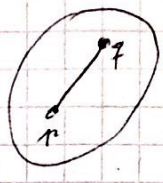
Konvexe Hülle

Definition (Erinnerung):

$K \subseteq \mathbb{R}^d$ „konvex" $:\Leftrightarrow$

$\forall\, p, q \in K: \quad \overline{pq} \subseteq K$



Def.:

Sei $S \subseteq \mathbb{R}^d$.

Die konvexe Hülle $CH(S)$ ist

$$CH(S) := \bigcap_{\substack{K \supseteq S \\ K \text{ konvex}}} K$$

(Bem.: diese Def. ist die mathem. schreibweise für diese (umgangssprachliche) Def.)

d.h., $CH(S)$ ist die kleinste konvexe Obermenge von S.

Aus Def. $\Rightarrow$ Schnitt zweier CH's ist wieder konvex.
Seien K_1, K_2 konvex $\Rightarrow K_1 \cap K_2$ ist konvex.

Im Folgenden betrachten wir nur endliche S, also Mengen von (diskreten) Punkten.

[Preparata/Shamos 4.91; Rolf Klein, Kap. 4.1]

Satz:

Die konvexe Hülle einer endlichen Menge S von Punkten im $\mathbb{R}^d$ ist ein (konvexes) Polyeder, dessen Ecken aus S sind.

Achtung: inkonsistente Verwendung der Begriffe Polyeder / Polytop in den verschiedenen Bereichen!

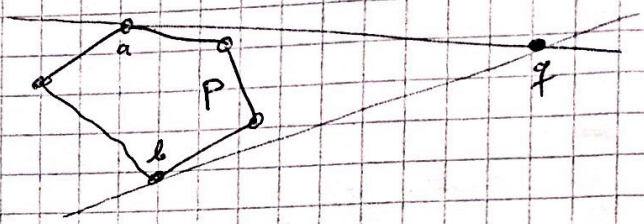
Beweis kommt gleich

il: show that line can intersect K at most in one contiguous interval
(sw. durch Widerspruch

Definition:

Gegeben ein Polygon P und ein Punkt q außerhalb P.
Eine "Tangente an P durch q" ist eine Gerade,
die P in 1 Pkt berührt und durch q geht.

(also eine supporting line)
"line through q supporting P"



Beobachtung:

Zu konvexem P gibt es
genau 2 Tangenten durch q.

Algo zur effizienten Berechnung kommt später.

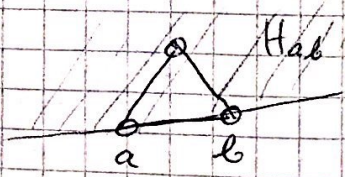
Beweis des o.g. Satzes: Induktion über $n = |S|$

Ind. anfang:

$S = \{a, b, c\} \Rightarrow$ Beh.: $CH(S) = \Delta abc$

Bew.: $\Delta abc = H_{ab} \cap H_{bc} \cap H_{ac}$

mit H_{ab} = von a, b induzierter Halbraum,
der Pkt c enthält.



Δabc ist konvex, da Schnitt zweier konvexer Mengen wieder konvex.

Δabc enthält P;

ist kleinste solche Menge: jede andere konvexe(!) Menge könnte man durch "Abschneiden" mittels einer der Halbebenen verkleinern.

Ind. schritt: $|P| > 3$

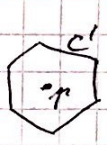
wähle irgend ein $p \in P$, setze $P' := P \setminus \{p\}$;

setze $C' = CH(P')$; für diese gilt Satz, d.h.

$\{\text{Ecken von } C'\} \in P' \subseteq P$.

Idee: konstruiere aus C' und p die $CH(P)$

Fall 1: $p \in C'$



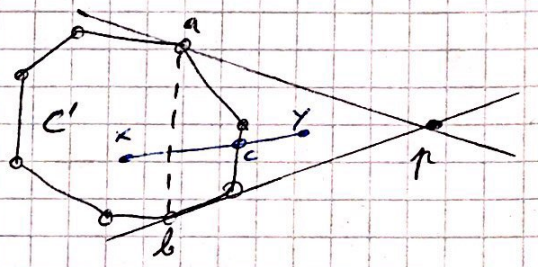
$\Rightarrow$ nichts zu tun, Satz gilt

Fall 2: $p \notin C'$

bestimme die beiden

Tangenten an C' durch p

$\rightarrow$ Pkte a, b (Stützgeraden touching at a, b)



Behauptung: $CH(P) = C' \cup \Delta abp$.

Klar ist: $\forall$ konvexe Mengen $K \supseteq P$: $C' \subseteq K \wedge \Delta abp \subseteq K$

$\Rightarrow C' \cup \Delta abp \subseteq CH(P)$

Noch zu zeigen: $C' \cup \Delta abp$ ist konvex

Es genügt zu zeigen, daß für $x \in C'$ und $y \in \Delta abp$ die Strecke $\overline{xy}$ in der Vereinigung liegt, weil

($\forall x \in C', y \in \Delta abp: \overline{xy} \subseteq C' \cup \Delta abp$)

C' und Δabp für sich schon konvex sind (d.h., falls x, y beide in C' oder Δabp, ist nichts zu zeigen).

Sei $x \in C'$, $y \in \Delta abp \setminus C'$,

setze $c :=$ Schnittpkt v. $\overline{xy}$ und Rand von C'

c muß zwischen den beiden Tangenten liegen (sonst wären es nicht die Tangenten

$\Rightarrow \overline{cy} \subseteq \Delta abp$

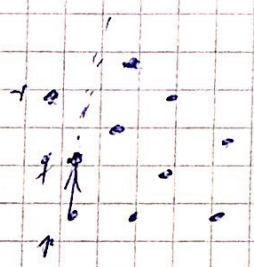
$\overline{xc} \subseteq C'$ sowieso klar $\Rightarrow \overline{xy} \subseteq C' \cup \Delta abp$.

(Man könnte den Satz auch ganz anders beweisen, z.B. durch Induktion über Schnitte von Halbebenen; aber dieser Bew. hier führt unmittelbar auf einen netten Algo!)

Berechnung der CH in 2D

Wir wissen: die CH(P) besteht aus einigen Pkten von P und den Strecken dazwischen.

