

Allgemein: Seien $p = (p_x, p_y)$, $q = (q_x, q_y)$ und $r = (r_x, r_y)$ drei Pkte im \mathbb{R}^2 . (3)

Sei \vec{p} der Vektor, der vom Nullpkt ausgeht und in Richtung p zeigt. Und seien \vec{q} und \vec{r} Vektoren, die von p ausgehen und in Richtung q bzw. r zeigen.

Betrachte nun das Parallelogramm, welches durch die Vektoren \vec{q} und \vec{r} aufgespannt wird. Sei $V(P) =$ die Fläche des Parallelogramms.

Beh: $V(P) = |\det A|$, wobei $A = \begin{pmatrix} q_x - p_x & q_y - p_y \\ r_x - p_x & r_y - p_y \end{pmatrix}$

Bem: diese Beh. überträgt sich auch auf \mathbb{R}^n .

Man betrachte hierbei $u_1, \dots, u_n \in \mathbb{R}^n$ und $P = \{u_1 a_1 + \dots + u_n a_n : 0 \leq a_i \leq 1, i \in \{1, \dots, n\}\}$.
 $\Rightarrow V(P) = |\det A|$, A entsprechend.

Wir betrachten Δpqr :

1.2.6.1 Satz: $p, q, r \in \mathbb{R}^2$, $A = \begin{pmatrix} q_x - p_x & q_y - p_y \\ r_x - p_x & r_y - p_y \end{pmatrix}$

\Rightarrow (i) $|\det A| = 2 \cdot$ Fläche von Δpqr .

(ii) $\text{sign}(\det A)$ gibt die Orientierung an:

-1: neg. orientiert (right turn)

0: kollinear.

+1: pos. orientiert (left turn).

Bew. siehe Lernordner.

1.2.6.2 Def: Wir definieren das geometrische Prädikat:

$$\text{orientation}(p, q, r) = \text{sign} \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = \text{sign}((q_x - p_x)(r_y - p_y) - (r_x - p_x)(q_y - p_y)).$$

D.h. orientation (p, q, r) und damit der Test $dr < dq$ kann mit zwei Multiplikationen und fünf Subtraktionen berechnet werden.

2) Falls $dr = dq$ (d.h. orientation $(p, q, r) = 0$) müssen wir herausfinden, welcher der Pkte q oder r weiter von p entfernt liegt.

Natur: Berechne Distanzen (mit eukl. Metrik):

$$\text{dist}(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

\leadsto Nachteil: Wurzel!

\Rightarrow Besser: Vergleiche Quadrate der Distanzen:

$$(p_x - q_x)^2 + (p_y - q_y)^2 \stackrel{?}{<} (p_x - r_x)^2 + (p_y - r_y)^2 ?$$

(\leadsto 4 Subtraktionen, 4 Multiplikationen, 2 Additionen).

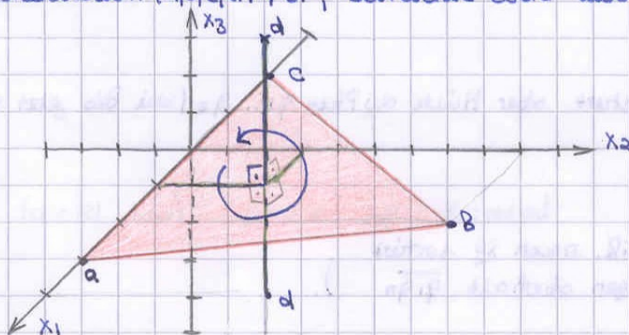
1.2.7. Übertragung auf höhere Dimensionen:

Das Orientierungsprädikat kann auf höhere Dimensionen verallgemeinert werden.

Insbesondere im \mathbb{R}^3 : seien $a, b, c, d \in \mathbb{R}^3$

orientation $(a, b, c, d) \in \{-1, 0, 1\}$

Sei orientation $(a, b, c, d) \neq 0$, betrachte Ebene durch a, b, c .



1.2.7.1 Satz: Für $a, b, c, d \in \mathbb{R}^3$ betrachte Ebene durch a, b, c .

Falls orientation $\neq 0$, so gilt:

(i) orientation $(a, b, c, d) = -1$, wenn von d aus $\Delta a, b, c$ negativ orientiert ist.
 orientation $(a, b, c, d) = +1$, wenn von d aus $\Delta a, b, c$ positiv orientiert ist.

