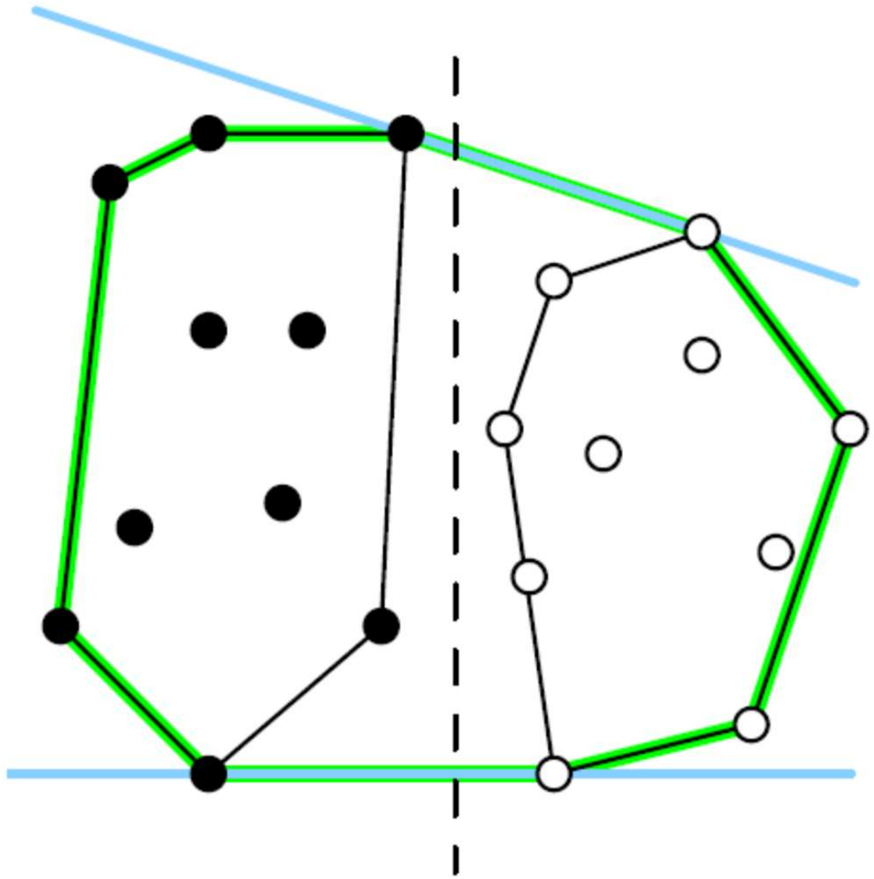
Somit: **O(n log n)**

Und in R^n ???

Konvexe Hülle (2D)

Divide and Conquer

- Aufteilen der Punktmenge in gleich große, ggf. disjunkte Mengen
- Bilden der konvexen Hülle bei hinreichend kleinen Punktzahlen
- Verschmelzen der konvexen Hüllen



Konvexe Hülle (2D)

Divide and Conquer

Algorithmus 6.3 Konvexe Hülle mittels Divide-and-conquer

$\textsc{ConvexHull}_{\text{D-A-Q}}(p_1, \ldots, p_n)$

Input: n Punkte $p_1, \ldots, p_n \in \mathbb{E}^2$

if $n \leq 6$ **then**
  $CH \leftarrow \mathsf{CH}(p_1, \ldots, p_n)$ {z. B. mittels eines naiven Algorithmus}
else
  $CH_1 \leftarrow \textsc{ConvexHull}_{\text{D-A-Q}}(p_1, \ldots, p_{\lfloor n/2 \rfloor})$
  $CH_2 \leftarrow \textsc{ConvexHull}_{\text{D-A-Q}}(p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n)$
  $CH \leftarrow \textsc{MergeHulls}(CH_1, CH_2)$
end if

Output: CH ist die konvexe Hülle der Punkte $p_1, \ldots, p_n$

Punkte entlang einer Dimension sortiert!!!

