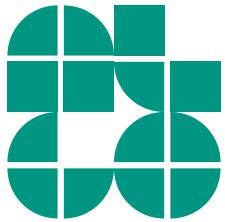
Übung Algorithmische Geometrie
Einführung & Konvexe Hülle

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

Andreas Gemsa

21.04.2011



AlgoGeom-Team

Dozent

- Martin Nöllenburg
- noellenburg@kit.edu
- Raum 319
- Sprechzeiten: Termin per Mail vereinbaren

Übungsleiter

- Andreas Gemsa
- gemsa@kit.edu
- Raum 322
- Sprechzeiten: Termin per Mail vereinbaren

Termine

- Vorlesung: Di 9:45 – 11:15 Uhr, Raum 131
- Übung: Do 10:15 – 11:00 Uhr, Raum 131 (ab 21.04.)

AlgoGeom-Team

Dozent

- Martin Nöllenburg
- noellenburg@kit.edu
- Raum 319
- Sprechzeiten: Termin per Mail vereinbaren

Übungsleiter

- Andreas Gemsa
- gemsa@kit.edu
- Raum 322
- Sprechzeiten: Termin per Mail vereinbaren

Termine

- Vorlesung: Di 9:45 – 11:15 Uhr, Raum 131
- Übung: Do 10:15 – 11:00 Uhr, Raum 131 (ab 21.04.)

Übungsbetrieb

Ausgabe Blatt 1
for Woche $i = 2 \ldots 14$ **do**
    if Tag == Dienstag **then**
        Ausgabe Blatt i
        Abgabe Blatt $(i-1)$
    if Tag == Donnerstag & !isFeiertag(Tag) **then**
        Besprechung Blatt $(i-1)$

- Bearbeitung der Aufgaben und Abgabe der Lösungen in Zweiergruppen erwünscht
- Übung wöchentlich 45 Minuten
- abweichende Regelung an Feiertagen
- ~~Achtung: erste Übung am 21.04. in Raum 118~~

Ablauf

- Besprechung ÜB1
  - Einstieg
  - Grundlegendes
  - Optimalität!
- Kontextsensitiver Algorithmus

- Optimalität?!

Fragen

Fragen zur Vorlesung

Algorithmen
Laufzeiten
...

Fragen zum Ablauf der Übung

...

Übungsblatt 1 - Aufgabe 1 - a)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

Übungsblatt 1 - Aufgabe 1 - a)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Übungsblatt 1 - Aufgabe 1 - a)

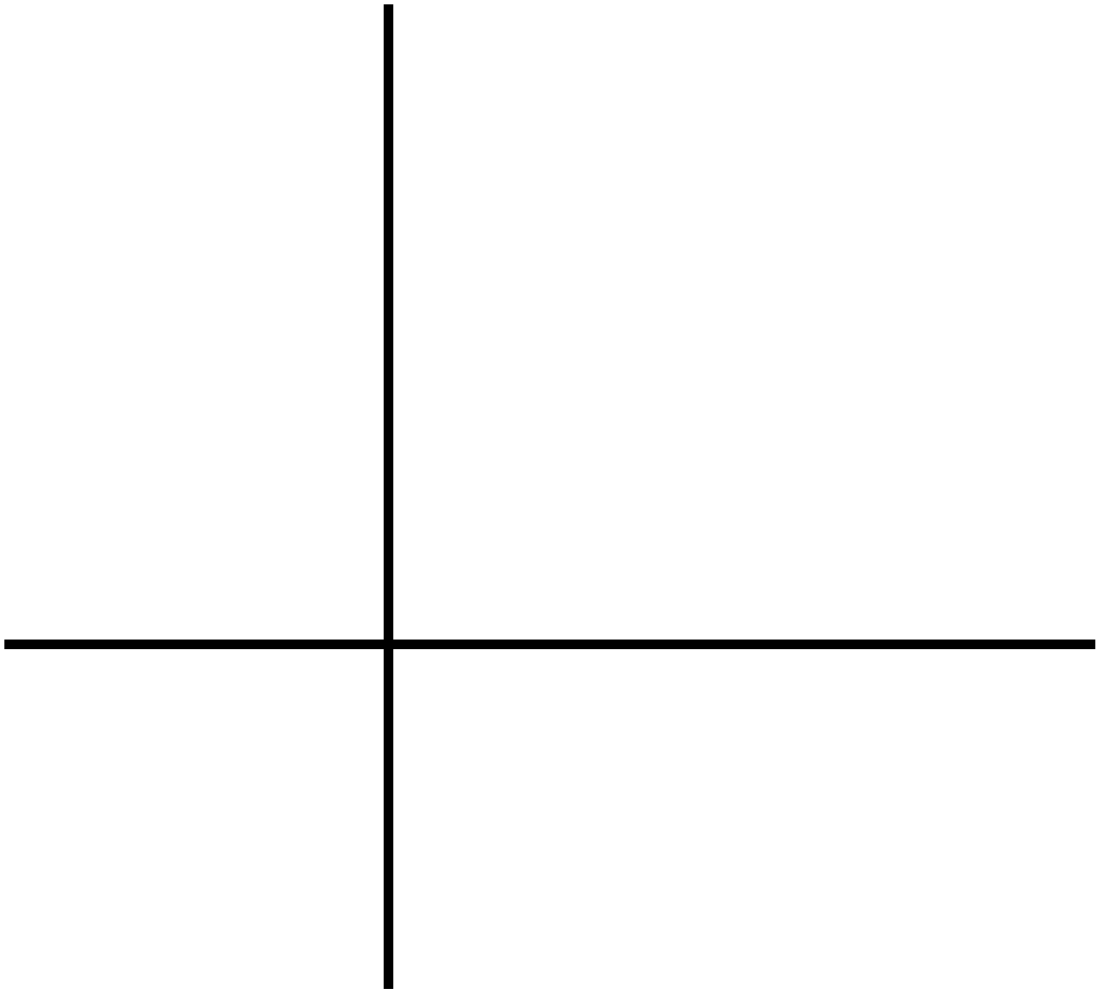
Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

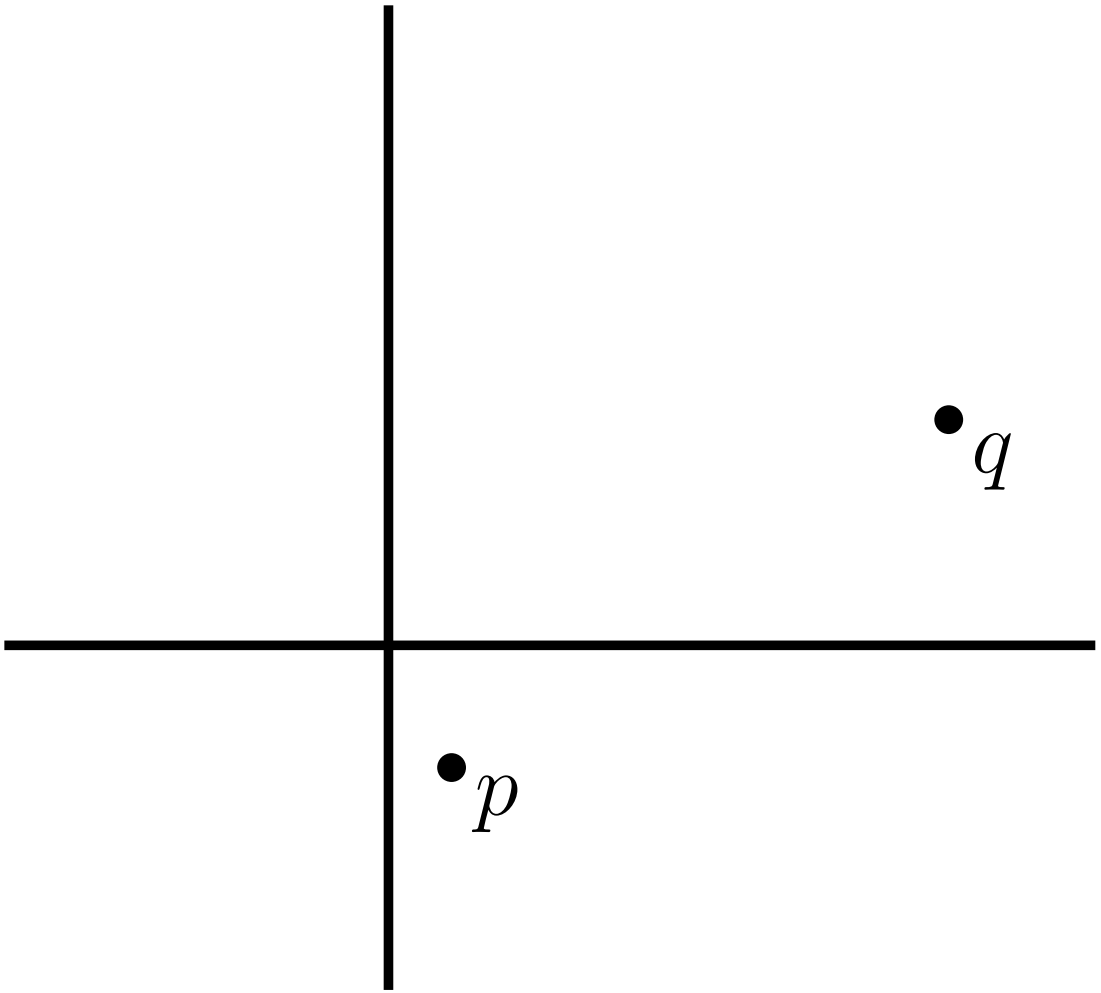
$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

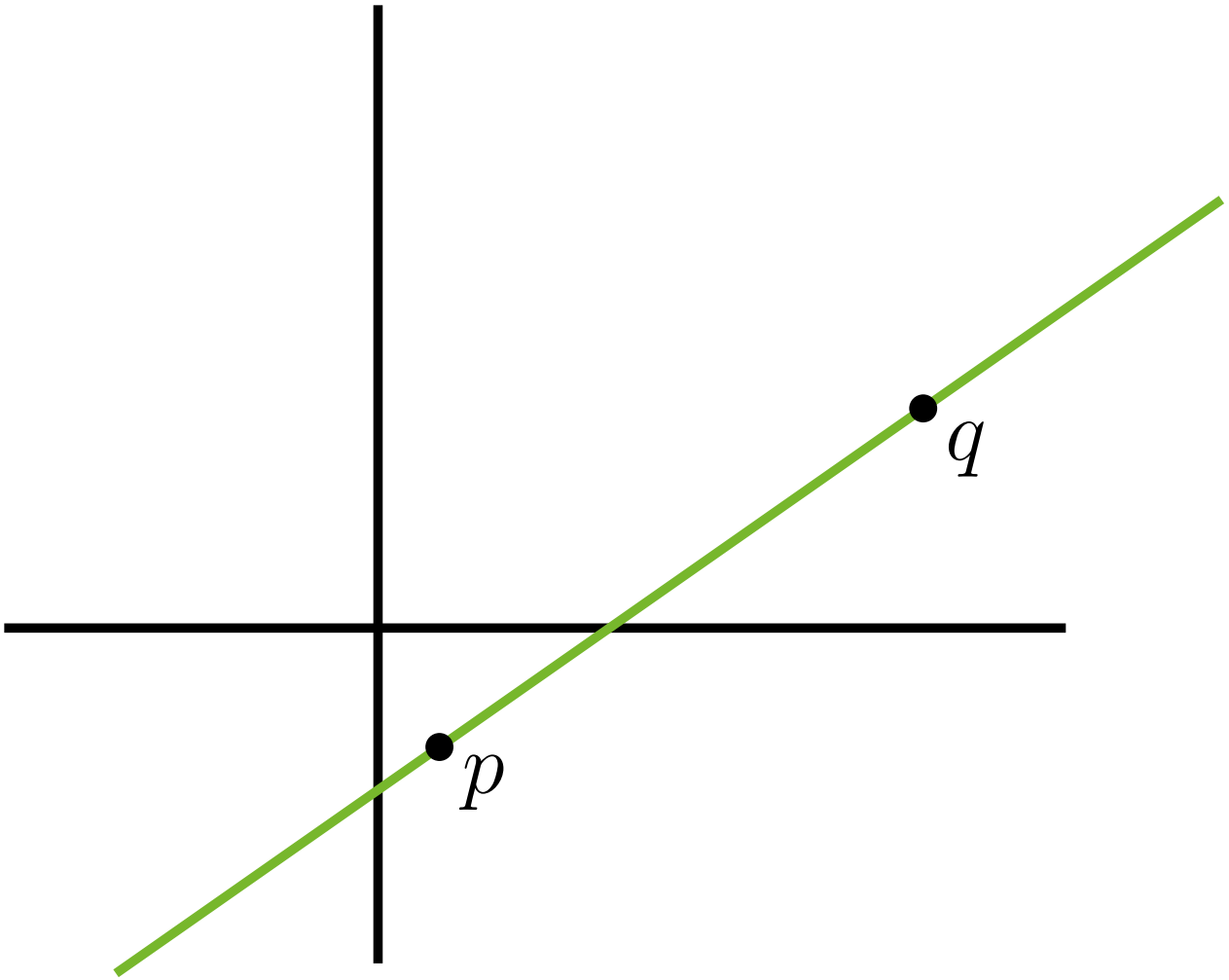
Aufgabe 1 - a)