(ii) orientation $(a, b, c, d) = \text{sign} \begin{vmatrix} a_{x_1} & a_{x_2} & a_{x_3} & 1 \\ b_{x_1} & b_{x_2} & b_{x_3} & 1 \\ c_{x_1} & c_{x_2} & c_{x_3} & 1 \\ d_{x_1} & d_{x_2} & d_{x_3} & 1 \end{vmatrix}$

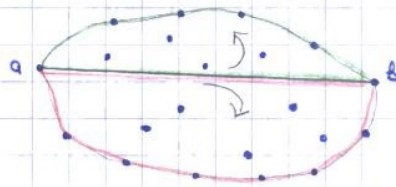
1.3 Algorithmus II: Graham's Scan.

1.3.1. Idee: Berechne "obere" und "untere" Hülle getrennt.

Sei a der linkeste und b der rechteste Pkte von S , d.h. $a = \text{Minimum bzgl. } x\text{-y-Sortierung}$ und $b = \text{Maximum bzgl. } x\text{-y-Sortierung}$.

Obere Hülle = Der Teil von $CH(S)$, der oberhalb von \overline{ab} liegt (inclusive a, b). $= CH(S_1), S_1 \subset S$.

Untere Hülle = Der Teil von $CH(S)$, der unterhalb von \overline{ab} liegt (incl. a, b) $= CH(S_2), S_2 \subset S$.



Beobachtung: $S_1 = \{p \in S : \text{orientation}(a, b, p) \geq 0\}$
und $S_2 = \{p \in S : \text{orientation}(a, b, p) < 0\}$.

Sei S_1 lexikografisch nach $x\text{-y}$ -Sortiert ($\Rightarrow n \log n$)

\leadsto sortierte Folge $q_1 \dots q_n = S_1$

$\Rightarrow a = q_1$ und $b = q_n$

Berechne obere Hülle als Teilfolge $x_1 \dots x_m$ von $q_1 \dots q_n$

(d.h. im Uhrzeigersinn).



Beobachtung:

1. Jeweils drei aufeinanderfolgende Ecken x_i, x_{i+1}, x_{i+2} bilden einen Rightturn, d.h. $\text{orientation}(x_i, x_{i+1}, x_{i+2}) < 0$.

2. (b, a, p) ist Rightturn $\forall p \in S_1$.

(d.h. $\forall p \in S_1 : \text{orientation}(q_n, q_1, p) < 0$).

1.3.2 Algorithmus:

verwalte stack $x_0, x_1 \dots x_t$ von potentiellen Ecken der oberen Hülle.

Am Ende: genau die Ecken der oberen Hülle.

Initialisierung: stack $S = q_n, q_1, q_2$. Bem: alle drei Pkte wg. lexik. Sort. genau festgelegt und müssen nicht gesucht werden.

Schleife: Betrachte nur $q_3 \dots q_{n-1}$.

Sei q_s aktueller Pkte

Invariante: $x_0, x_1 \dots x_t$ ist Teilfolge von $q_n, q_1 \dots q_s$

mit: $t \geq 2, x_0 = q_n, x_1 = q_1, x_t = q_s$

besser x_1 statt x_0 $x_0, x_1 \dots x_t$ ist konvexes Polygon

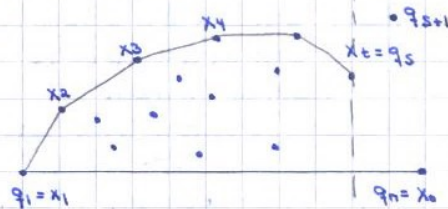
2) obere Hülle von S ist Teilfolge von $x_1 \dots x_t q_{s+1} \dots q_n$.

Schritt: $s \mapsto s+1$

wähle (x_{t-1}, x_t, q_{s+1}) nicht Rightturn da $\text{pop}(x_t)$

od

$\text{push}(q_{s+1})$.



Beobachtung: stack speichert obere Hülle der Pkte $q_1 \dots q_s$ (die bis jetzt gewesen).

Die Funktion:

UPPER_HULL($q_1 \dots q_n$) bessers: $q_1 \dots q_m \Rightarrow b = q_m$ (weil $|S| = n$)

(Vorb. 1) $q_1 \dots q_n$ lexik. nach $x\text{-y}$ sortiert

2) $q_2 \dots q_{n-1}$ liegen oberhalb $\overline{q_1 q_n}$).

Stack von Pkten. S

(stack <point> S_i)

$S_i.\text{push}(q_n)$;

$S_i.\text{push}(q_1)$;

$S_i.\text{push}(q_2)$;

(Ann. $n \geq 2$).