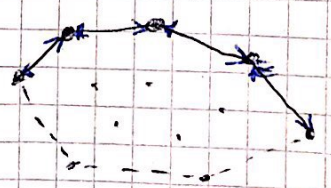
Naiver Algo:
```
C := ∅   // Menge der Kanten von CH(P)
forall (p,q) ∈ P×P, p≠q:
    forall r ∈ P \ {p,q}:
        if r liegt links von pq:
            verwerfe (p,q)
    füge (p,q) zu C hinzu
sortiere C in CCW
```



Laufzeit: $O(n^3)$

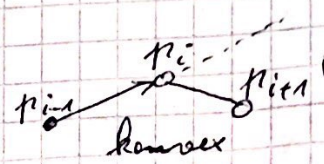
Idee: wende Algorithmentechnik "inkrementelle Berechnung" an
Hier: Punkte nach und nach zu bis dahin bestehenden CH hinzufügen (die natürlich nur für die bis dahin "gesehenen" Pkte gültig sein kann

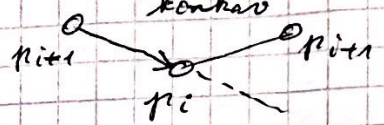
Definition
Die "obere Hülle" $\widehat{CH}(P)$:= alle diejenigen Pkte aus CH(P), die von ~~max.~~ min. nach ~~min.~~ max. x-Koord. laufen, wenn man CH(P) in ~~CCW~~ CW-Order ausgibt.
analog "untere Hülle".



Def.:

Ein Vertex p_i eines Polygonzugs P heißt "konkav" :⇔ p_{i+1} liegt links von $\overline{p_{i-1} p_i}$. ⇔ $\Delta(\Delta p_{i-1} p_i p_{i+1}) > 0$





Achtung: diese Def hängt davon ab, in welcher Reihenfolge der kommende Algo vorgeht – würde er z.B. die obere Hülle von rechts nach links berechnen, müßte man es genau umgekehrt definieren!

Algo: Graham's Scan

Input: P = Ecke in der Ebene

sortiere P bzgl. x-Koord → $p_1, \dots, p_n$

init $\hat{C} := \{p_1, p_2\}$ // working set für $\hat{CH}$

for $i = 3 \dots n$:

$\hat{C} \mathrel{+}= \{p_i\}$ // p_i zu $\hat{C}$ hinzufügen

while $|\hat{C}| > 2 \wedge$ // repariere $\hat{C}$ ggf.

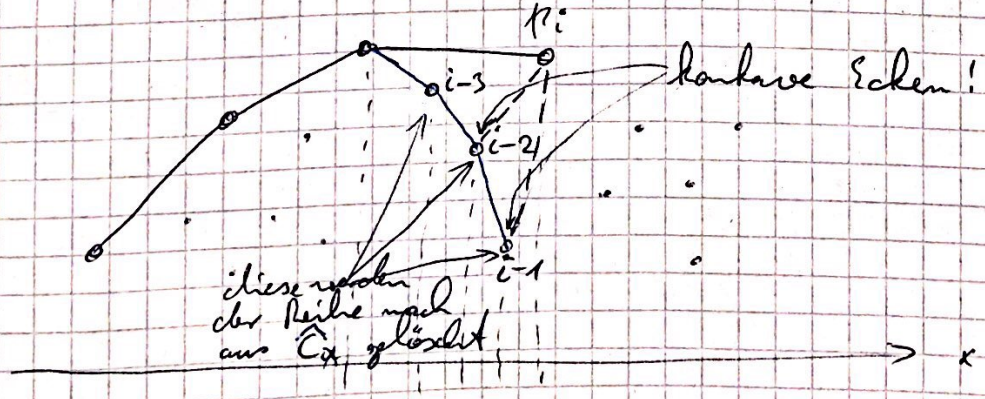
$(p_{i-2}, p_{i-1}, p_i) \in \hat{C}$ bilden konkave Ecke:

lösche p_{i-1} aus $\hat{C}$

berechne analog die untere Hülle $\check{C}$

Dieses Tripel soll immer die letzten 3 Ecke aus dem aktuellen $\hat{C}$ sein: ist formal nicht korrekt, so wie es gesch. ist, aber hoffentl. doch klar

Bsp.:



Laufzeit: $O(n \log n)$

Bew.: jeder Pkt kann nur 1x aus $\hat{C}_u$ gelöscht werden $\Rightarrow$ innere while-Schleife kann max. nx insgesamt durchlaufen werden $\Rightarrow$ nach dem Sortieren braucht der Rest $O(n)$ Zeit; das ganze nochmal für C_l $\Rightarrow$ Beh.

Fazit: Graham's Scan hat optimale worst-case Laufzeit.

Erste einfache Anwendung:

Geg. drei Gase G_1, G_2, G_3;
jedes besteht zu x_i Prozent aus Stickstoff, zu y_i Prozent aus Sauerstoff.

Frage: Kann man G_1, G_2, G_3 so mischen, daß ein Gas $G=(x,y)$ entsteht? [das also genau x Prozent Stickstoff und y Prozent Sauerstoff enthält?]

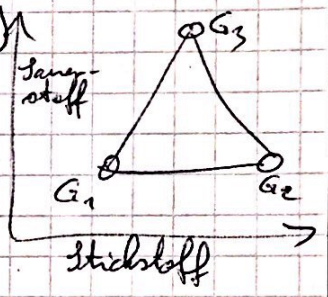
Lsg.: "Mischen" bedeutet

$$G = \sum \lambda_i G_i \quad , \quad \lambda_i \geq 0 \quad , \quad \sum \lambda_i = 1$$

Da uns nur Mischungsverhältnisse interessieren $\rightarrow \sum \lambda_i = 1$

Antwort: "Ja" $\Leftrightarrow$ $G \in \mathcal{CH}(G_1, G_2, G_3)$

$\Leftrightarrow \exists \lambda_i : \lambda_i \geq 0 \wedge \sum \lambda_i = 1$

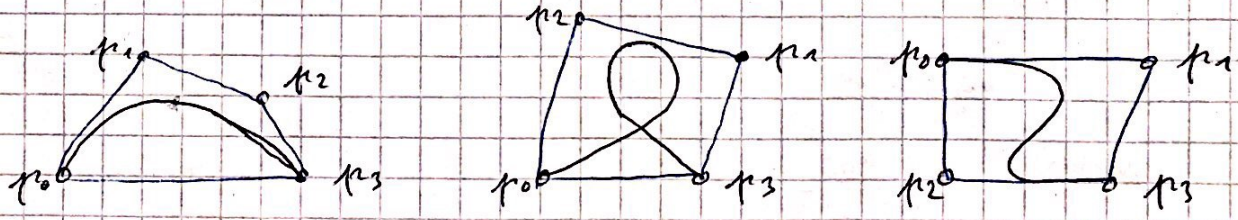


analog mit Farben (in 3D)

Andere einfache Anwendung:

Die "convex hull property" von Bézier-Kurven:
eine Bézier-Kurve, aufgespannt durch Kontrollpkte $P = \{p_1, \dots, p_k\}$, liegt immer vollständig in $\mathcal{CH}(P)$.

Bsp.:



Kann man z.B. ausnutzen, um Schnittberechnungen zwischen Bézier-Kurven zu beschleunigen.

Def.: $$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i p_i \quad , \quad 0 \leq t \leq 1$$

Problemstellung präzisiert:

Gegeben n Pkte in der Ebene;

bestimme das konvexe Polygon $CH(P)$ durch

Ausgabe der Eckpkte entgegen dem Uhrzeigersinn.