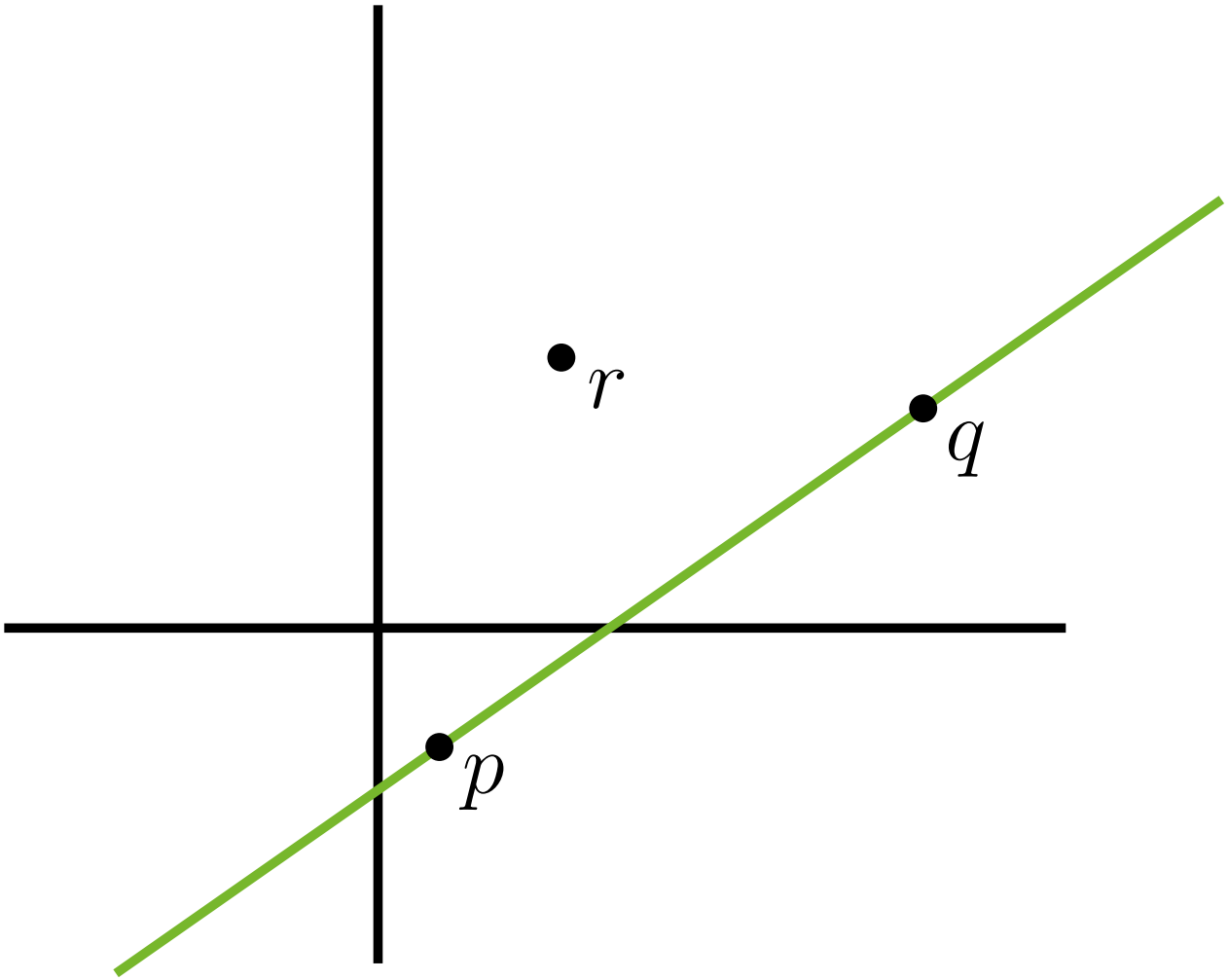
Aufgabe 1 - a)



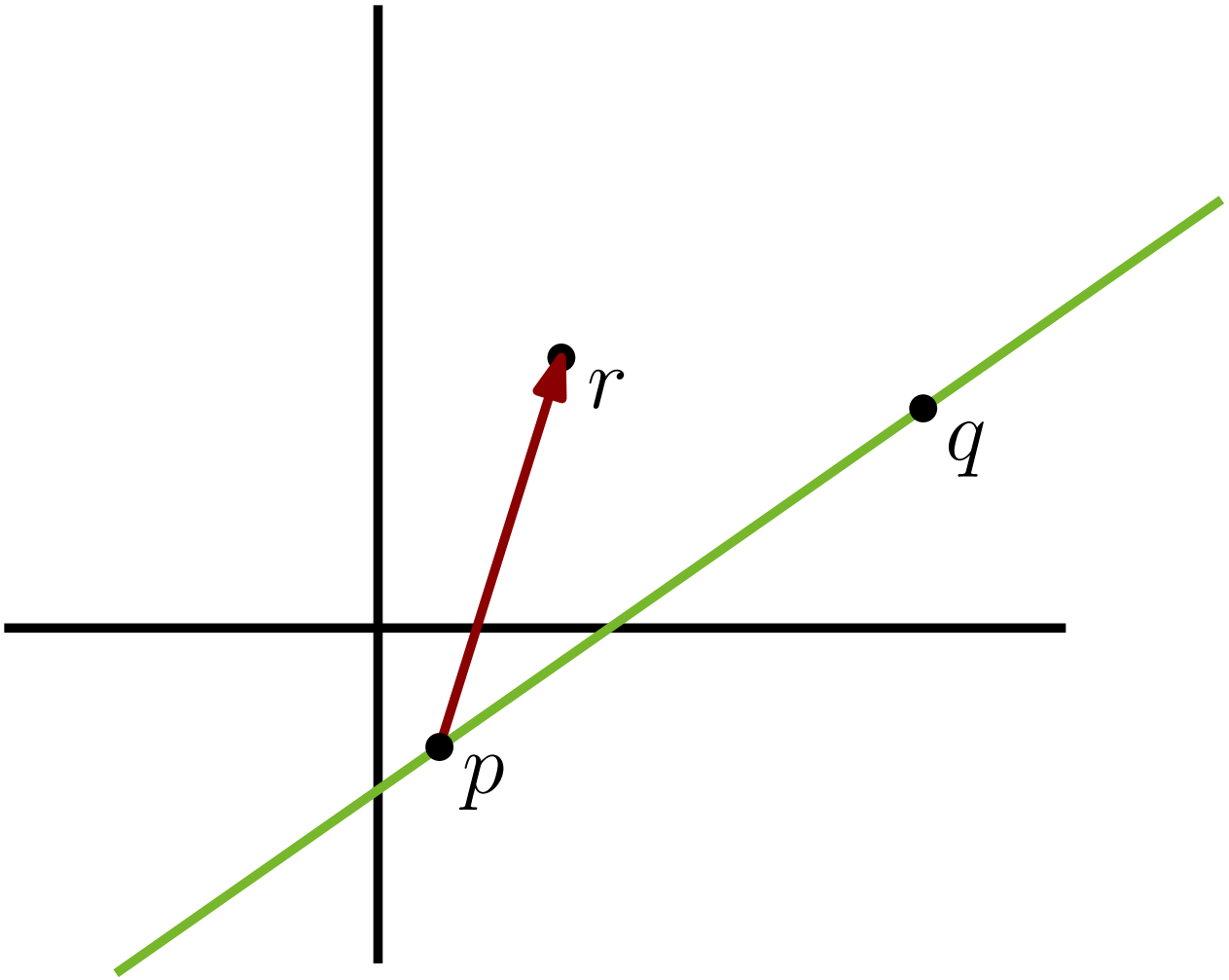
Aufgabe 1 - a)



Aufgabe 1 - a)

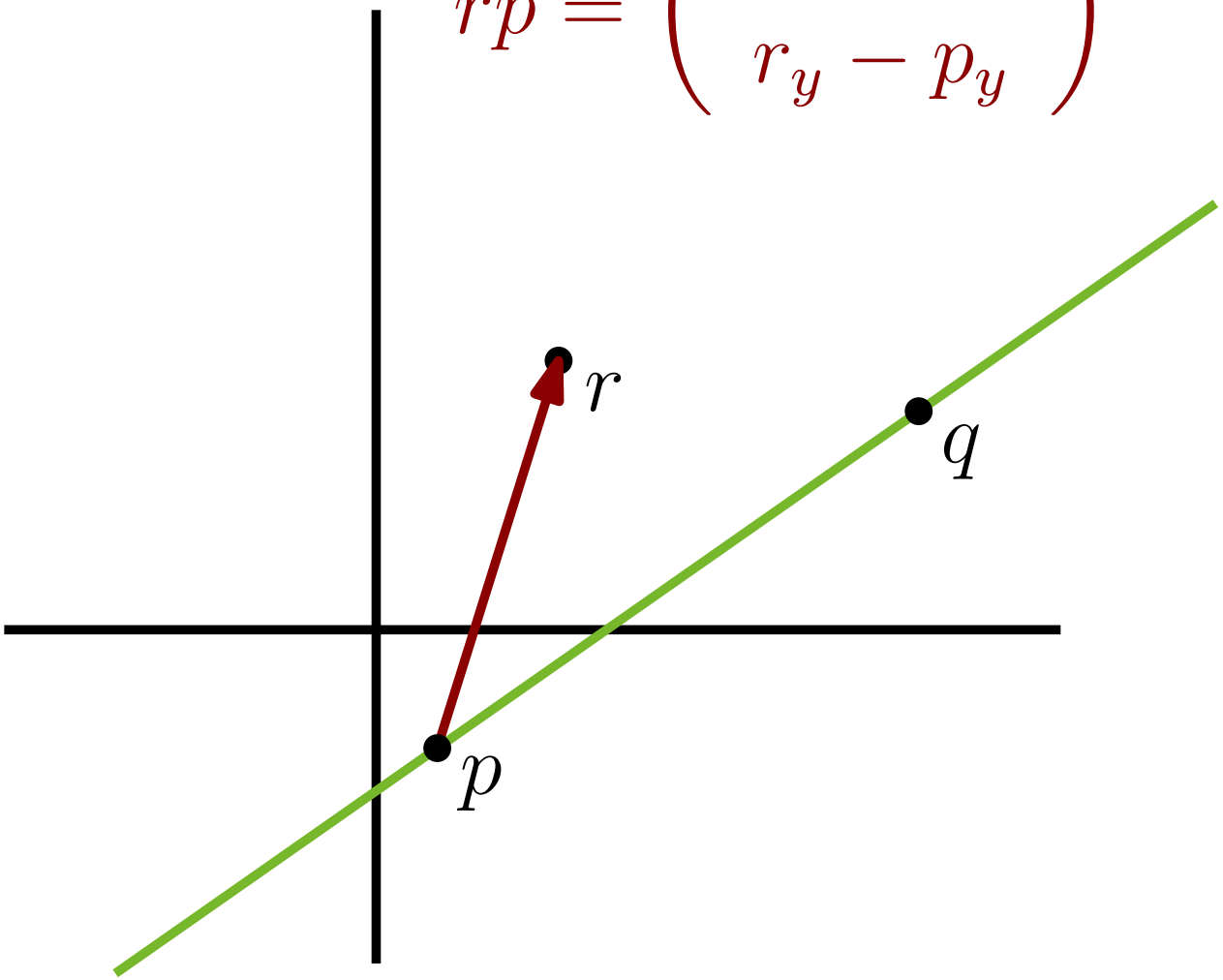


Aufgabe 1 - a)