→


```

for s=3 to n-1 do
  x ← S.top();      gibt das oberste Element an.
  y ← S.top_pred();  eins unter dem obersten auf dem Stack
  while orientation(y, x, q_s) ≥ 0 do
    S.pop();
    x ← y;
    y ← S.top_pred();
  od
  S.push(q_s);
od
return S.
    
```

While-Schleife terminiert spätestens, wenn S nur noch die Pkte q_n, q_1 enthält. (weil alle Pkte $q_2 \dots q_{n-1}$ oberhalb von $\overline{q_1 q_n}$ liegen.).

1.3.3. Beispiel für die Pkt:

Anfang: Stack $q_6 q_1 q_2$.

s=3

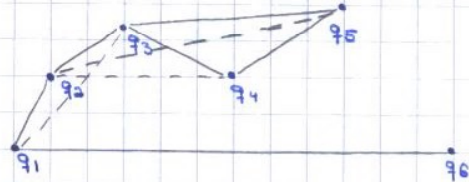
$x = q_2 \wedge y = q_1$
 $orientation(q_1, q_2, q_2) = -1$
 \Rightarrow Stack $q_6 q_1 q_2 q_3$

s=4

$x = q_3 \wedge y = q_2$
 $orientation(q_2, q_3, q_4) = -1 \Rightarrow$ Stack $q_6 q_1 q_2 q_3 q_4$

s=5

$x = q_4 \wedge y = q_3$
 $orientation(q_3, q_4, q_5) = +1 \Rightarrow$ Stack $q_6 q_1 q_2 q_3$, $x = q_3, y = q_2$
 $orientation(q_2, q_3, q_5) = -1 \Rightarrow$ Stack $q_6 q_1 q_2 q_3 q_5$



Fertig!

1.3.4. Korrektheit.

(1) Invariante ist stets erfüllt.

Zeige mit Induktion, dass Invariante stets erfüllt ist.

Ind.anf.: $S = q_n q_1 q_2$

- 1) $t = 2, q_n = x_0, q_1 = x_1, q_2 = x_2$
- 2) $x_0 x_1 x_2$ konv. Polygon
- 3) obere Hülle von S ist TF von $x_0 x_1 x_2 q_2 \dots q_{n-1}$.

Indann: die Invariante sei für $x_0 x_1 \dots x_t$ erfüllt ($t \in \mathbb{N}$).

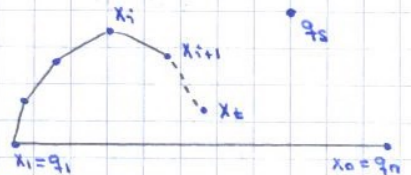
Indbeh: Sei q_s der nächste Pkt bzgl. der lexikogr. Sortierung.

Wenn $orientation(x_{t-1}, x_t, q_s) = -1$ dann ist die Beh. trivialerweise erfüllt.

Sei also $orientation(x_{t-1}, x_t, q_s) \in \{1, 0\}$

z.z. $x_0 \dots x_t$ ist TF von $q_n q_1 \dots q_s$ mit

- 1) $t \geq 2, x_0 = q_n, x_1 = q_1, x_t = q_s$
- 2) $x_0 \dots x_t$ konv. Polygon
- 3) obere Hülle von S ist TF von $x_1 \dots x_t q_{s+1} \dots q_n$.



$\Rightarrow S = x_0 x_1 \dots x_{t-1}$

... wird solange gepopt bis $orientation(y, x, q_s) < 0$.

$\Rightarrow S = x_0 x_1 \dots x_i$

$S.push(q_s) \Rightarrow S = x_0 x_1 \dots x_i x_t$, wobei $x_t = q_s$

$\Rightarrow x_0 x_1 \dots x_i x_t$ ist TF von $q_n q_1 \dots q_s$

zu 1) $t \geq 2, x_0 = q_n, x_1 = q_1$ und $x_t = q_s$ nach Def. \Rightarrow ok.

weil $(x_0 x_1 q_s)$ Rightturn $\Rightarrow i \geq 1 \Rightarrow t \geq 2$.

zu 2) Folge $x_0 \dots x_i$ unverändert und (x_{i-1}, x_i, q_s) bilden Rightturn

$\Rightarrow x_0 \dots x_i x_t$ auch konvexes Polygon.

zu 3) Pkte $x_{i+1} \dots x_{t-1}$ wurden entfernt. Sie gehören nicht zu oberen Hülle, da sie rechts von (oder auf) Strecke $\overline{x_i q_s}$ liegen. \Rightarrow ok.