Konvexe Hülle (2D)

Divide and Conquer

Algorithmus 6.4 Konvexe Hülle zweier konvexer Polygone

MERGEHULLS(P_1, P_2)

Input: Zwei konvexe Polygone $P_1, P_2 \subset \mathbb{E}^2$

Bestimme einen Punkt $\tilde{p} \in P_1$

~~**if** $\tilde{p} \in P_2$ **then**~~

~~Verschmelze die (jeweils bez. $\tilde{p}$ angular sortierten) Eckenfolgen von P_1 und P_2 zu einer Folge CH~~

~~**else**~~

Falls sortierte Punktemenge, disjunkt

Bestimme die beiden Berührecken p_l und p_r der Tangenten durch $\tilde{p}$ an das Polygon P_2

Verschmelze die (bez. $\tilde{p}$ angular sortierten) Eckenfolge von P_1 und die Teilfolge von P_2 zwischen p_r bis p_l zu einer Folge CH

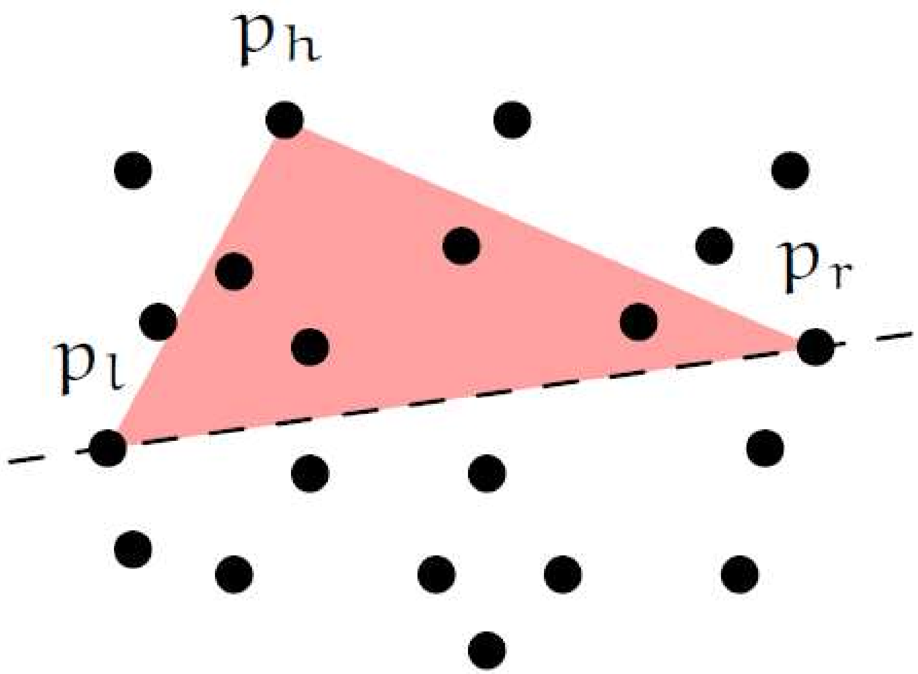
end if

Wende den Scan-Schritt des Graham's scans auf die Folge CH an und eliminiere alle konkaven Ecken aus CH

Output: CH ist die konvexe Hülle der beiden Polygone P_1 und P_2

Konvexe Hülle (2D)

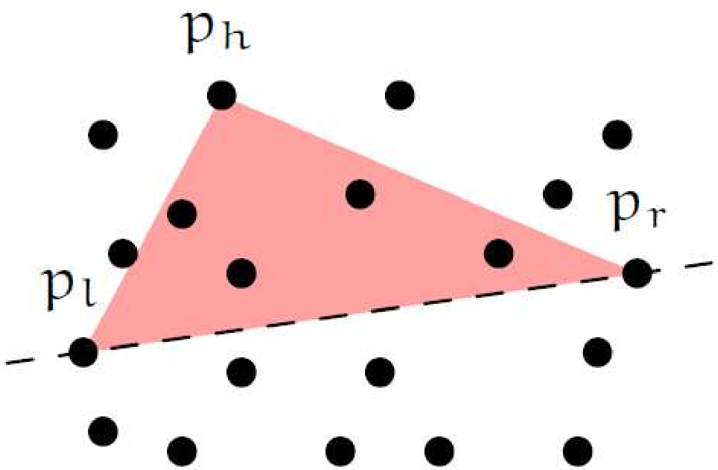
Quick Hull



http://www.cse.yorku.ca/~aaw/Hang/quick_hull/QuickHull.html

Konvexe Hülle (2D)