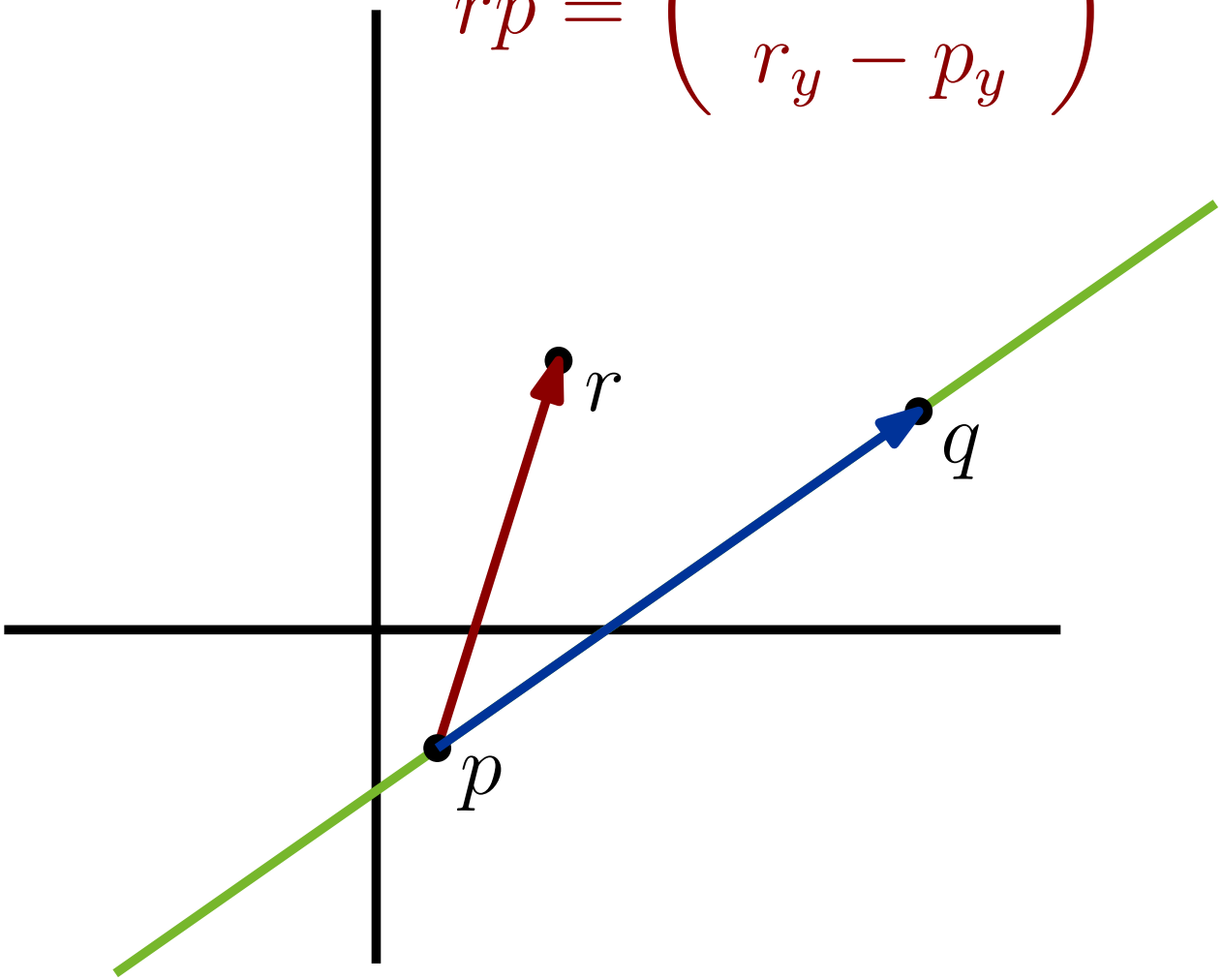
Aufgabe 1 - a)

$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix}$$



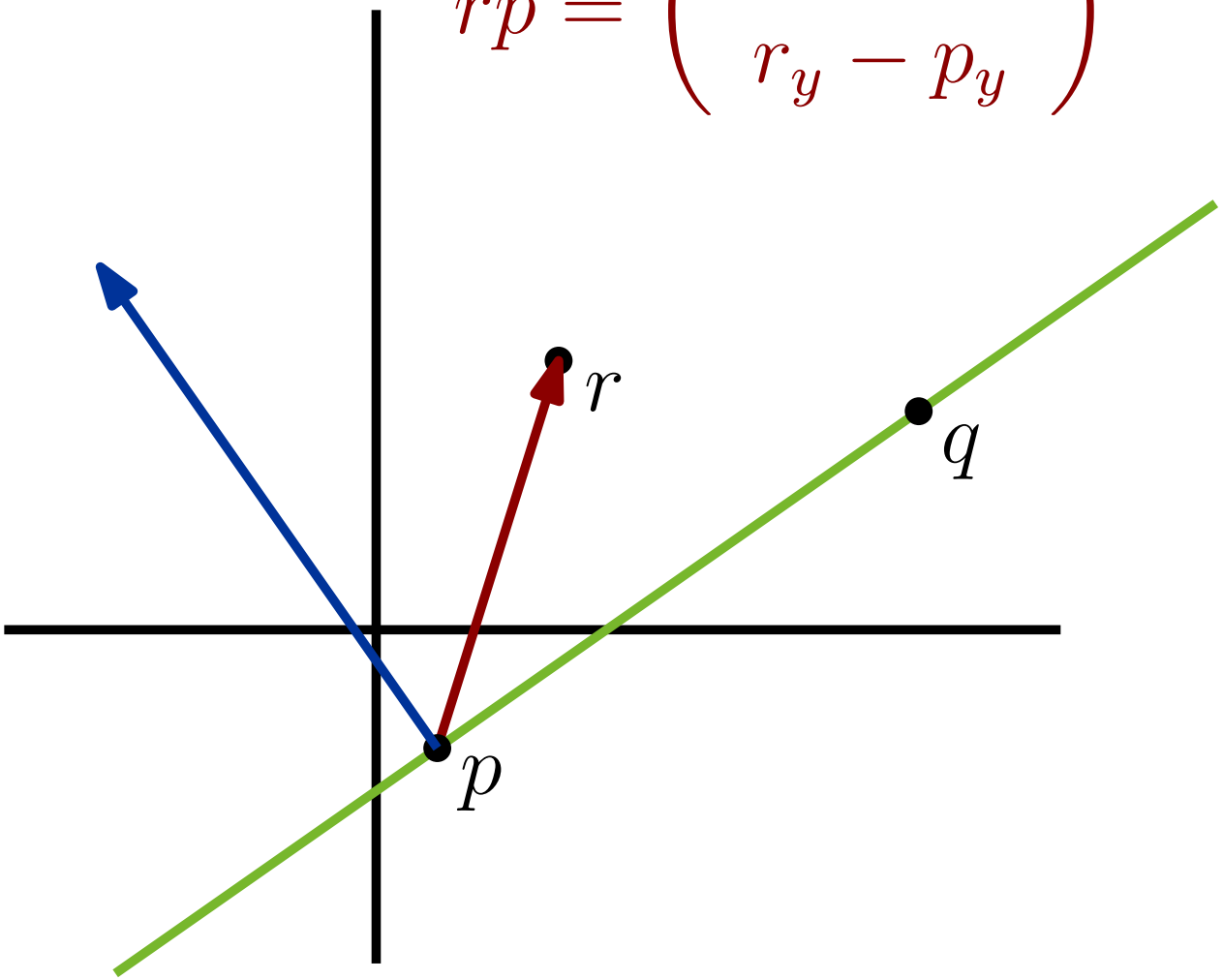
Aufgabe 1 - a)

$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix}$$



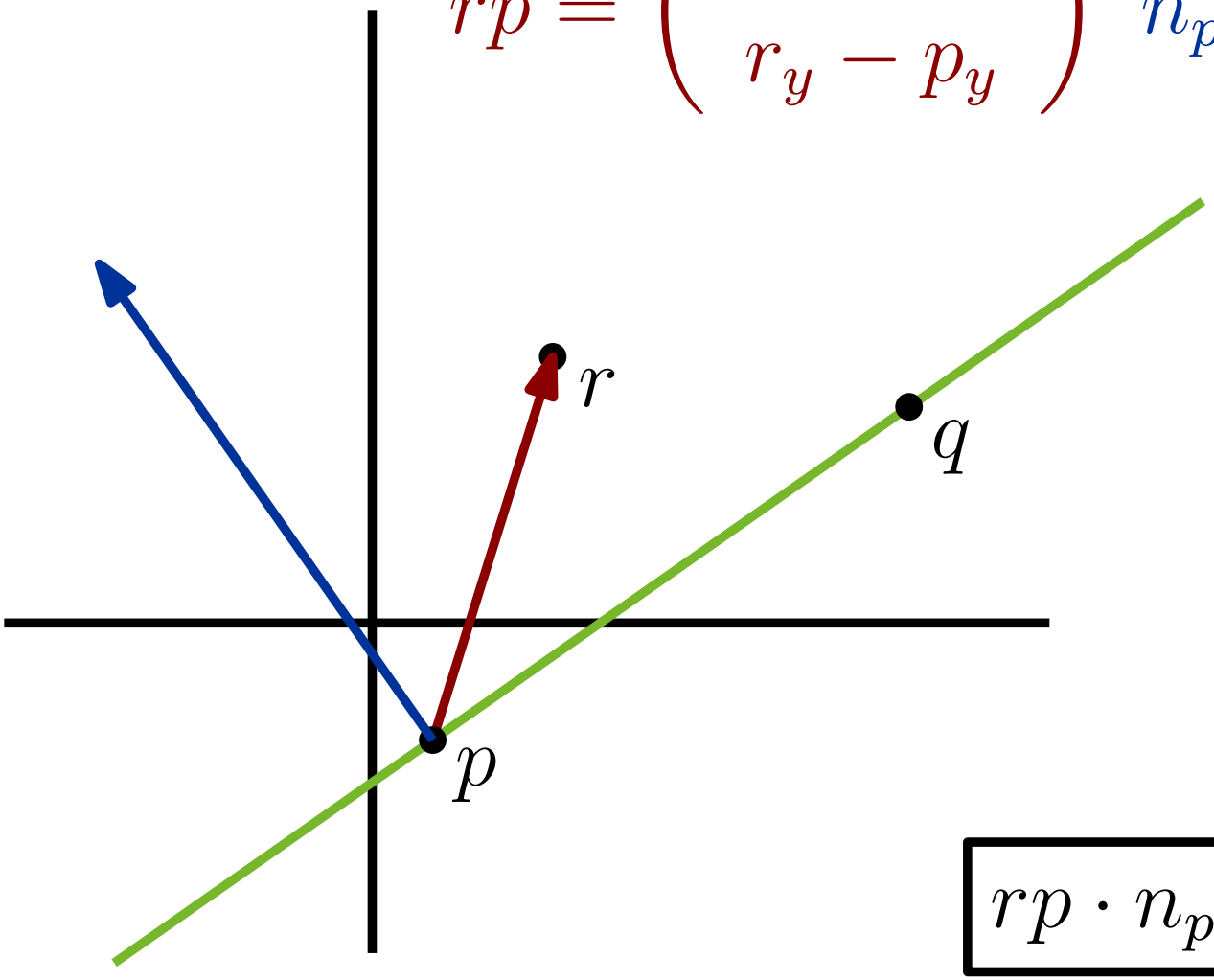
Aufgabe 1 - a)

$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix}$$



Aufgabe 1 - a)

$$rp = \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix} \quad n_{pq} = \begin{pmatrix} p_y - q_y \\ q_x - p_x \end{pmatrix}$$



$$rp \cdot n_{pq} = \det(A)$$

Aufgabe 1 - a)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

Aufgabe 1 - a)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

Aufgabe 1 - a) + b)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

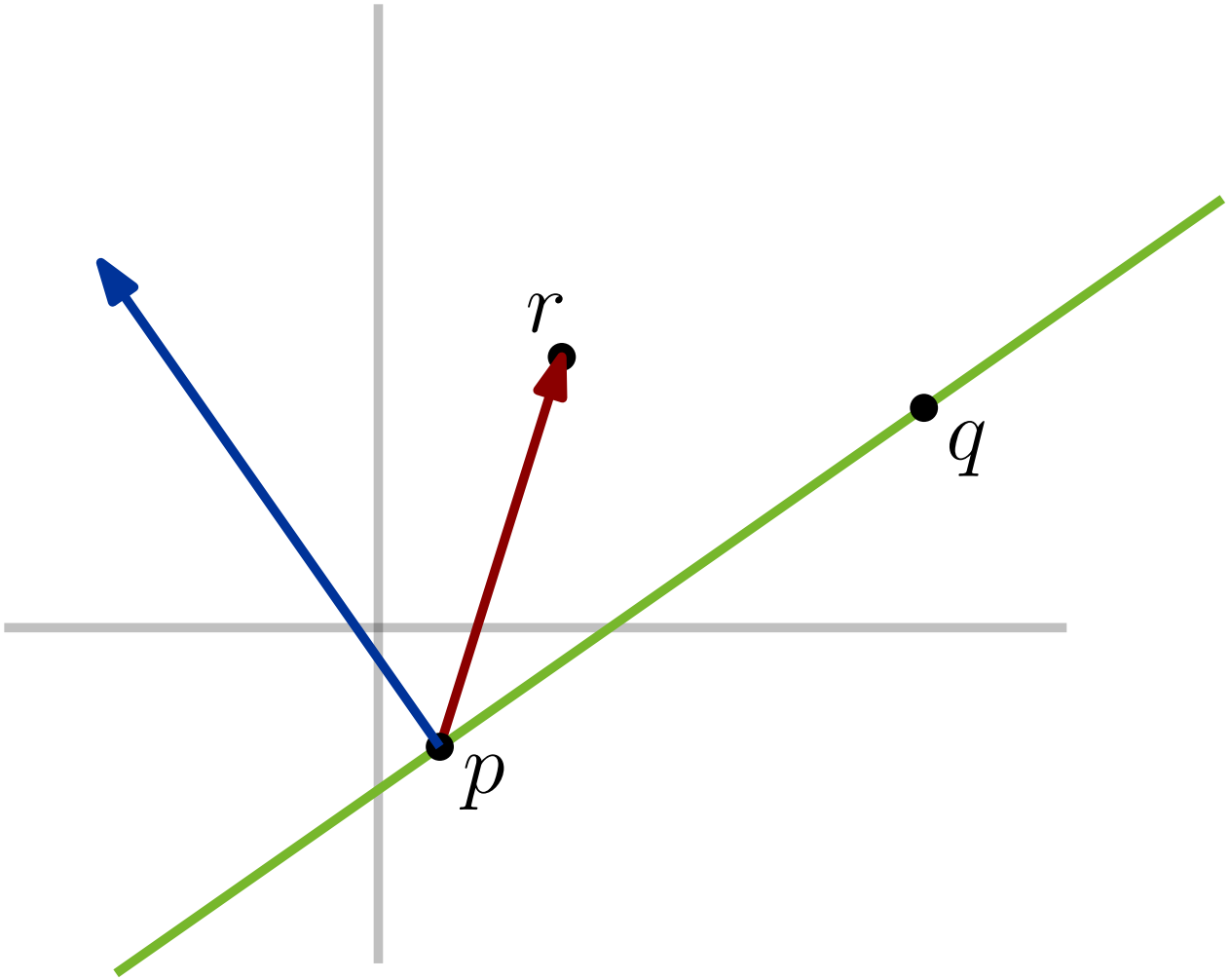
Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