\Rightarrow Beh.

überall x_1 .

(2) Bei Termination erhält Stack S die obere Hülle.

Voraussetzung: die Invariante ist für den letzten Pkt ($s=n$) erfüllt.

d.h. alle Pkte sind abgearbeitet.

Wissen also: $x_1 \dots x_t$ ist TF von $q_1 \dots q_n$ mit $\text{bea: } \text{lauf} \geq 1$, weil $x_0 = x_t = q_n$.

1) $t \geq 2$, $x_0 = q_n$, $x_1 = q_1$ und $x_t = q_n$

2) $x_1 \dots x_t$ konv. Polygon

3) obere Hülle ist TF von $x_1 \dots x_t$.

Z.Z. obere Hülle = $x_1 \dots x_t$ also nicht echte Teilmenge.

Ann: nein, obere Hülle $\subset x_1 \dots x_t$

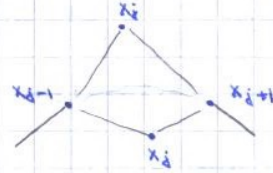
$\Rightarrow \exists x_j \in \{x_1 \dots x_t\}$ mit $x_j \notin$ obere Hülle

\Rightarrow 1. Fall: $\text{orientation}(x_{j-1}, x_j, x_{j+1}) = +1 \Rightarrow \checkmark$ zu 2)

2. Fall: $\text{orientation}(x_{j-1}, x_j, x_{j+1}) = -1 \Rightarrow \checkmark$ zu 2)

\Rightarrow Ann. falsch

\Rightarrow Beh.



13.5 Laufzeit:

Beobachtung: Jeder Pkt kann höchstens einmal aus S entfernt werden.

Rumpf hat Kosten $O(1)$ (Orientierung, konstante Zahl von Stack-Operationen).

Da jeder Pkt höchstens einmal aus S entfernt werden kann \Rightarrow

\Rightarrow while-Schleife wird höchstens n mal eingehalten \Rightarrow forschleife kostet $uns O(n)$.

In der while-Schleife konst. Operationen, außerhalb auch

\Rightarrow Gesamtkosten der UPPER-Pkt: $O(n)$.

(Analog berechnet man die untere Hülle).

Die Funktion insgesamt:

CONVEX_HULL(S)

1. $S \leftarrow \text{sort}(x_j)$;

$a \leftarrow S.\text{min}()$;

$b \leftarrow S.\text{max}()$;

2. forall $p \in S$ do eigentlich $S \setminus \{a, b\}$

3. if ($\text{orientation}(a, b, p) > 0$) $S_1.\text{append}(p)$

4. if ($\text{orientation}(a, b, p) < 0$) $S_2.\text{append}(p)$

5. od

6. $S_1.\text{push}(a)$;

$S_1.\text{append}(b)$;

7. $S_2.\text{push}(a)$;

$S_2.\text{append}(b)$;

8. $H_1 \leftarrow \text{UPPER_HULL}(S_1)$

9. $H_2 \leftarrow \text{LOWER_HULL}(S_2)$

10. H_1 umdrehen

11. H_2 an H_1 anhängen

12. doppeltes Vorkommen von a & b löschen.

$O(n \log n)$ ($|S| = n$)

$O(1)$

$O(1)$

$O(n)$

(S_1, S_2 Listen \Rightarrow append kostet $O(1)$, $n \times \Rightarrow O(n)$)

$O(n)$

(falls S_1 Liste \Rightarrow

$O(1)$

einfügen am Ende kostet $O(n)$)

$O(n)$

$O(1)$

$O(m) = O(n)$

$O(n-m) = O(n)$

$O(n)$

$O(1)$

$O(n)$

Zusammenfassung:

CONVEX_HULL(S)

1) Sortiere S lexik.

2) $a \leftarrow \min$ \wedge $b \leftarrow \max$

3) $S_1 \leftarrow \{p \in S : b, a, p \text{ Rightturn}\} \cup \{a, b\}$

4) $S_2 \leftarrow \{p \in S : b, a, p \text{ Leftturn}\} \cup \{a, b\}$

5) UPPER_HULL(S_1)

6) LOWER_HULL(S_2)

7) " Zusammenkleben

Zeit:

$O(n \log n)$

$O(1)$

$O(n)$

$O(n)$

$O(n)$

$O(n)$

$O(n)$

1.3.6. Satz: Sei S Menge von n Pkten im \mathbb{R}^2

- a) CH(S) kann in Zeit $O(n \log n)$ berechnet werden (worst case)
- b. Falls S lexik. nach xy-Koord. sortiert ist, dann kann CH(S) in Zeit $O(n)$ berechnet werden.