Jetzt erst: zunächst einfache Anwendungen, dann Eigenschaften
den 2D-Algo, dann evtl. Prädikate, dann Eigenschaften der CH,
dann Beweis mit Reduktion auf Sortieren

Oder: 1. CH in 2D 1. Einfache Anwendungen, 2. CH in 2D, 3. Untere Schranke (Reduktion auf Sortieren), 4. Zusammenhang, 5. Konvexe Kombination, 6. Geom. Prädikate 7. Geom. robustness

Def.: Gegeben k Pkte $p_1, \dots, p_k \in \mathbb{R}^d$. ($k \le d+1$, und $k > d+1$ OK)

Eigenschaften

Eine "konvexe Kombination" dieser Pkte ist

$$p = \sum_{i=1}^{k} \lambda_i p_i \quad , \quad \sum \lambda_i = 1 \; , \quad \forall \lambda_i \ge 0 \; .$$

Satz (o. Bew.):

Geg. $P = \{p_1, \dots, p_k\} \in \mathbb{R}^d$.

$$CH(P) = \{ \underbrace{\sum_{i=1}^{k} \lambda_i p_i \mid \sum \lambda_i = 1 \; , \; \lambda_i \ge 0}_{= \text{konvexe Kombination}} \} .$$

Die λ_i heißen "barycentrische Koord."

Proof: via induction

start with p_1, p_2: $\forall \lambda \in [0,1] : \lambda p_1 + (1-\lambda) p_2 \in \overline{p_1 p_2}$ $\quad = CH(p_1, p_2)$

$\Leftrightarrow \quad \lambda_1 p_1 + \lambda_2 p_2 \in \overline{p_1 p_2}$, with $\lambda_1 + \lambda_2 = 1$, $\lambda_1, \lambda_2 \ge 0$

Noch ein paar Eigenschaften der konvexen Hülle

Satz:

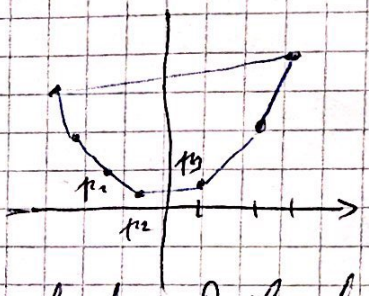
Die Konstruktion der konvexen Hülle von n Punkten in der Ebene erfordert $\Omega(n \log n)$ Zeit.

Bew.:

Wir zeigen: Sortieren kann man in linearer Zeit $O(n)$ auf die konvexe Hülle reduzieren.

Geg.: $x_1, \dots, x_n \in \mathbb{R}$

Konstruiere $P = \{ (x_i, y_i) \mid y_i = x_i^2 \}$



Klar ist: alle p_i erscheinen in $\mathcal{CH}(P)$

Ein Algo für $\mathcal{CH}(P)$ liefert die x_i in sortierter Reihenfolge (modulo eines Index-Shifts)

Satz: ("Gummiband-Satz")

Der Rand der konvexen Hülle $\mathcal{CH}(P)$ einer Menge Punkte P in der Ebene ist der kürzeste geschlossene, einfache Weg um P herum.

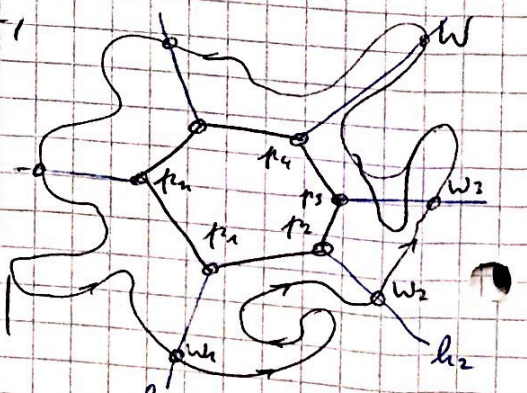
(Erinnerung: Gummiband)

Bew.:

Sei W ein Weg um P herum. (und einfach)

Zu zeigen: W ist mindestens so lang wie $\partial \mathcal{CH}(P)$.

Schicke Strahlen h_i von p_i nach außen, die Winkelhalbierende sind; laufe auf W ab herum und bezeichne mit $w_i :=$ erster Schnittpunkt von W mit h_i



Nun ist $\| w_{i+1} - p_i \| \leq \| w_{i+1} - w_i \| \leq | w_{i,i+1} |$

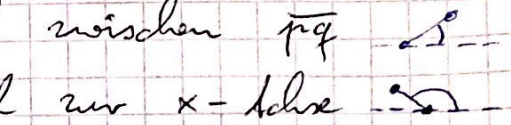
↑ Wegstück auf W von w_i bis w_{i+1}

[Preparata/Shamos S.105
Cormen et al., Kap. 33.3, S. 955]

Algo "Jarvis' March":

Beobachtung: wenn $\overline{pq}$ Kante der $CH(P)$ ist, dann muß es einen Pkt $r \in P$ geben, so daß $\overline{qr}$ die nächste Kante ist. Falls $\overline{pq} \in CH(P) \Rightarrow \exists r \in P: \overline{qr} \in CH(P)$

Bezeichnung: $\sphericalangle(\overline{pq}, x)$ = Winkel zwischen $\overline{pq}$ und der Geraden durch p parallel zur x-Achse



Konstruktion der rechten Hülle:

p_0 := kleinster Pkt aus P bzgl. lexikographischer Sortierung entlang y (!) (1)

while $p_i <$ größter Pkt bzgl. Sortierung:

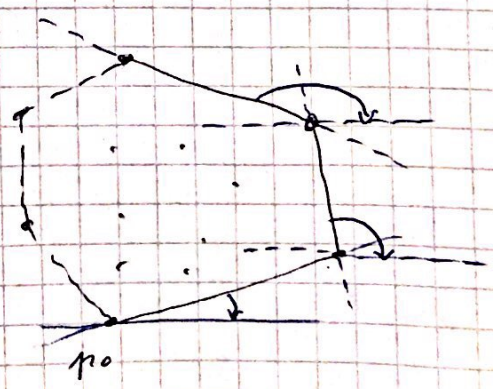
lösche p_i aus P, output p_i

p_{i+1} := Pkt aus P mit min. $\sphericalangle(\overline{p_i p_{i+1}}, x)$ (2)

Frage: Implementierung von (2) ist klar?
(einfach 1x über P laufen)

Laufzeit:
Sei h = Anzahl Ecken auf $CH(P) = |CH(P)|$
→ while wird h Mal durchlaufen;
Schritt (2) hat Aufwand $O(n)$, dito Schritt (1)
$\Rightarrow O(n \cdot h) \leq O(n^2)$

Falls $h \ll \log n$ dann ist also Jarvis' March schneller als Graham's Scan.



Bemerkung: Jarvis' March verwendet die "gift-wrapping"-Technik; diese läßt sich auch auf höhere Dim. übertragen.

Definition:
Ein Algorithmus ist "output-sensitive", wenn seine Komplexität von der Größe der Ausgabe (mit) abhängt, d.h., $T_A \in O(f(n,h))$ mit h = Größe der Ausgabe.