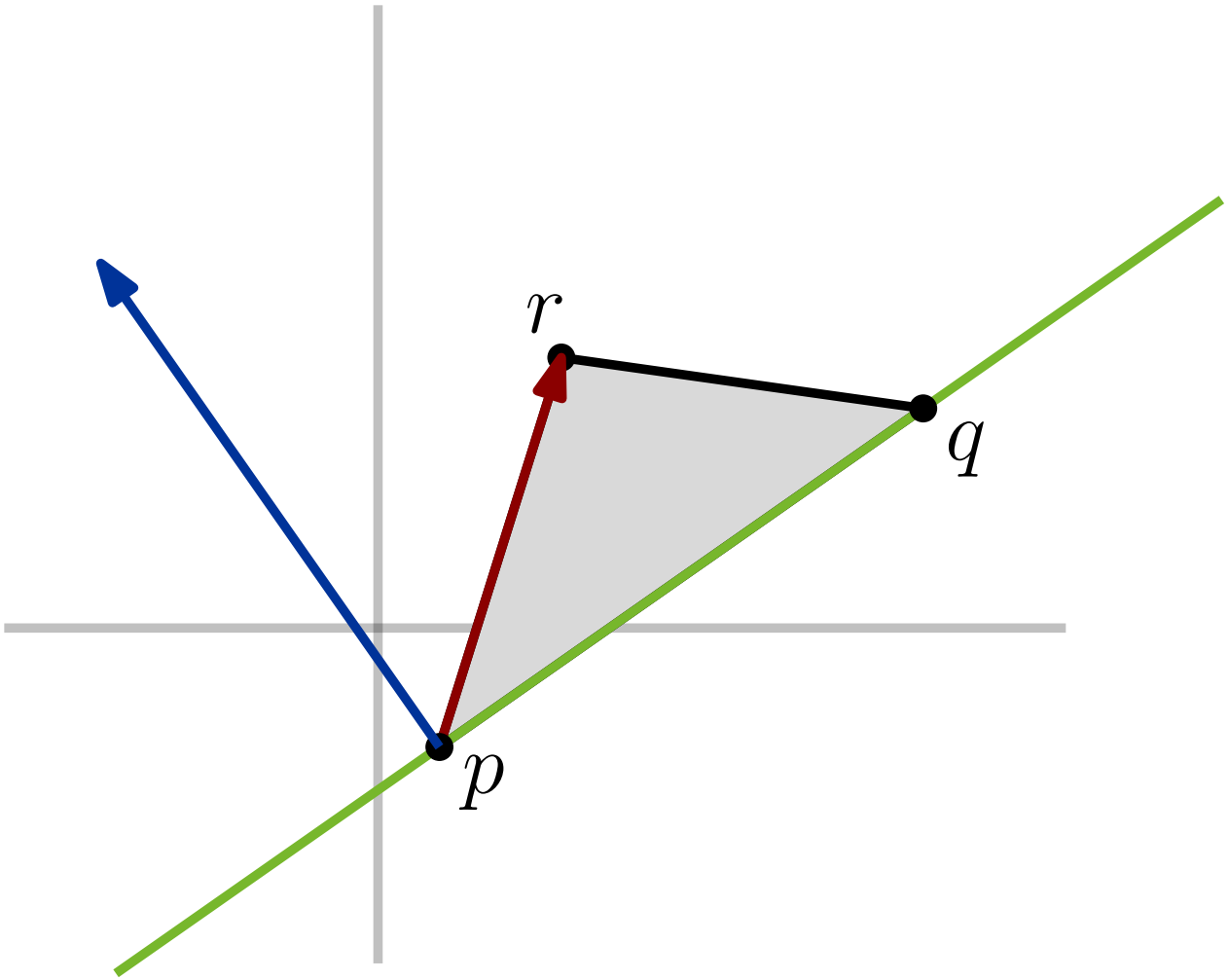
Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

Beweise: $|\det(A)| =$ doppelter Flächeninhalt des Dreiecks pqr

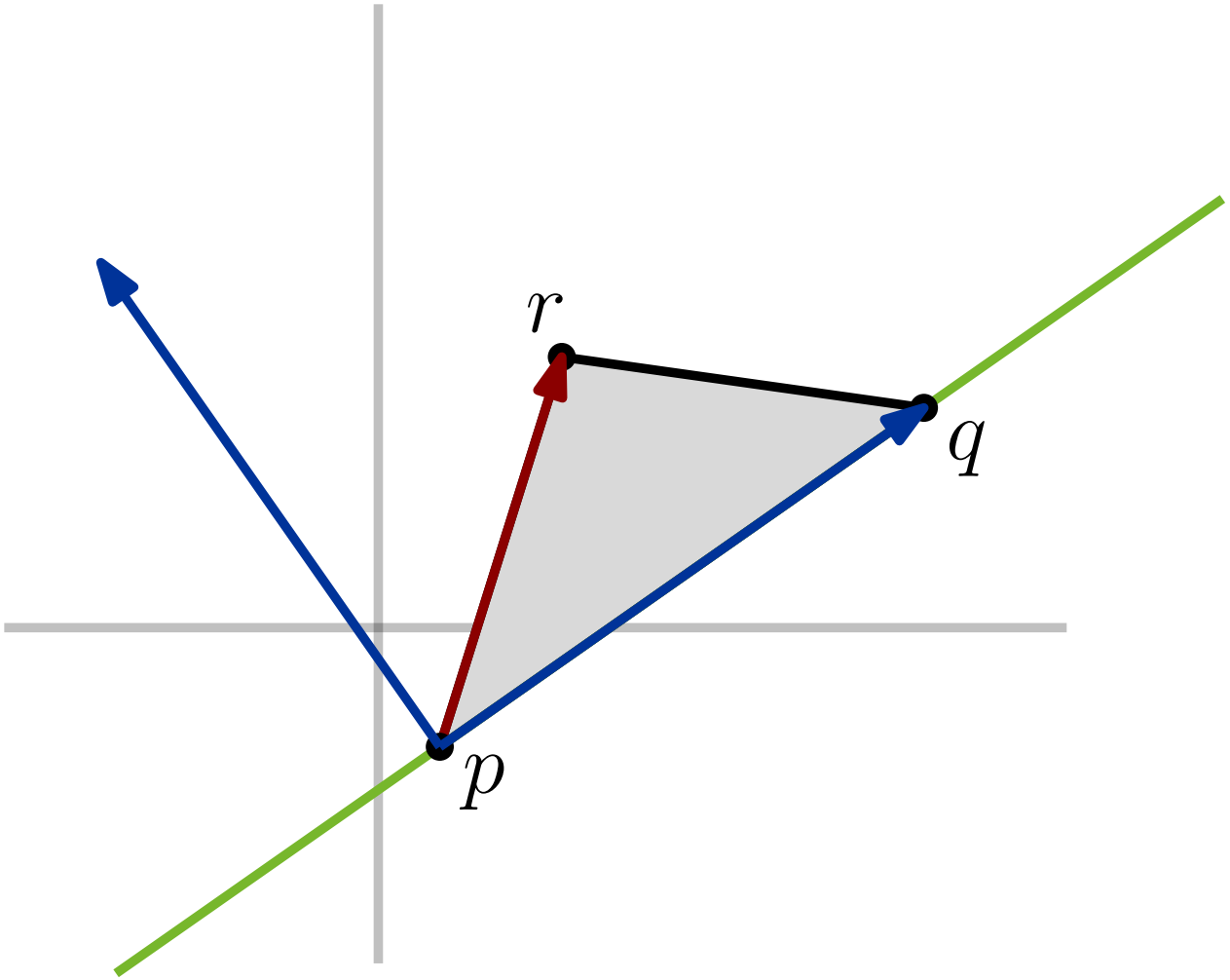
Aufgabe 1 - b)



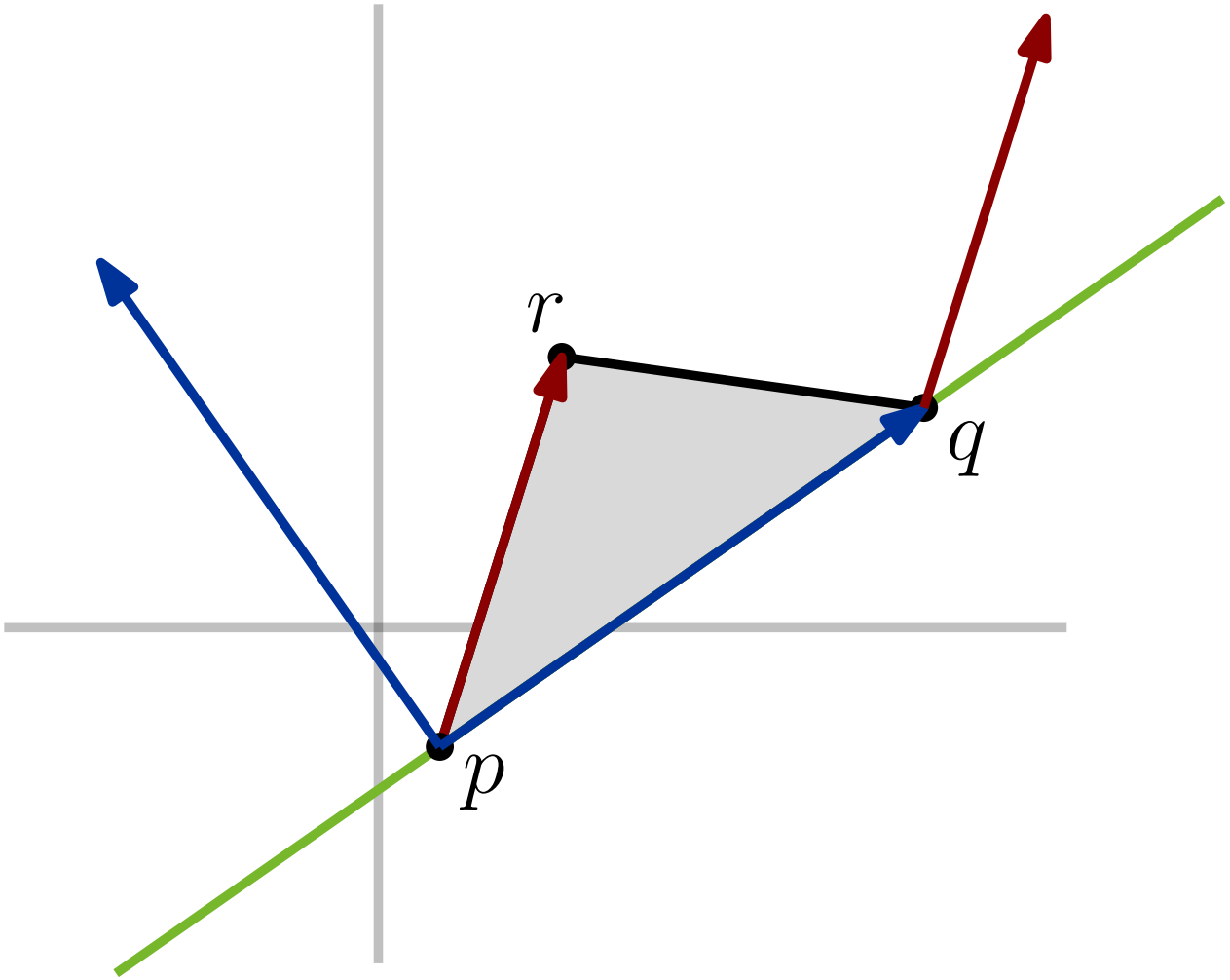
Aufgabe 1 - b)