Bew. siehe oben.

1.3.7. Bemerkung:

- 1) Alg. optimal, da CH-Problem (i.a.) genauso schwierig ist wie sortieren.
Bew. Übung.
- 2). obere bzw. untere Hülle sind auch für sich alleine wichtig (für bestimmte Probleme)
- 3). Optimalität gilt für worst case; es gibt Situationen, in denen Alg. I. besser ist (wenn #Ecken von CH(S) $< \log n$).

1.3.8. Varianten von Graham's Scan:

- a) siehe Übung
- b). untere u. obere Hülle gleichzeitig berechnen
- c). Berechnung der gesamten Hülle in einer Phase.

Idee: 1. sortiere $S = p_1 \dots p_n$.
 2. Allgemeiner Schritt:

Bearbeite $p_i, i = 4 \dots n$.
 Situation: $C_{i-1} := CH(\{p_1 \dots p_{i-1}\})$ ist berechnet.
 p_{i-1} ist maximal in $\{p_1 \dots p_{i-1}\}$
 $\Rightarrow p_{i-1}$ ist die rechteste Ecke von C_{i-1} .
 $\Rightarrow \overline{p_{i-1} p_i} \cap C_{i-1} = \{p_{i-1}\}$.

Der eigentliche Schritt:

Berechne Berührungspkte t und b der beiden Tangenten von p_i an C_{i-1} .

Genauer: $t \leftarrow p_{i-1}$ dh. orientation($p_i, t, succ(t)$) ≤ 0
 while ($p_i, t, succ(t)$) nicht rightturn do
 $t \leftarrow succ(t)$
 od
 $b \leftarrow p_{i-1}$ dh. orientation($p_i, b, pred(b)$) ≥ 0
 while ($p_i, b, pred(b)$) nicht rightturn do
 $b \leftarrow pred(b)$
 od

Bzw. hier ähnliche Argumentation:
 die # unterwegs besuchten Pkte kann in $\Omega(n)$ liegen,
 aber jeder von ihnen wird anschließend aus
 der konvexen Hülle entfernt und kann danach
 nie wieder als Eckpt in Erscheinung treten
 Also wird er auch nie wieder besucht
 \Rightarrow Es kann insgesamt höchstens n Besuche geben
 $\Rightarrow O(n \log n)$ Laufzeit

Nachdem wir nun b und t berechnet haben:
 • Entferne alle Pkte zwischen b und t
 • Füge p_i nach C_i ein.

Weitere Implementierungsdetails und Laufzeitanalyse siehe 3. Übung.
 Bsp. zu c) siehe Lemoralner.

1.4. Algorithmus III: Divide & Conquer

Triangulierung schauen

1.4.1. Spezialfall: Konvexe Hülle von zwei konvexen Polygonen P und Q.

$P = p_1 \dots p_m, Q = q_1 \dots q_n$, Ecken gg. Uhrzeigerstn sortiert.

Aufgabe: Berechne $CH(P \cup Q)$

Triviale Lsg: Graham's Scan auf die Menge aller Ecken. $O(n \log n)$

Bessere Lsg: Ausnutzen der Polygonstruktur. $O(n)$

- 1) Finde Sortierung aller Ecken in Zeit $O(n)$
- 2) Berechne die Hülle in Zeit $O(n)$ (siehe S.1.3.6. 8).

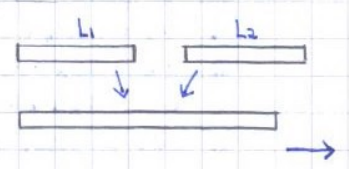
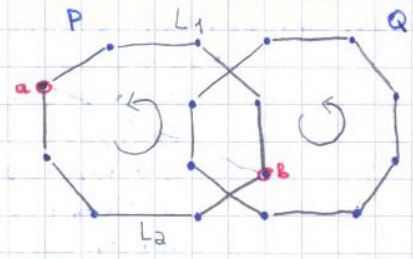
zu 1): Betrachte P:

1) Finde extreme Ecken a und b in Zeit $O(m)$

L_2 := untere Polygonzug
 starte bei a und Laufe gg. Uhrzeigerstn über P bis b erreicht ist.

L_1 := obere Polygonzug

Laufe von b nach a. und drehe um.