(Die triviale Ebt $f(n,h) = g(n) + h$ ist hier natürlich nicht gemeint! Die Ausgabe ausgeben braucht immer $O(h)$.)

Jarvis' March ist also 'output-sensitive.

Nachteil: falls $h \in O(n) \Rightarrow T_{Jarvis} \in O(n^2)$!

Frage: gibt es einen output-sensitive Algo, der auch im worst-case nicht schlechter als $O(n \log n)$ ist?

[Timothy Chan:
Optimal output-sensitive convex hull algorithms...
Discrete Comp Geom 1996]

Idee: die Wrapping-Steps (2) beschleunigen durch Preprocessing

Ansatz:
1. Partitioniere P, berechne CHs für die Teilmengen, 2. mache dann Wrapping über eine Menge von CHs; die Hoffnung dabei ist, daß das Wrapping so viel schneller geht.
3. Zusätzlicher Trick: finde die "richtige" Anzahl Teilmengen durch geschicktes "Raten".

Im folgenden berechnen wir – wie bei Jarvis' March – nur die "rechte Hülle"; die linke Hülle geht dann analog.

Algo HullG(P, n, h'): "G" für "h being guessed"

input: $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$

$3 \le m \le n$, $h' \ge 1$ // h' = geratener Wert für h

setze $k = \lceil \frac{n}{m} \rceil$

partitioniere $P = P_1 \dot\cup \dots \dot\cup P_k$, $k = \lceil \frac{n}{m} \rceil$

for $i = 1 \dots k$:

berechne $C_i := CH(P_i)$ // C_i enthält Vertices in CCW
// die C_i können überlappen!
// lexikograph. Sortierung

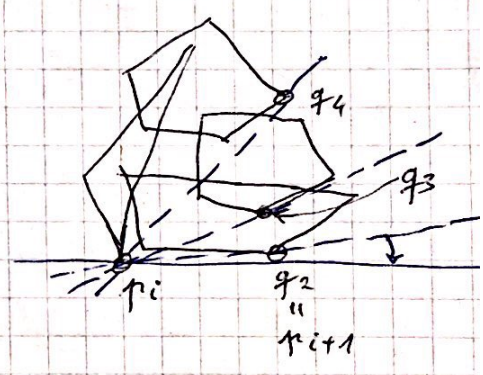
$p_1 :=$ tiefster Pkt aus P bzgl. y

for $l = 1 \dots h'$:

for $i = 1 \dots k$: (1)

berechne die "untere" Tangente an C_i durch p_i:

$q_i :=$ der Pkt aus P_i / C_i auf dieser Tangente



setze $p_{i+1} :=$ der Pkt aus $\{q_1, \dots, q_k\}$, der $\sphericalangle(\overline{p_i q}, x)$ minimiert

if p_{i+1} = oberster Pkt aus P: (*)

return Flag "Hülle ist vollständig" und $(p_1, \dots, p_l)$

end for l

return "Hülle ist unvollständig" (**)

Falls $h' \ge h$, dann muß der Fall (*) irgendwann eintreten; sonst wird (**) am Ende ausgeführt.

Laufzeit T_{HullG}:

Klar ist: $\forall i: |P_i| \leq m \Rightarrow$ jedes C_i kostet $O(m \log m)$ Zeit (z.B. mit Graham's Scan) $\Rightarrow$ Preprocessing kostet $O(k \cdot m \log m) = O(n \log m)$. ($k = n/m$)

Jede Tangente kostet $O(\log m)$ Zeit $\Rightarrow$

Schleife (1) kostet $O(k \log m) \Rightarrow$

Gesamtlaufzeit ist $O(h' \frac{n}{m} \log m + n \log m)$.

Wähle $m := h' \Rightarrow T_{HullG}(n, h') \in O(n \log h')$.

Frage: wie kann man h' so finden, daß $h' \geq h$ und die Gesamtlaufzeit nicht wesentlich größer wird?

Idee: exponentielle Suche

```
Algo Hull(P):
    for t = 1, 2, ... :
        m := h' := 2^(2^t)
        C = HullG(P, m, h')
        if "C ist vollständig":
            return C                    // wir haben h' ≥ h erreicht
// Schleife muß stoppen, sobald nämlich 2^(2^t) = h' ≥ h
```

Laufzeit T_{Hull}:

Schleifendurchläufe $= \lceil \log \log h \rceil$

t-te Iteration benötigt Zeit $O(n \log h') = O(n \cdot 2^t)$

Gesamtlaufzeit ist

$$\sum_{t=1}^{\lceil \log \log h \rceil} O(n \cdot 2^t) = O\left(n \cdot \sum_{t=1}^{\log \log h} 2^t\right) = O\left(n \cdot 2^{\log \log h + 1}\right) = O(n \log h).$$

Konvexe Hülle in 3D

Frage: was ist die Komplexität der CH in 3D?

Bezeichnung: "Komplexität" einer geometrischen Datenstruktur $= f(n)$, wobei n = Größe der Eingabe (z.B. Anzahl Pkte) und $f(n)$ = Größe der Ausgabe (z.B. Anzahl Pkte + "kombinatorische Struktur"; in diesem Fall also Beschr. der Polygone/Facetten der CH).

Frage: wieso ist Komplexität der CH in 3D plötzlich nicht mehr so klar?
Weil jetzt ein Pkt zu ganz vielen Facetten gehören kann!

Satz (Euler-Formel):

Sei P ein konvexes Polyeder, $v = |V_P|$, $e = |E_P|$, $f = |F_P|$. Dann gilt

$$v - e + f = 2.$$

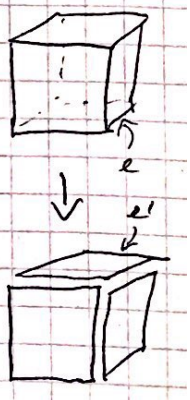
Bew.: siehe CG1 (Randrepräsentationen)

Satz:

In jedem konvexen Polyeder ist $v, e, f \in \Theta(v, e, f)$.
(D.h., $e \in \Theta(v)$, $v \in \Theta(f)$, etc.)

Bew.:

Ersetze jede Kante in E durch 2 Kanten $\Rightarrow E'$



Jede Kante in E' gehört zu genau 1 Facette,
jede Facette hat mind. 3 Kanten, also

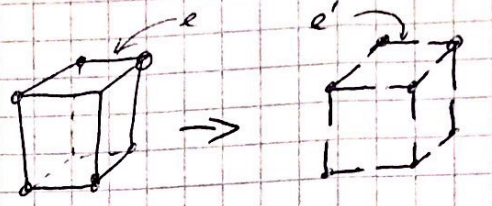
$$3f \le e' = 2e \quad \Rightarrow \quad \underline{f \le \tfrac{2}{3} e}$$

Einsetzen in Euler-Formel:

$$2 = v - e + f \le v - e + \tfrac{2}{3} e \quad \Rightarrow \quad v - \tfrac{1}{3} e \ge 2$$

$$\Rightarrow \quad \underline{e \le 3v - 6} \qquad \Rightarrow \tfrac{3}{2} f \le e \le 3v - 6$$

Analog über Vertices: jeder Vertex ist zu mind. 3 Kanten inzident $\Rightarrow$ $v \leq \frac{2}{3} e$, $e \leq 3f - 6$



Zusammen:

$$\frac{3}{2} v \leq e \leq 3v - 6$$

$$\frac{3}{2} f \leq e \leq 3f - 6$$

$$\frac{1}{2} f + 3 \leq v \leq 2f - 4$$

...