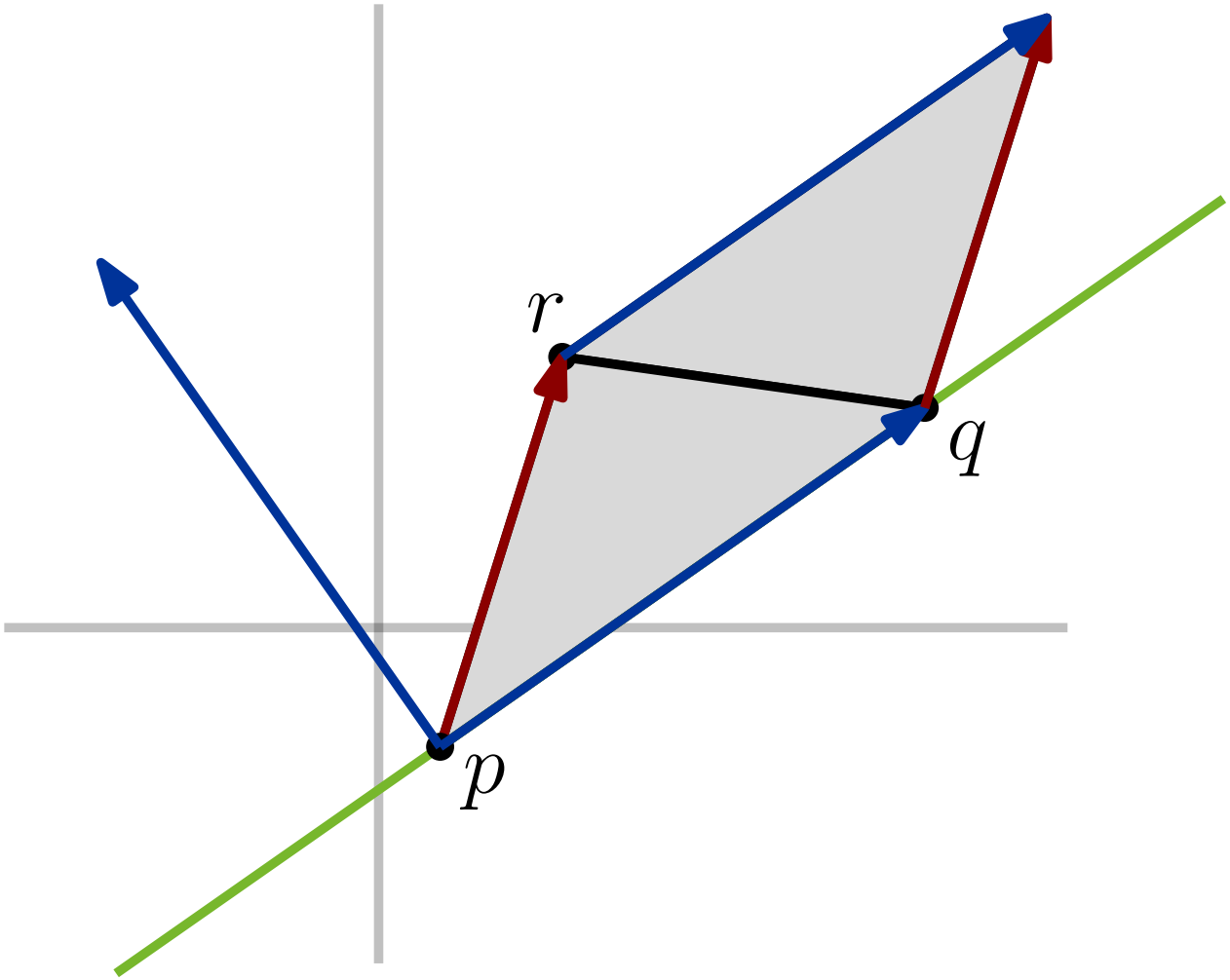
Aufgabe 1 - b)



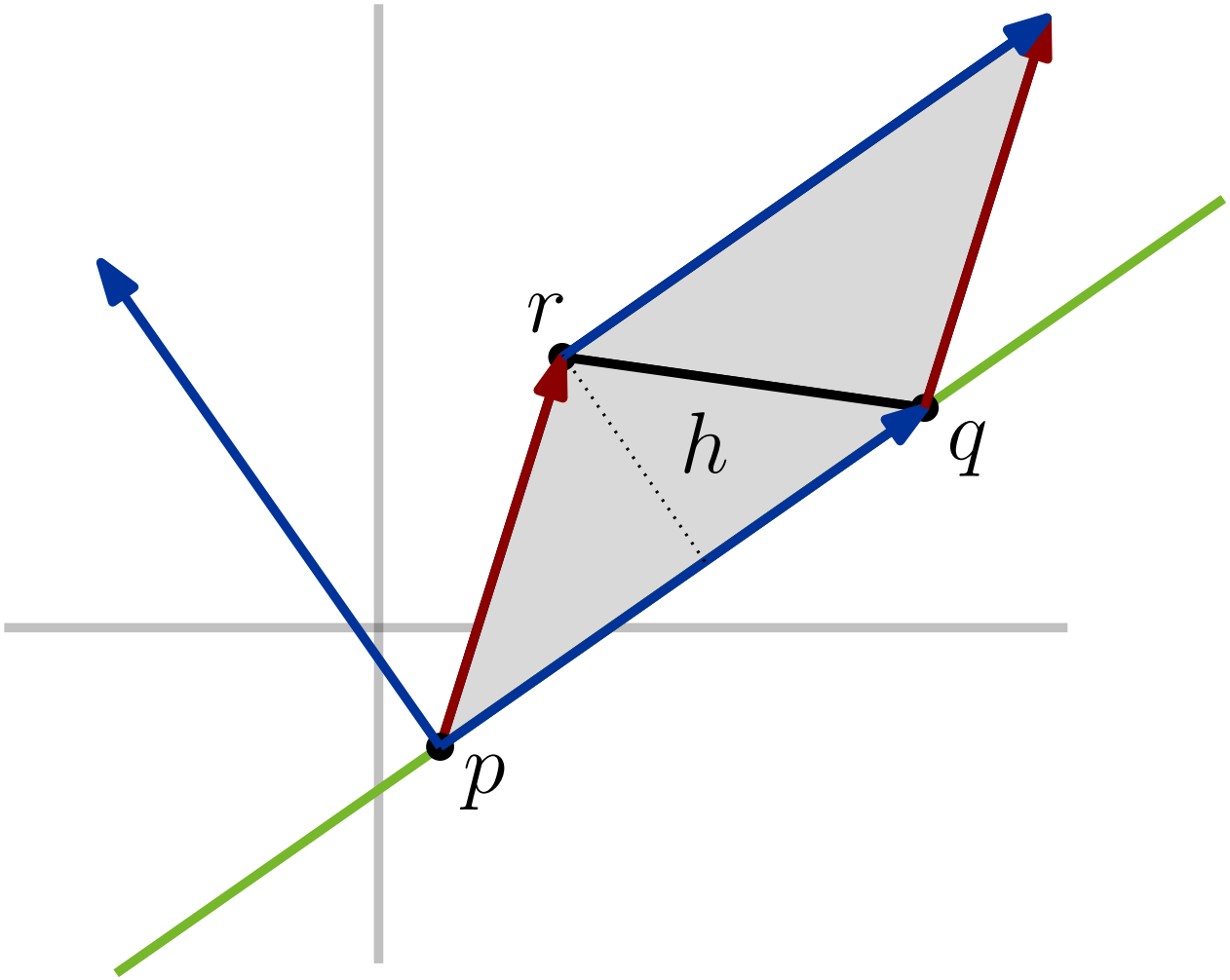
Aufgabe 1 - b)



Aufgabe 1 - b)

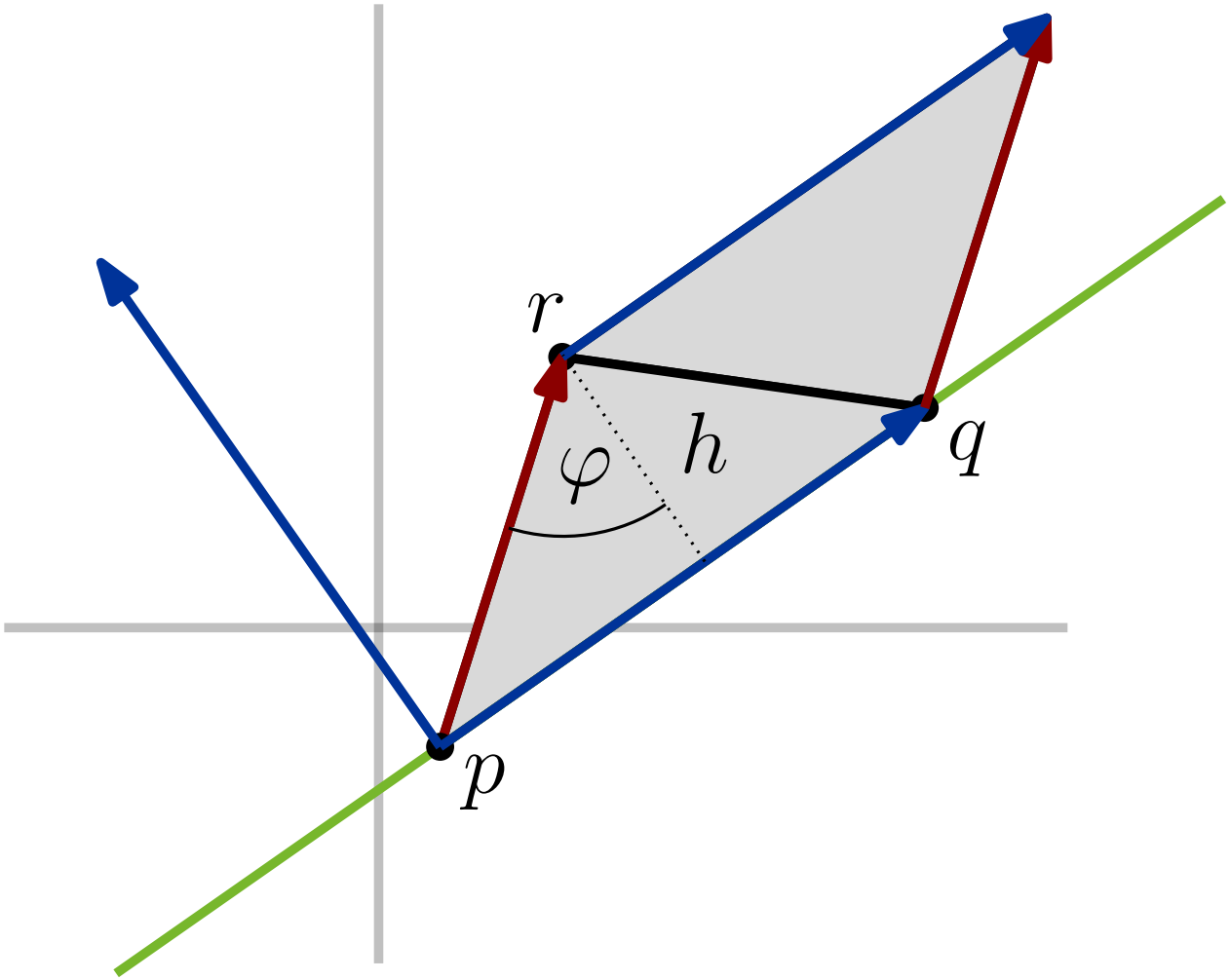


Aufgabe 1 - b)