(ii) Mische L_1 und L_2 zu einer sortierten Gesamtliste L_p zusammen in Zeit $O(m)$ (\rightarrow siehe Mergesort)

Analog: Sortiere Folge L_2 der Ecken von Q in Zeit $O(p)$.

(iii) Mische L_p und L_2 in Zeit $O(m+p) = O(n)$ (wobei $n := m+p$) zu einer sortierten Gesamtfolge zusammen.

\rightarrow dann Graham's Scan.

1.4.2 Allgemein: Sei $S \subset \mathbb{R}^2$, $|S|=n$.

CONVEX_HULL(S).

if $|S|=1$ then

output S

else

teile S in zwei möglichst gleich große Teile S_1 und S_2 } DEVIDE

(z.B. $|S_1| = \lceil |S|/2 \rceil$ und $|S_2| = \lfloor |S|/2 \rfloor$)

$P \leftarrow \text{CONVEX_HULL}(S_1)$

$Q \leftarrow \text{CONVEX_HULL}(S_2)$

berechne $\text{CH}(P \cup Q)$ wie in letzter Vorlesung gezeigt } MISCHSCHRITT

fi.

\uparrow
hier wird die eigentliche
Rechnung durchgeführt.

1.4.3 Laufzeit:

Teilen: $O(n)$

Mischen: $O(n)$

Merge auf Listen

Graham's Scan (ohne Sortieren).

$$\Rightarrow T(n) = \begin{cases} c_0, & n=1 \\ c_1 \cdot n + 2 \cdot T(n/2), & n>1 \end{cases}$$

Das ist also dieselbe Rekursion wie für Mergesort (aber mit anderen Konstanten natürlich).

\Rightarrow Gesamtlaufzeit des Verfahrens ist $O(n \log n)$ (siehe Analyse von Mergesort).

1.5 Eine Anwendung vom CONVEX HULL

1.5.1 Problem: Geg. sind n Halbebenen $H_1 \dots H_n$ von \mathbb{R}^2

Berechne $P = \bigcap_{i=1}^n H_i$

1.5.2 Def: Eine abgeschlossene Halbebene = $\{ \text{alle Pkte auf gleicher Seite einer Geraden } k \} \cup k$.

1.5.3 Ann: Schnitt P ist konvexes (möglicherweise unbeschränktes) Polygon, da der Schnitt konvexer Mengen wieder konvex ist.

1.5.4 Ziel und Lösungsansatz:

Ziel: Berechne die Folge der Ecken von P gg. den Uhrzeigersinn.

Lösung: Zurückführung auf konvexe Hülle einer geeigneten Pktmenge.

Dazu transformieren wir die definierenden Geraden in "duale" Pkte.

1.5.5 Definition: (Geometr. Transformation)

a). Sei $L = \{ y : y = ax + b, x \text{ bel.}, a, b \text{ fest} \}$ eine nicht vertikale Gerade.
Geradengleichung von L .

Der Punkt $D(L) := (a, b)$ heißt der duale Punkt zu L .

b). Sei $p = (a, b) \in \mathbb{R}^2$ ein Pkt. Die Gerade $D(p) = \{ y : y = -ax + b, x \text{ beliebig} \}$ heißt die duale Gerade zu p .

Abbildung D erlaubt die relative Lage von Objekten.

1.5.5 Lemma: Pkt p liegt auf (oberhalb / unterhalb) einer Geraden L

\Leftrightarrow Gerade $D(p)$ liegt auf (oberhalb / unterhalb) Pkt $D(L)$.

Beweis: Sei $p = (p_x, p_y)$, $L = \{ y \in \mathbb{R} : y = ax + b, x \in \mathbb{R} \}$ (a, b fest).

$\Rightarrow D(L) = (a, b)$ und $D(p) = \{ y : y = -p_x x + p_y, x \in \mathbb{R} \}$

• sei p gelegen auf L

$\Leftrightarrow p_y = a \cdot p_x + b$

$\Leftrightarrow -b = ap_x - p_y$

$\Leftrightarrow b = -p_x a + p_y$

$\Leftrightarrow D(p)$ liegt auf $D(L)$.

• sei p gelegen oberhalb L

$\Leftrightarrow ax + b < p_y \quad \forall x \in \mathbb{R}$

Bzw. sogar $ap_x + b < p_y$.

$\Leftrightarrow -ap_x - b > -p_y$

$\Leftrightarrow -ap_x + p_y > b$

$\Leftrightarrow D(p)$ liegt oberhalb $D(L)$.

(unterhalb analog).