Corollar:

Die Komplexität der CH in 3D ist $O(n)$.

Bemerkungen:

1. Diese Komplexitätsaussagen gelten genauso für einfache, geschlossene (nicht-konvexe) Polyeder, und planare Graphen.
2. In d Dimensionen gibt $f \in \Theta(n^{\lfloor d/2 \rfloor})$!

   ?(Es kann also in höheren Dim. höchstens output-sensitive sensible Algos geben, die sub-exponentielle Laufzeit haben!)

   $\Rightarrow$ in higher dimensions, we cannot hope to find gb algos with time complexity less than $O(n^{d/2})$!

   $\rightarrow$ we can only hope for better output-sensitive complexities

Der Clarkson-Shor-Algo: [1989]
[Man-gelbes Buch]

Arbeitet inkrementell und randomisiert.

Idee des Algos: wie in dem Beweis des ersten Satzes, der besagt, daß die CH ein konvexes Polyeder ist.

Darstellung: mittels DCEL (s. CG1)

Start: suche 4 Pkte aus P, die nicht in einer Ebene liegen; starte mit p_1, p_2; suche p_3, das nicht auf der Geraden durch $\overline{p_1 p_2}$ liegt; suche p_4, das nicht in der Ebene durch $\Delta p_1 p_2 p_3$ liegt.

(Falls das nicht geht, liegen alle Pkt in einer Ebene ⇒ berechne CH mit 2D-Algo.)

Permutiere $p_5, \dots, p_n$ zufällig.

Schleife über $p_r := p_5 \dots p_n$:

berechne $C_r := CH(p_1, \dots, p_r)$

aus p_r und $C_{r-1} = CH(p_1, \dots, p_{r-1})$

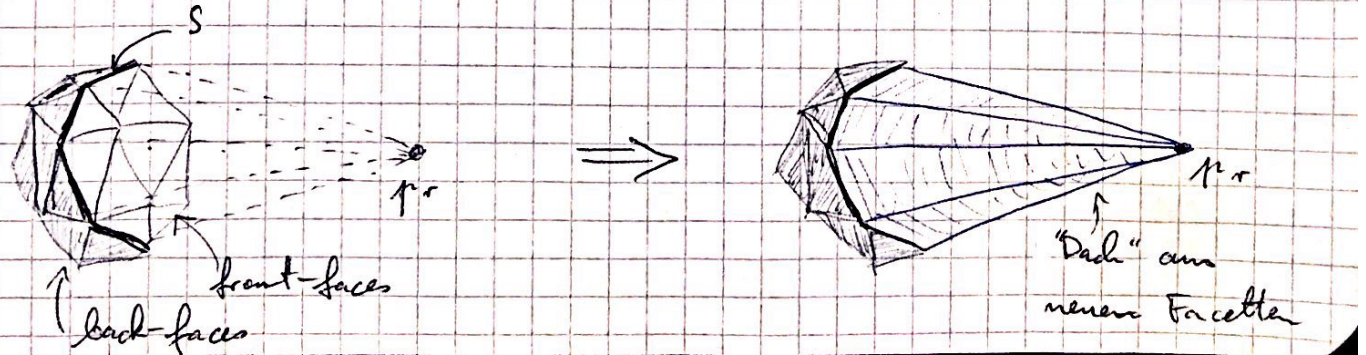
1. Fall: $p_r \in C_{r-1}$ ⇒ nichts zu tun, verwerfe p_r

2. Fall: $p_r \notin C_{r-1}$ ⇒

Frage: wie testet man das? sieht man gleich

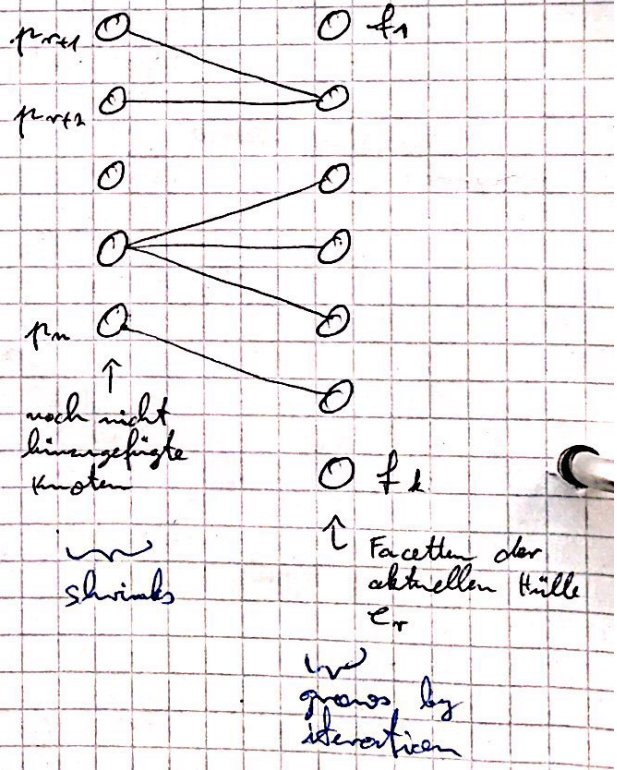
front- / back-faces aus C_{r-1} bzgl. p_r bilden je eine zus. hängende Region; die Kanten dazwischen bilden eine zus. hängende Kurve ⇒ "Silhouette" S

Lösche alle front-faces, schließe Polyeder wieder durch neue Dreiecke zwischen p_r und je einer Silhouettenkante.



Wie findet man schnell alle front-faces?
(Brute-force: alle testen → $O(n^2)$ – Algo.)

Idee: der "Konflikt-Graph"
Besteht aus folgenden "Konfliktmengen":
Für jede Facette $f \in \mathcal{C}_r$: $P_{conf}(f) \subseteq \{p_{r+1}, \dots, p_n\}$ = alle Pkte, die f "sehen" können (bzgl. aller also f ein front-face ist);
für jeden Pkt p_s, $s > r$: $F_{conf}(p_s) \subseteq F(\mathcal{C}_r)$ = alle Facetten, die p_s "sehen" kann.
Sprechweise: $p \in P_{conf}(f)$ "ist in Konflikt mit Facette f"
(beide können nicht zusammen in der CH existieren).
Datenstruktur: ein bipartiter Graph $\mathcal{G}$



Verwendung von $\mathcal{G}$:
Beim Einfügen von p_r,
lösche $F_{conf}(p_r)$ aus $\mathcal{C}_{r-1}$;
teste bei jedem $f \in F_{conf}(p_r)$,
ob eine der inzidenten Kanten zu p_r auf dem aus $S \to$ Dreieck hinzieht.
Aufwand: $O(|F_{conf}|)$

Aktualisierung von $\mathcal{G}$:
Initialisierung: alle Pkte $p_5, \dots, p_n$ gegen $\mathcal{C}_4$ testen.