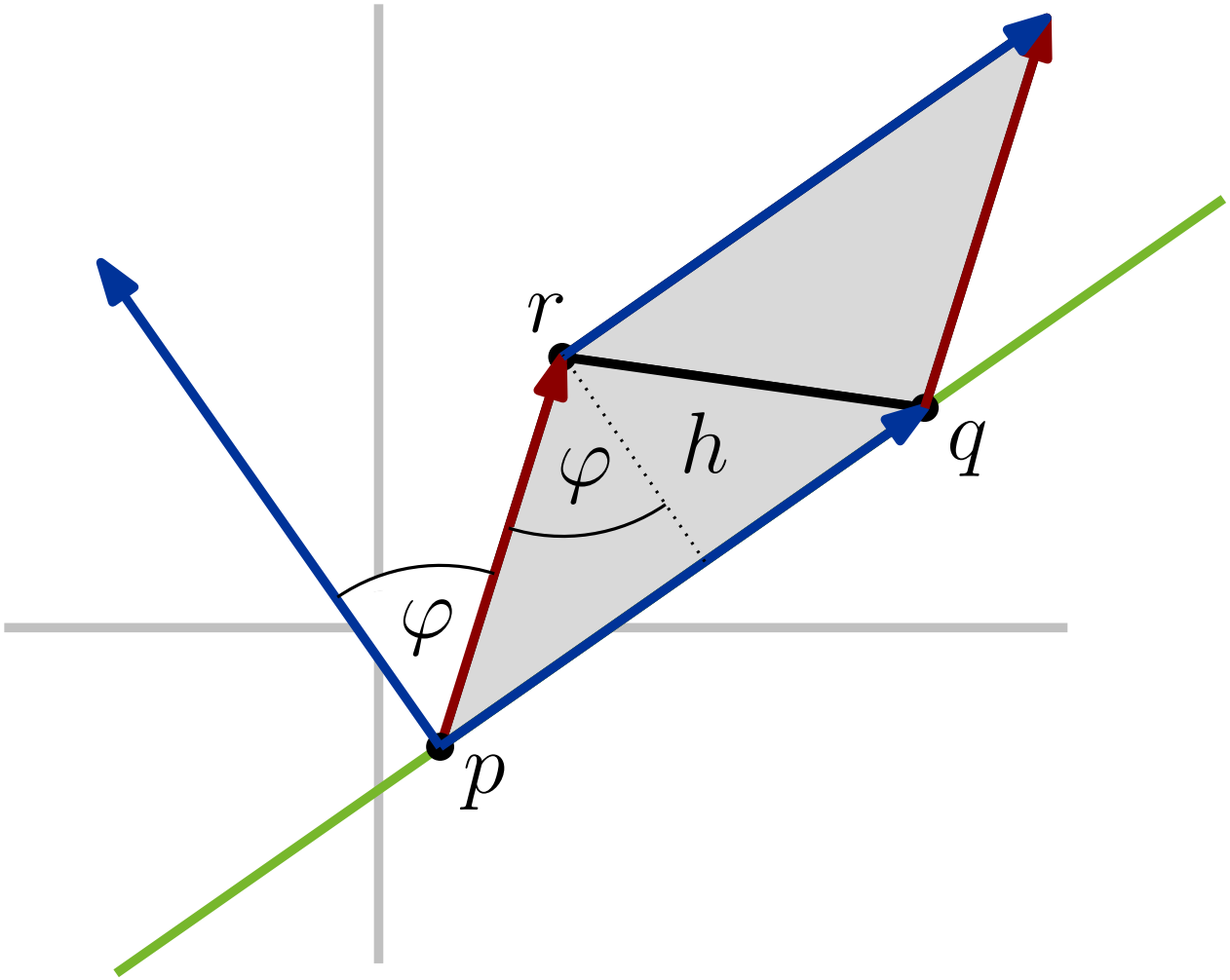
$$A_{pqr} = |pq| \cdot h$$

Aufgabe 1 - b)



$$A_{pqr} = |pq| \cdot h$$

Aufgabe 1 - b)



$$A_{pqr} = |pq| \cdot h$$

Aufgabe 1 - a) + b)

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

Beweise: $|\det(A)| =$ doppelter Flächeninhalt des Dreiecks pqr

Aufgabe 1 - a) + b)

Karlsruhe Institute of Technology

Gerichtete Gerade g durch $p = (p_x, p_y)$ und $q = (q_x, q_y)$

Punkt $r = (r_x, r_y)$

$$A = \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}$$

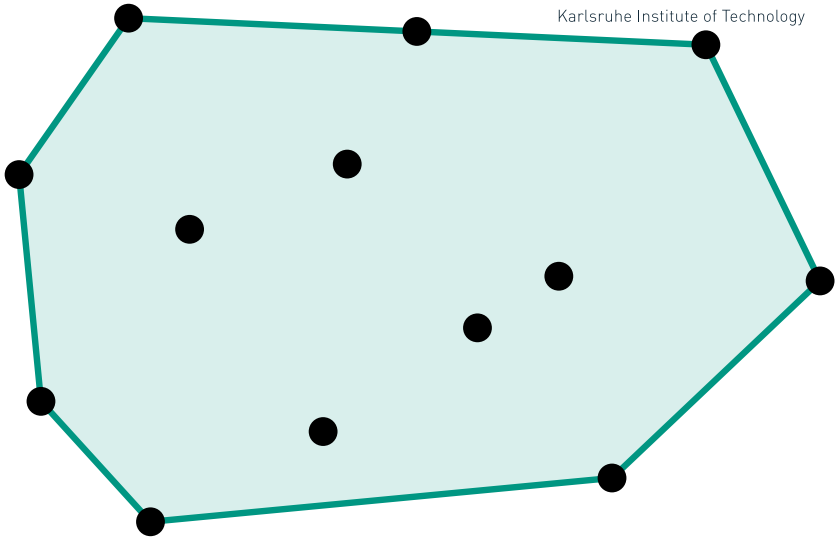
Beweise: Vorzeichen von $\det(A)$ zeigt an ob r links oder rechts von g liegt.

Beweise: $|\det(A)| =$ doppelter Flächeninhalt des Dreiecks pqr

Aufgabe 2

Lemma:

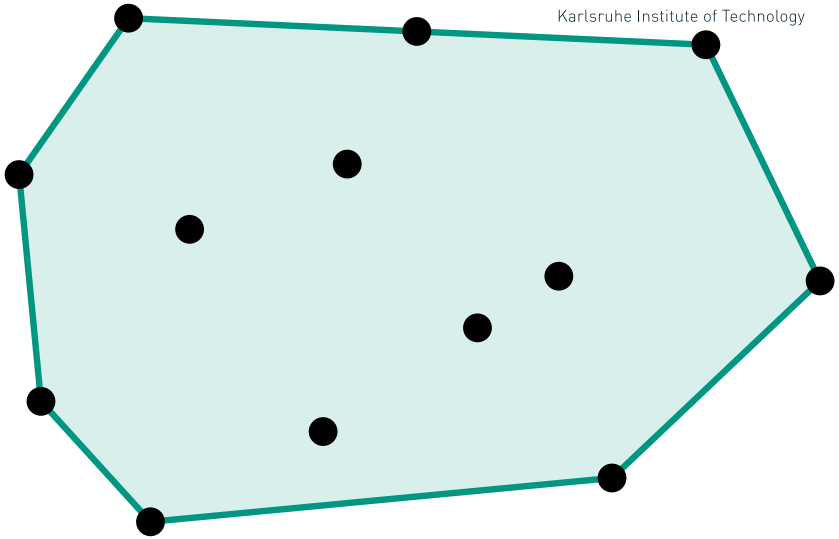
Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.

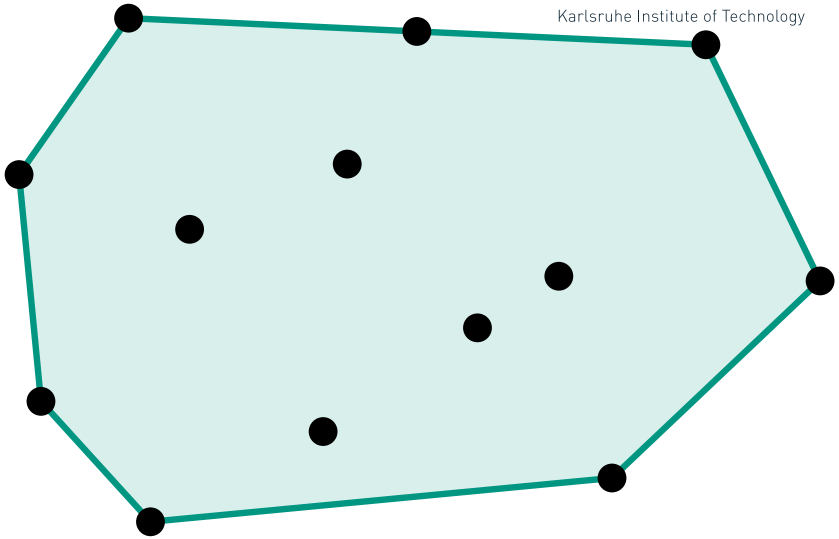


Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



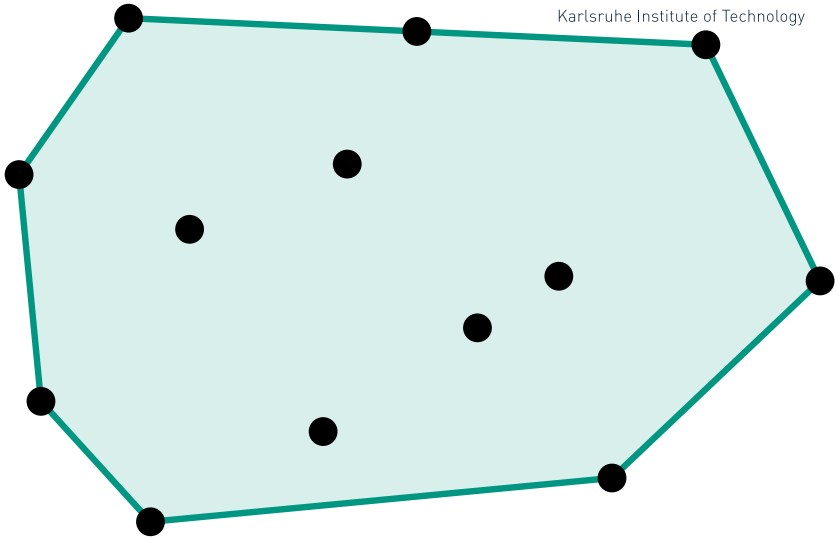
Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



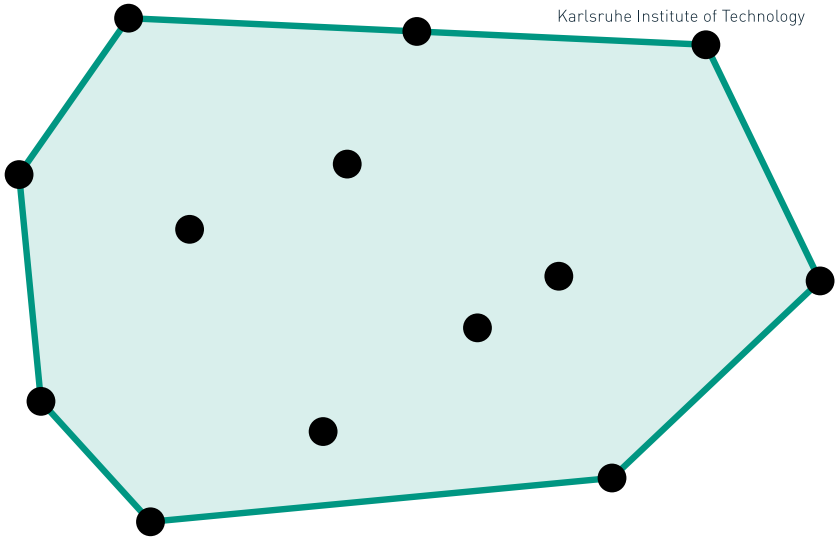
Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

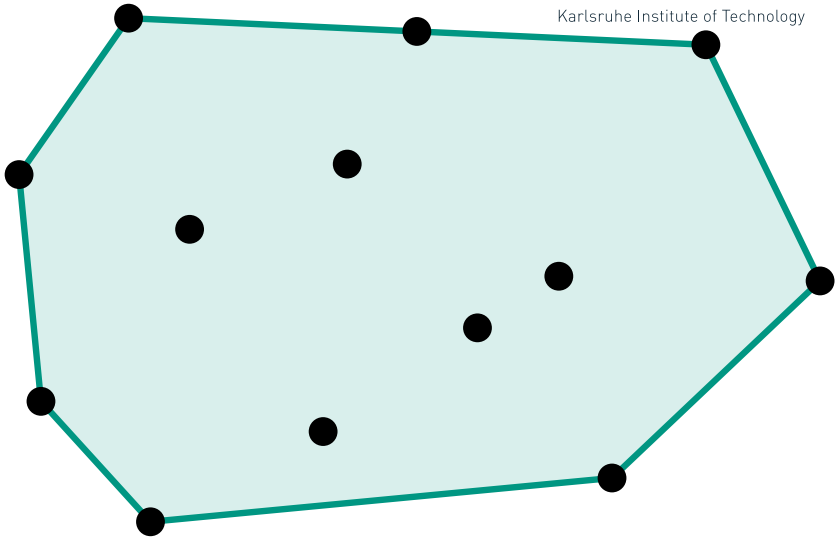
b) Schnitt zweier konvexen Mengen ist konvex.

Def: Eine Menge $P \subseteq \mathbb{R}^2$ heißt **konvex**, wenn für je zwei Punkte $p, q \in P$ auch gilt $\overline{pq} \in P$.

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

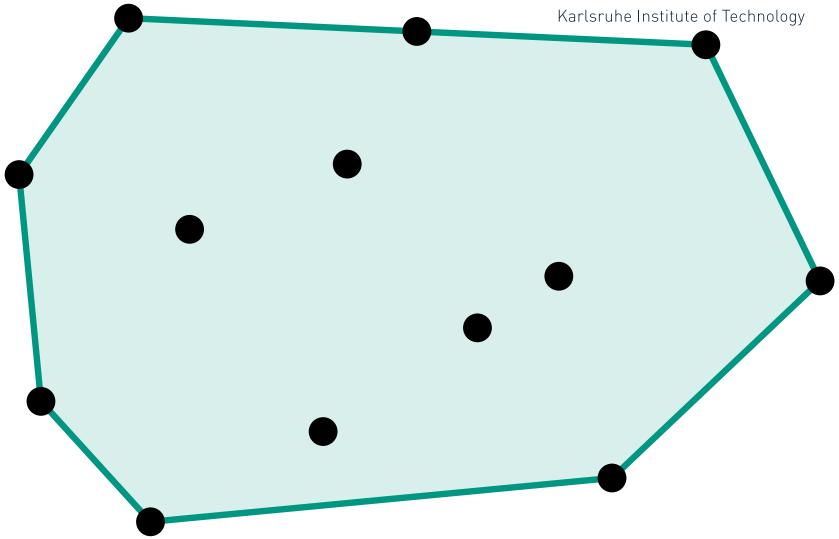
a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

b) Schnitt zweier konvexen Mengen ist konvex.