Quick Hull



- Wir bestimmen zunächst zwei Punkte p_l und p_r, die garantiert Eckpunkte der konvexen Hülle sind (z. B. mittels minimaler und maximaler x-Koordinate). Die Gerade $\overline{p_l\, p_r}$ zerlegt dann die Menge der Punkte in zwei Hälften.

- Zur Bestimmung der oberen Hälfte der konvexen Hülle (analog für die untere Hälfte) suchen wir einen Punkt p_h, der ebenfalls garantiert auf der konvexen Hülle liegt, z. B. den Punkt maximalen Abstands zu $\overline{p_l\, p_r}$. Alle Punkte in dem durch p_l, p_r und p_h induzierten Dreieck können jetzt entfernt werden, da sie innerhalb der konvexen Hülle liegen.

- Für die verbliebenen Punkte links von $\overline{p_l\, p_h}$ und links von $\overline{p_h\, p_r}$ wird nun rekursiv genauso verfahren, d. h. der Punkt p_h übernimmt die Rolle von p_r bzw. p_l.

Konvexe Hülle (2D)

Optimale Algorithmen

O (n log k)????

Chan‘s Algorithmus:

- Aufteilung in k Punktmengen
- Graham‘s Scan pro Punktmenge
- Jarvis March auf obigen Hüllen

Konvexe Hülle (3D)

Interessant:

- Gift Wrapping
- Quick Hull
- Divide and Conquer

Konvexe Hülle (3D)

Jarvis‘s March, Gift Wrapping

Skizze:

- Starten mit „äußerer Kante“ (z.B. Projektion nach 2D)
- Zu jeder offenen Kante: den Punkt finden, der mit der Kante ein Dreieck bildet, sodass alle Punkte links davon
- Dies für alle offenen Kanten wiederholen

Konvexe Hülle (3D)

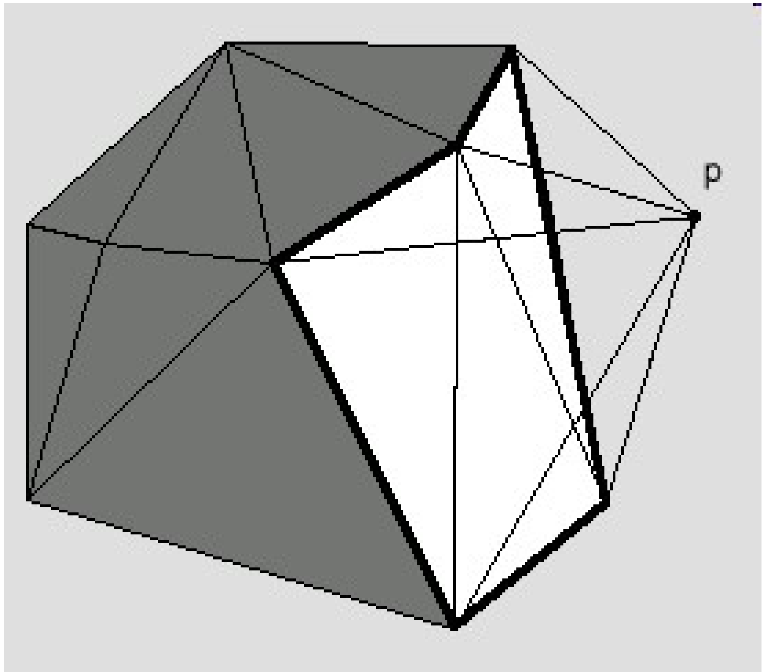
Quick Hull (auch: Inkrementell)