Beim Einfügen von p_r: 1. alle Nachbarn von p_r in $\mathcal{G}$ löschen ($= F_{conf}(p_r)$)
2. p_r löschen aus $\mathcal{G}$
3. neue Knoten für neue Facetten erzeugen in $\mathcal{G}$
4. Kanten in $\mathcal{G}$ für neue Konflikte erstellen

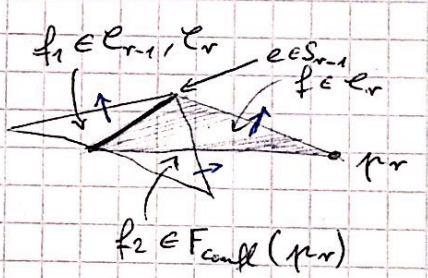
Beobachtung: falls $f \in \mathcal{C}_{r-1}$ und $f \in \mathcal{C}_r$ (also $f \notin F_{conf}(p_r)$) dann bleibt $P_{conf}(f)$ unverändert
$\Rightarrow$ wir brauchen nur $P_{conf}(f) \subseteq \{p_{r+1}, \dots, p_n\}$ erstellen für die neuen Facetten im "Dach" zu p_r.

Sei f eine neue Facette, $f_1 \in \mathcal{C}_{r-1}$ back-facing bzgl p_r, $f_2 \in \mathcal{C}_{r-1}$ front-facing; dann gilt:

$$P_{conf}(f) \subseteq P_{conf}(f_1) \cup P_{conf}(f_2)$$

denn

$$H^+(f) \subseteq H^+(f_1) \cup H^+(f_2)$$



Scanne also für Schritt 4 einfach nur die Konfliktmengen der beiden inzidenten Pgone zu e.

Bemerkung: kann passieren, daß koplanare Facetten entstehen $\Rightarrow$ einfach mergen.
(Details habe ich weggelassen; läßt sich leicht feststellen; Laufzeit ändert sich nicht)

Pseudo-Code:

bestimme $p_1, \dots, p_4$ die Tetraeder bilden

$C_4 := CH(p_1, \dots, p_4)$

Initialisiere S mit allen Paaren (p_s, f) mit $f \in C_4$, $s > 4$

for $r = 5 \dots n$:

  if $F_{conf}(p_r) = \emptyset$:

    weiter mit nächstem r // $p_r \in C_{r-1}$

  teste alle Kanten aller $f \in F_{conf}(p_r)$, ob sie zur Silhouette gehören und füge sie ggf. zu S hinzu

  for all $e \in S$:

    erzeuge neue Facette f aus e und p_r

    erzeuge Knoten in G für f

    for all $p \in P_{conf}(f_1) \cup P_{conf}(f_2)$:

      if f ist front-facing bzgl p:

        füge Kante (p, f) in G ein

    f_2 und dessen inzidente Kanten aus G löschen

Satz:

Die konvexe Hülle in 3D kann in <u>erwarteter</u> Laufzeit $O(n \log n)$ berechnet werden.

[worst case of Clarkson-Shor is $O(n^3)$]

Beweisskizze:

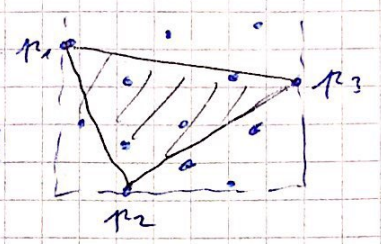
1. Zeige, daß obiger Algo $\leq 6n$ Facetten (temporär) erzeugt
2. Zeige, daß $\sum_e |P_{conf}(f_1)| + |P_{conf}(f_2)| \in O(n \log n)$, wobei die Summe über alle Kanten geht, die irgendwann einmal im Verlauf als Silhouettenkante auftreten.

Note: best-known output-sensitive algo has time complexity $O(n \log k + (nk)^{1 - \frac{1}{d/2+\varepsilon}} \cdot \log^{O(1)} n)$

k = size of output

Akl-Toussaint's throw-away heuristic

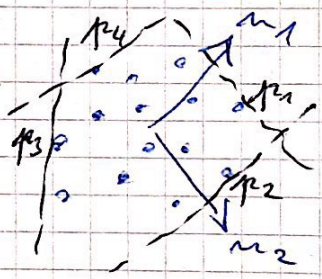
Find pts p_1 with min x-coord,
p_2 = min y-coord, p_3 = max x-coord.
Clearly, p_1, p_2, p_3 = vertices on CH!
→ Throw away all pts inside
$\triangle p_1 p_2 p_3$



In 3D: - find 4 pts realizing one of the sides of bbox(P)
- throw away all pts inside tetrahedron $p_1, \dots, p_4$

Repeat with other 3/4 sides of the bbox.

Could extend:
choose 3/4 random vectors $v_1, \dots, v_4$
find p_i^* realizing max of $\{v_i \cdot p_j\}$
form triangle / tetrahedron
throw-away all pts inside
repeat a few times (each iteration $\in O(n)$)



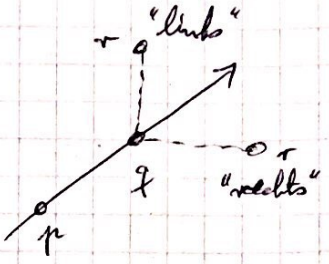
Could turn this into a probabilistic algo for CH ?!

Einfache geometrische Prädikate / Subroutinen

Prädikat "links/rechts":

Geg.: 3 Pkte $p, q, r \in \mathbb{R}^2$

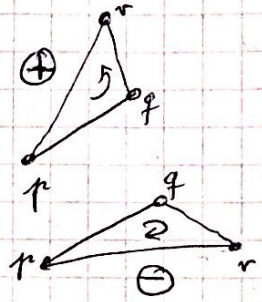
Frage: Liegt r "links" oder "rechts" von $\overline{pq}$



Lsg.: $n = \begin{pmatrix} -(q_x - p_y) \\ (q_x - p_x) \end{pmatrix}$, $F(r) = n(r-p)$

Äquivalentes Prädikat:

Bildet Δpqr einen "positiven" oder einen "negativen" Flächeninhalt?