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

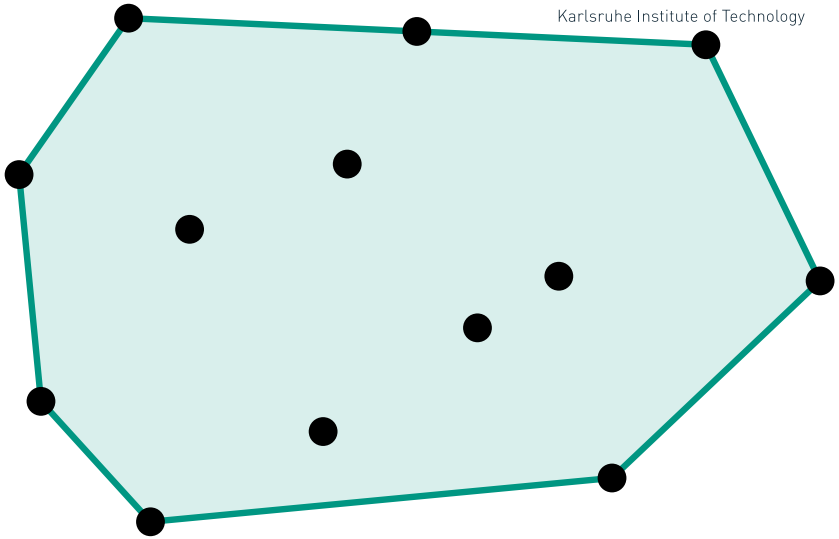
a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

b) Schnitt zweier konvexen Mengen ist konvex.

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

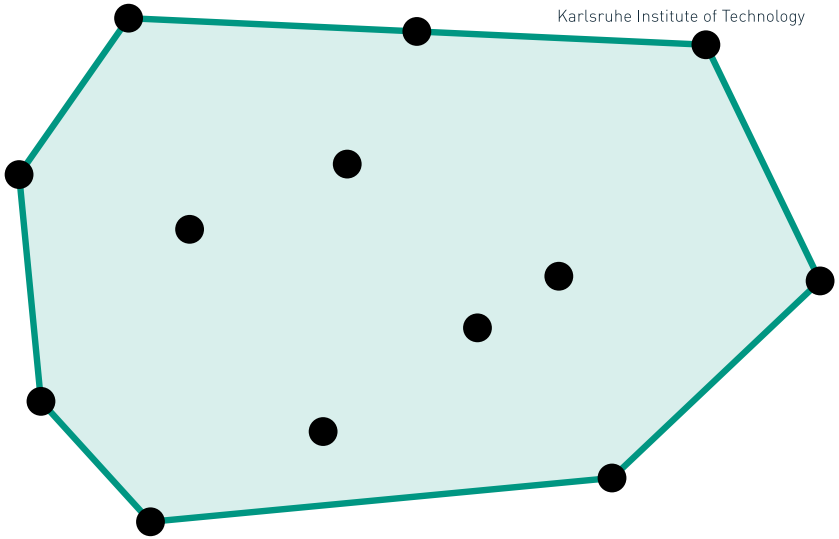
b) Schnitt zweier konvexen Mengen ist konvex.

c) $CH(P)$ ist ein Polygon.

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

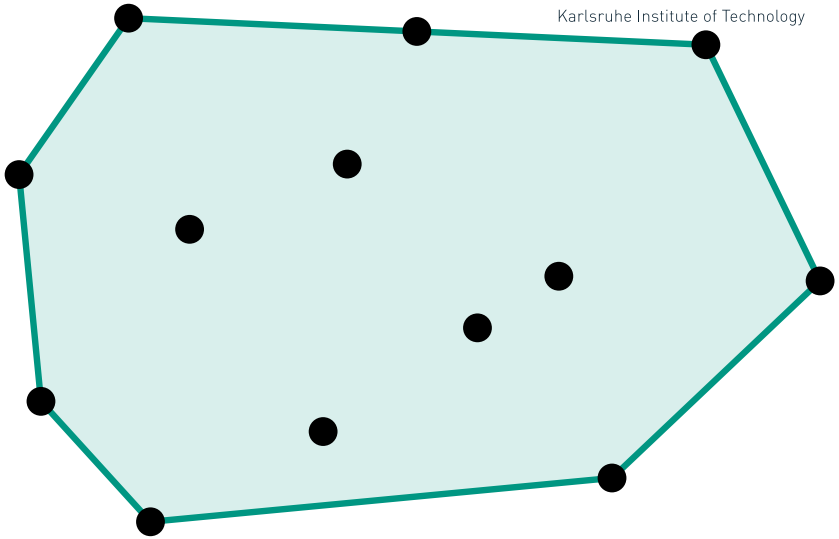
b) Schnitt zweier konvexen Mengen ist konvex.

c) $CH(P)$ ist ein Polygon.

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

b) Schnitt zweier konvexen Mengen ist konvex.

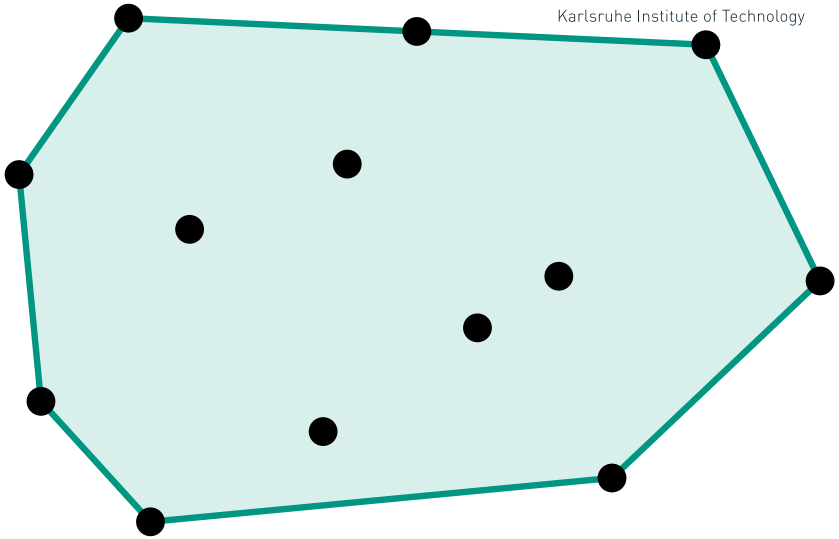
c) $CH(P)$ ist ein Polygon.

d) Alle Ecken von $CH(P)$ liegen in P.

Aufgabe 2

Lemma:

Für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ ist $CH(P)$ ein konvexes Polygon, das P enthält und dessen Ecken in P liegen.



Mathematik: $CH(P) = \bigcap_{C \supseteq P:\ C \text{ konvex}} C$

a) Die konvexe Hülle $CH(P)$ enthält alle Punkte aus P.

b) Schnitt zweier konvexen Mengen ist konvex.

c) $CH(P)$ ist ein Polygon.

d) Alle Ecken von $CH(P)$ liegen in P.

Aufgabe 3

Karlsruhe Institute of Technology

a) Berechnung konvexer Hülle benötigt $\Omega(n \log n)$.

Aufgabe 3

a) Berechnung konvexer Hülle benötigt $\Omega(n \log n)$.

b) Gegeben: einfaches [nicht notwendigerweise kovexes] Polygon. Konstruiere daraus in $\mathcal{O}(n)$ seine konvexe Hülle. Warum ist das kein Widerspruch zu a) ?

Aufgabe 3

a) Berechnung konvexer Hülle benötigt $\Omega(n \log n)$.

b) Gegeben: einfaches [nicht notwendigerweise kovexes] Polygon. Konstruiere daraus in $\mathcal{O}(n)$ seine konvexe Hülle. Warum ist das kein Widerspruch zu a) ?

UpperConvexHull(P)
  $\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sortiere P von links nach rechts
  $L \leftarrow \langle p_1, p_2 \rangle$
  for $i \leftarrow 3$ **to** n **do**
    L.append(p_i)
    while $|L| > 2$ **and** letzte 3 Punkte in L kein Rechtsknick **do**
      entferne vorletzten Punkt aus L
  return L

Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

while true **do**

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

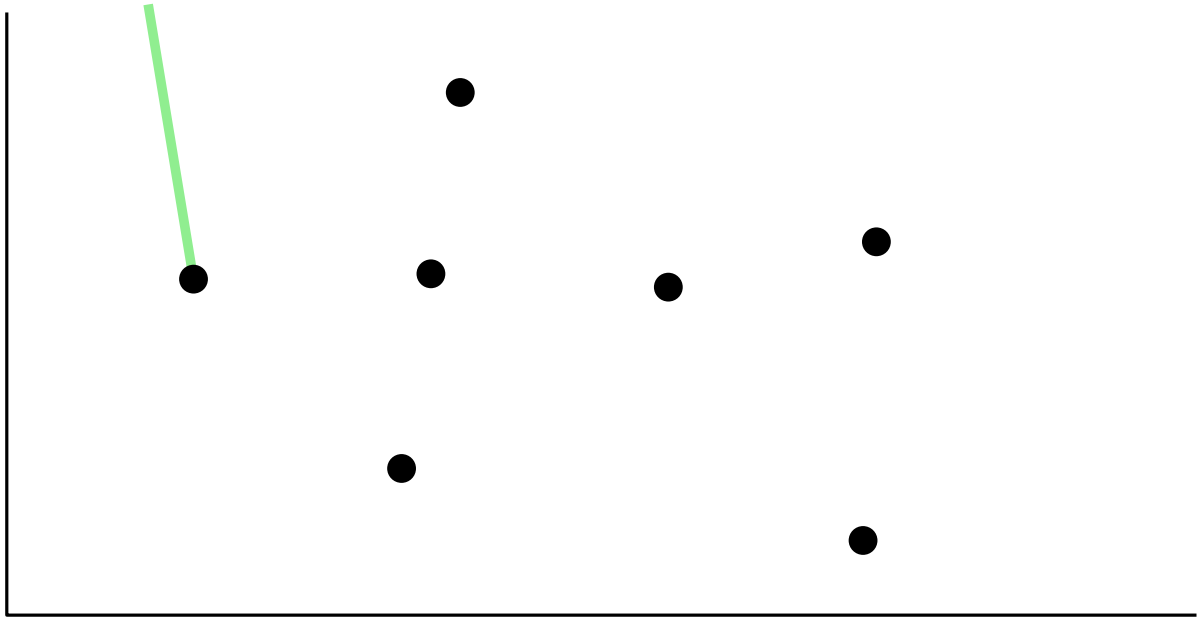
    break

return $(p_1, \ldots, p_h)$

Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P
while true **do**
    Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.
    if $p_{k+1} == p_1$ **then**
        break
return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