Wie in 2D, nur

Problem: Zu einem Punkt „Tangentendreiecke“ bilden

Kanten finden, die zwischen von Punkt sichtbaren und nicht sichtbaren Facetten liegen

Entweder: Beginnend bei dem Dreieck, dem P zugeordnet ist, mittels Topologieinfo



Konvexe Hülle (3D)

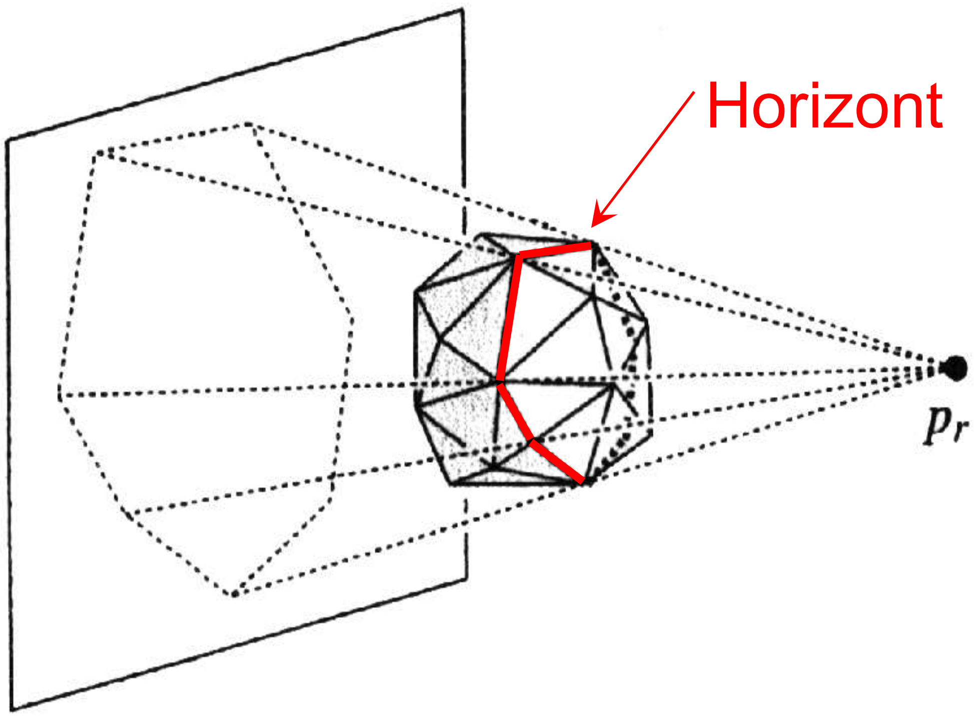
Quick Hull

Wie in 2D, nur

Problem: Zu einem Punkt „Tangentendreiecke“ bilden

Kanten finden, die zwischen von Punkt sichtbaren und nicht sichtbaren Facetten liegen

Oder: Projektion der Eck-Punkte nach 2D, bestimmen der konvexen Hülle in 2D, diese entspricht in 3D dem „Horizont“



Konvexe Hülle (3D)

Divide and Conquer

Ebenfalls wie in 2D,

Interessant: Konstruktion eines „verbindenden Zylinders“

- Untere gemeinsame Kante: Punkte der beiden Teilhüllen (analog 2D: 2 Punkte, deren Gerade keine Facette schneidet, iterativ)
- Diese zu einem Dreieck verbinden mit geeigneter Kante einer der beiden Hüllen
- Abwechselnd von dort aus weitermachen mit jeweils einem Punkt der einen oder anderen Hülle

Konvexe Hülle (2D)

Inkrementeller Algorithmus

Zum Verständnis:

`http://www.cse.unsw.edu.au/~lambert/java/3d/hull.html`

(Wunderbare, uralte Seite!!!!)

`(mittlerweile als Kopie auf:`
`http://www.cp.eng.chula.ac.th/~somchai/2110427/2542/Resources/Java-Animations/Lambert/hull.html)`