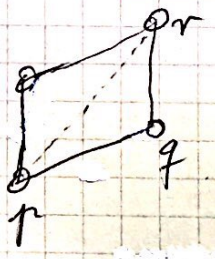
Berechnung:

$$\underset{\text{"area}}{A}(\Delta pqr) = \frac{1}{2}\begin{vmatrix} p_x & q_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix}$$

Bew.:

Flächeninhalt des Parallelogramms =

$$2 \cdot A(\Delta pqr) = \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = \begin{vmatrix} p_x - q_x & p_y - q_y & 0 \\ q_x & q_y & 1 \\ r_x - q_x & r_y - q_y & 0 \end{vmatrix}$$

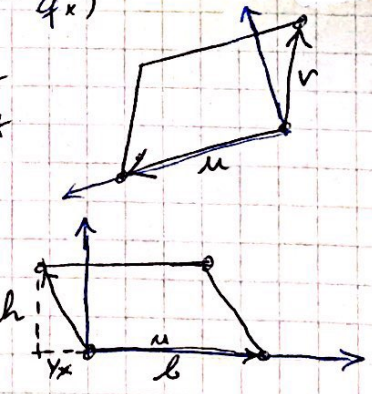


Bem.: diese Formel ist nichts anderes als $F(r)$ von oben, nur mit anderen Zeilen.

$$= (p_x - q_x)(r_y - q_y) - (p_y - q_y)(r_x - q_x)$$

$$= u_x v_y - u_y v_x \quad \text{| setze } u := p - q,\ v := r - q$$

$$= b \cdot h \quad \text{| wähle Orientierung des Koord.syst. so, daß } u = (b\ 0) \text{ und Ursprung in } q \Rightarrow \text{Höhe } h = v_y$$



Hier sieht man sofort, $\Rightarrow$

daß $A(\Delta pqr) < 0$ wird, sobald der Pkt r unterhalb der x-Achse wandert, also sobald $h < 0$.

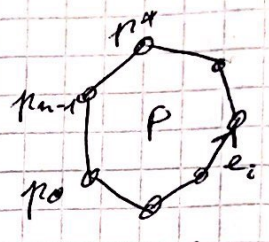
(Durch das Vorzeichen beim Flächeninhalt des Dreiecks / Parallelogramms haben wir also im Grunde nichts anderes getan, als ein Koord.syst. im Dreieck zu verankern, und die Höhe dieshalb mit VZ zu versehen!)

Hier: Orientierungstest für Tetraeder einschieben, 2 Seiten weiter

Extreme Pkte eines konvexen Polygons: Intervallhalbierung in binärsuchen Folgen

Geg.: konvexes Pgon P mit n Vertices $p^0, p^1, \dots, p^{n-1}, p^n = p^0$

Aufgabe: bestimme den extremalen Pkt bzgl. der pos. y-Achse, d.h., p^* mit $p^*_y = \max$



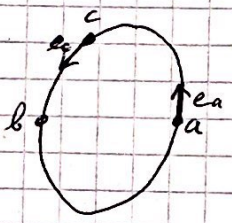
Def.: $e_i := p_{i+1} - p_i$

Annahme: wir wissen, daß p^* zwischen p_a und p_b liegt, d.h. in der Menge $\{p_a, p_{a+1}, \dots, p_b\}$ (wobei "+" = Addition modulo n!).

Setze $c := \frac{1}{2}(a+b)$

Fallunterscheidung: (6 Fälle)

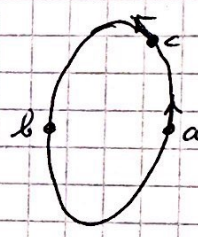
I. e_a ↑ (zeigt nach oben):



1. e_c ↘

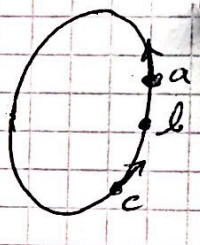
Dann wähle als nächstes Intervall a', b' ...

⇒ (a, c)



2. e_c ↑, p_c oberhalb p_a

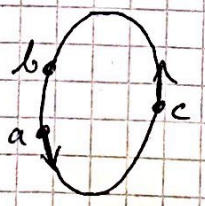
⇒ (c, b)



3. e_c ↑, p_c unterhalb p_a

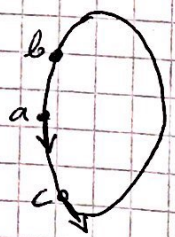
⇒ (a, c)

II. e_a ↘:



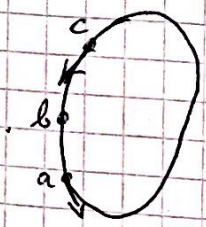
1. e_c ↗

⇒ (c, b)



2. e_c ↘, p_c unterhalb p_a

⇒ (c, b)



3. e_c ↘, p_c oberhalb p_a

⇒ (a, c)

aufgelöste Fallunterscheidung: gegenüberliegende Seite!

Start mit a=0, b=n ⇒ $0 \le a < c < b \le n$;

und wir müssen nicht modulo n rechnen.

Bem.: die o.g. Fälle reichen; z.B. in Fall II gilt: p_a muß auf linker Seite von P liegen ⇒ auch p_b, da [a, b] ja p^* enthält laut Voraussetzung!

Laufzeit: $O(\log n)$ (kein Preprocessing nötig!)

Bemerkung: Variante ist Fibonacci-Suche (s. Numerical Recipes)
hat einen etwas niedrigeren Faktor

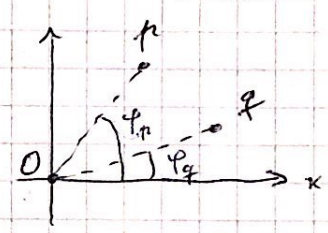
Analog: Tangente finden geht in $O(\log n)$ → Übungsaufg.! 

Aufgabe: 2 Pkte bzgl. Polarwinkel sortieren

Geg.: $p, q \in \mathbb{R}^2$

Frage: hat p oder q den größeren Polarwinkel?
(also: bildet p/q den größeren Winkel mit der x-Achse?)

Naive Lsg.: kartesische Koord. in
Polarkoord (r, φ) umwandeln (teuer)



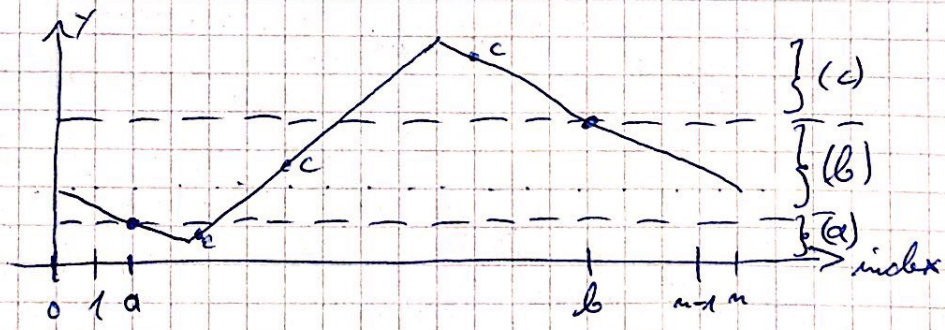
Besser: $\varphi_p > \varphi_q \Leftrightarrow A(\triangle 0qp) > 0$!

Vorteil: es genügen ein paar Add./Mult.

Einfachere Fallunterscheidung:

o ~~BdA~~: $a < b$

I. 3 Fälle:



a) $p_y^a < p_y^c < p_y^a < p_y^b \Rightarrow a' := c,\ b' := b$

b) $p_y^a < p_y^c < p_y^b \Rightarrow a' = c,\ b' = b$

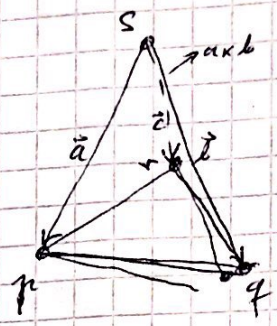
c) $p_y^a < p_y^b < p_y^c \Rightarrow a' = a,\ b' = c$

II. Analog 3 weitere Fälle mit $p_y^a > p_y^b$

Orientierungstest in 3D:

Gegeben 4 Pkte $p, q, r, s \in \mathbb{R}^3$

Frage: liegt s "oberhalb" / "unterhalb" der supporting plane von p, q, r?



Definition:

s "oberhalb" p, q, r $:\Leftrightarrow$

von s aus betrachtet erscheint p, q, r in CCW order $\Leftrightarrow$

s liegt auf der selben Seite wie die Normale von p, q, r $\Leftrightarrow$

$$\vec{c} \cdot (\vec{a} \times \vec{b}) > 0 \quad , \text{ mit } a = p - s,\ b = q - s,\ c = r - s \Leftrightarrow$$

$$\det \begin{pmatrix} p_x & p_y & p_z & 1 \\ q_x & q_y & q_z & 1 \\ r_x & r_y & r_z & 1 \\ s_x & s_y & s_z & 1 \end{pmatrix} > 0$$

Bew:

In der Det untere Zeile von oben darüber abziehen, und Det entwickeln nach letzter Zeile. Dann Spatprodukt ausschreiben und mit Det vergleichen.

Bemerkung:

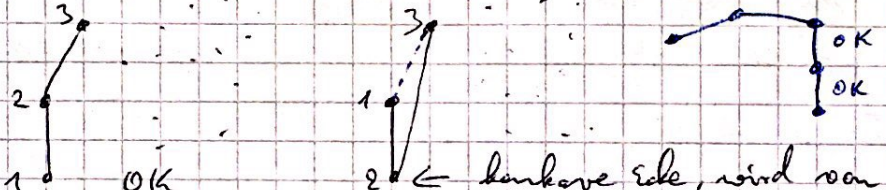
1. Vol (Spates durch p, q, r, s aufgespannt) $= c \cdot (a \times b)$
   → "Spatprodukt"
2. Vol (Tetraeders durch p, q, r, s) $= \frac{1}{6} \det \begin{pmatrix} -p-1 \\ -q-1 \\ -r-1 \\ -s-1 \end{pmatrix}$
   → vorzeichenbehaftetes Volumen.
3. Det $= c \cdot (a \times b) = 0 \Leftrightarrow p, q, r, s$ liegen in einer Ebene

Geometrische Robustheit

Hier: nur am Beispiel der 2D-CH

Frage: wo kann unser Algo "Graham's Scan" schief gehen? wo haben wir stillschweigend Annahmen über den Input gemacht?

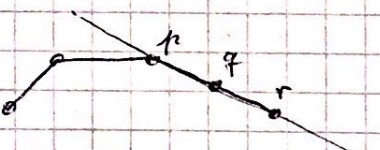
1. Keine gleichen x-Koord. — Problem? → Ü.aufgabe
   Nein: Reihenfolge der p_i mit gleichem x ist egal
   → Sort-Algo muss nicht stabil sein



← konkave Ecke, wird von Algo entfernt

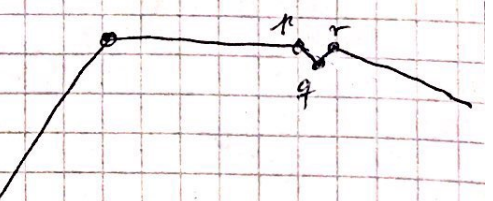
   Einziger Fall: vertikale Kante in der CH am linken oder rechten Ende taucht in der oberen und unteren Hülle auf → extra checken (Frage: klappt das auch, wenn mehrere Ekte kollinear und vertikal sind?..)

2. Kollineare Ekte:
   Unser Prädikat $A(\Delta pqr) = 0$



   ⇒ schreibe in der Impl. "<= 0" oder evtl. "<= eps".

3. Rundungsfehler:
   Wenn p, q, r sehr dicht beieinander liegen, kann $A(\Delta pqr) > 0$ liefern, wg. begrenzter Anzahl Bits.



   Lsg a) solche "Cluster" ersetzen durch 1 Ekt
   Lsg b) geometrische Prädikate numerisch "geschickt" implementieren

Der Begriff "allgemeine Lage" (general position):
Man findet häufig einen Satz der startet "we assume a set of points in general position...".
Ist nicht präzise definiert; bedeutet, daß alle im Folgenden verwendeten geometrischen Prädikate eindeutig sind, also keine Grenzfälle entstehen ("=0").
Läuft meistens auf folgende Bedingungen hinaus:
1. keine 3 Pkte kollinear
2. in 3D: keine 4 Pkte koplanar
3. keine doppelten x-Koord (vor allem bei Sortier- oder sweep-plane)
4. evtl.: keine 4/5 auf einer gemeinsamen Kugel (in 2D / 3D)

Definitionen:
Ein "geometrisches Problem" ist eine Abb.

$$P: X \to Y$$

wobei $X = \mathbb{R}^{d \cdot n}$ (= set of correct/allowed inputs), $Y = C \times \mathbb{R}^m$, $C =$ "kombinatorischer Anteil der Lsg".
Bsp.: für 2D-CH ist $Y = C = \{$ all possible permutations aller k-Tupel aus $\mathbb{N}$, $k \leq n \}$
(bei 2D-CH werden nämlich im Prinzip nur einige Pkte der Eingabe ausgewählt und in eine Reihenfolge gebracht.)

Ein "geometrischer Algo" ist genauso eine Abb.

$$A: X \to Y$$

Achtung: i.A. $\exists x: P(x) \neq A(x)$!

Definition:

$\mathcal{A}$ heißt "robust" :$\Leftrightarrow$

$\forall x \in X \; \exists x' \in X: \; \mathcal{A}(x) = P(x')$ ich denke eher: $\mathcal{A}(x') = P(x)$

(D.h.: es gibt keine korrekte Eingabe, bei der der Algo. kompletten Blödsinn produziert, also eine Ausgabe, die nicht der Ausgabespezifikation genügt.)

$\mathcal{A}$ heißt "stabil" :$\Leftrightarrow$

$\forall x \in X \; \exists x' \in X$, x' nahe bei x: $\mathcal{A}(x) = P(x')$

Frage: ist 2D-CH robust / stabil?

(ohne / mit Robustheitsmaßnahmen ...)