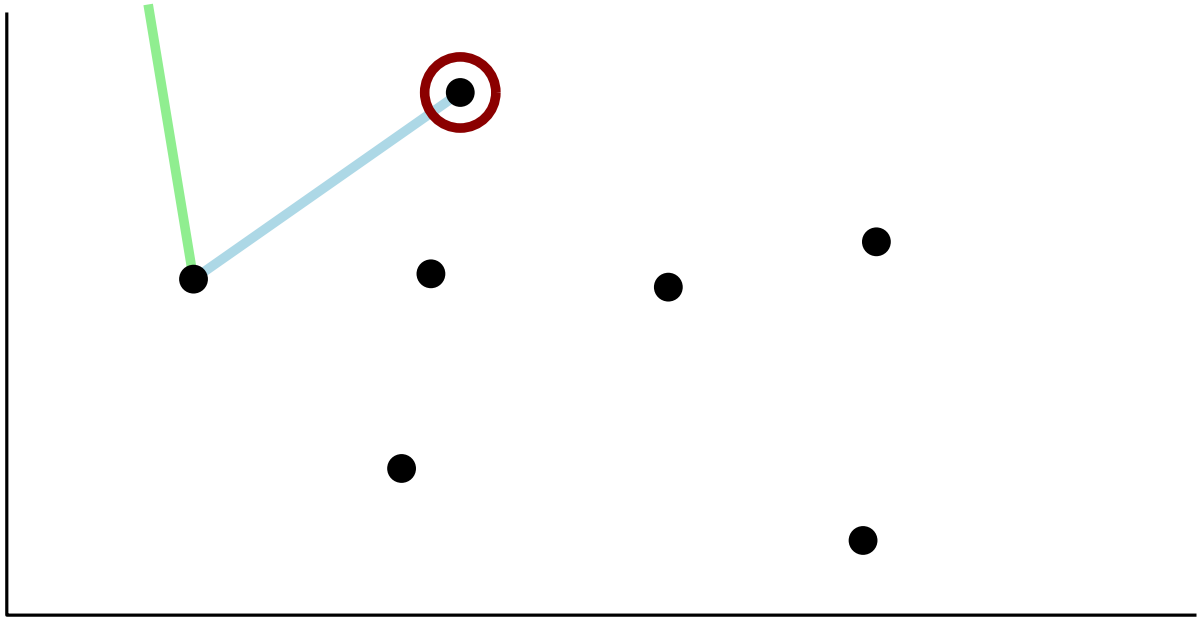
while true **do**

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

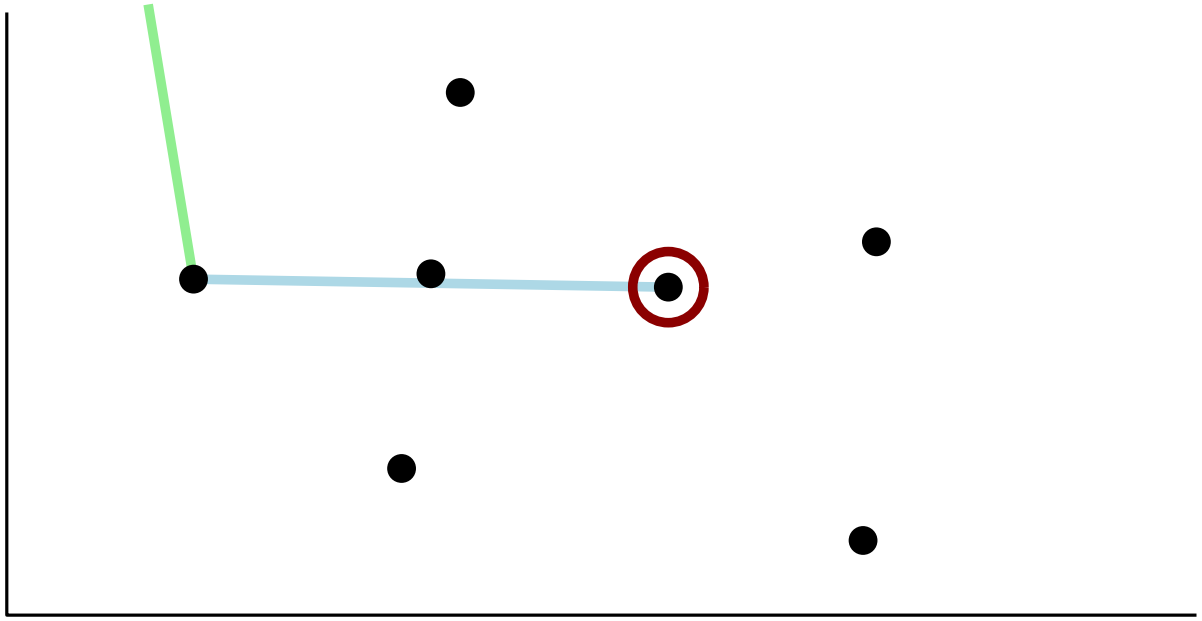
while true **do**

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

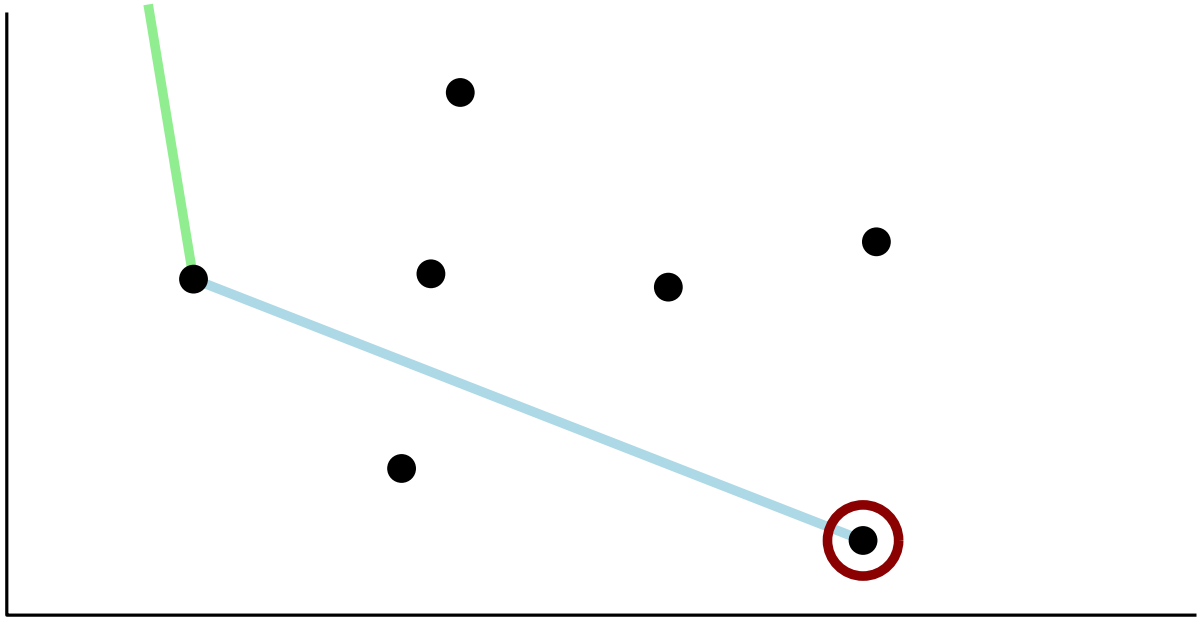
while true **do**

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

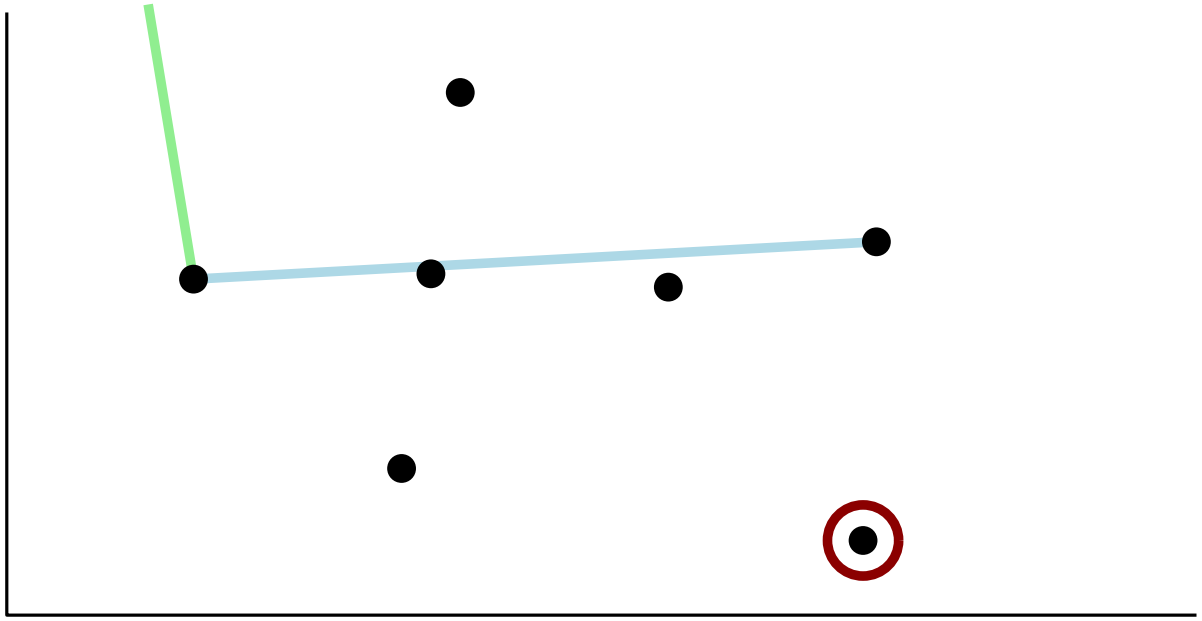
while true **do**

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

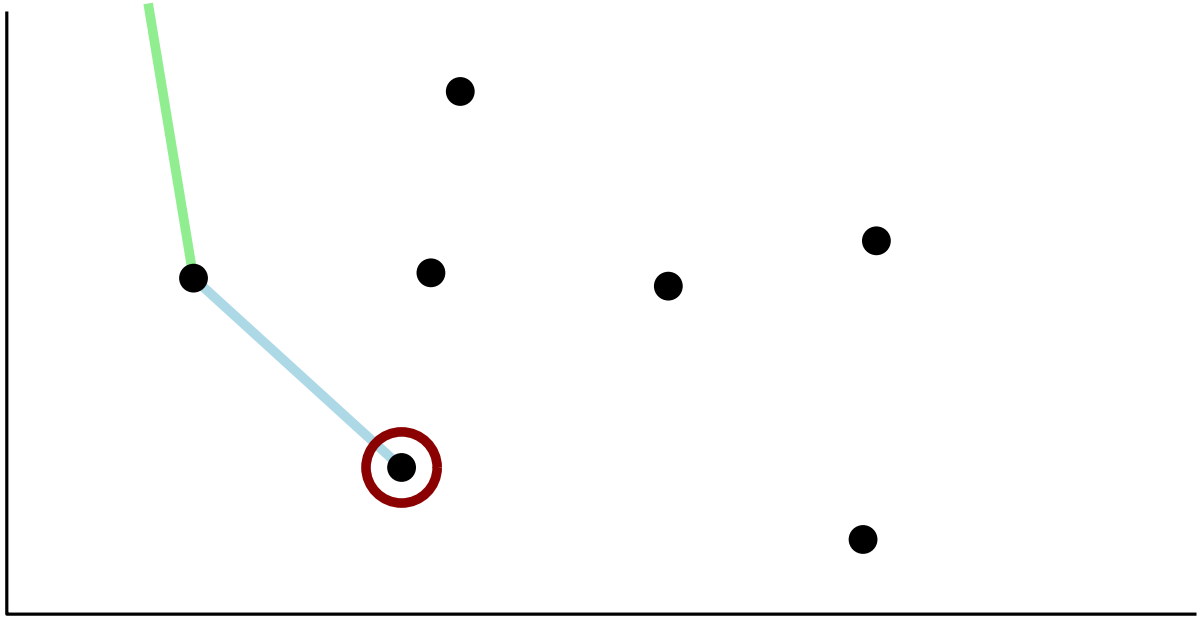
while true **do**

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

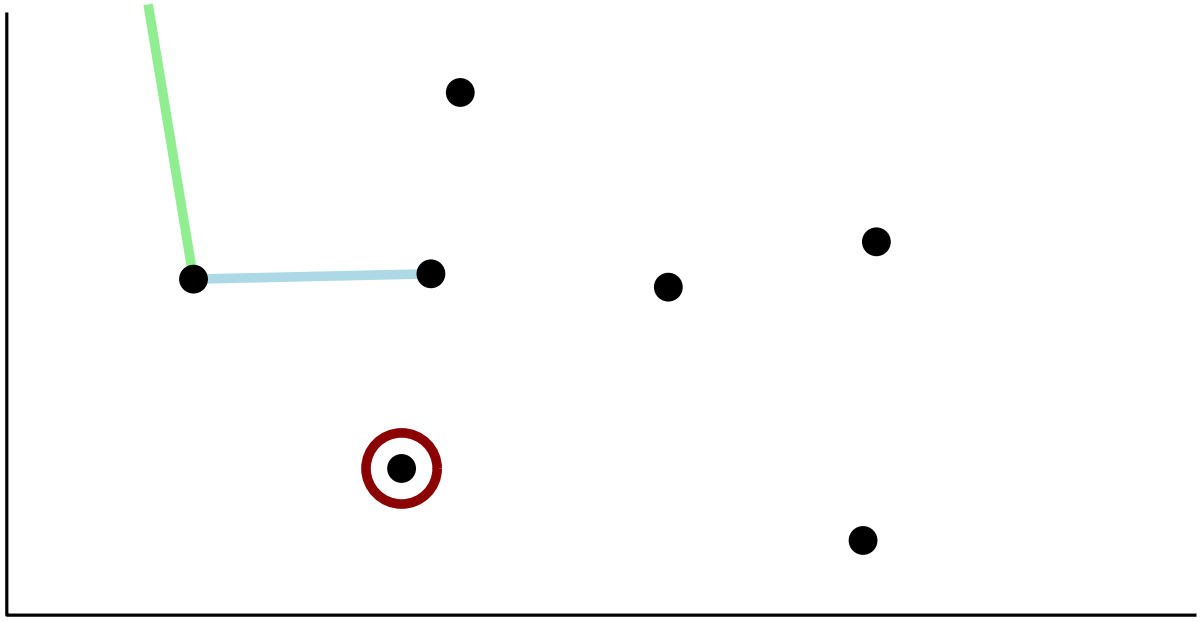
while true **do**

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

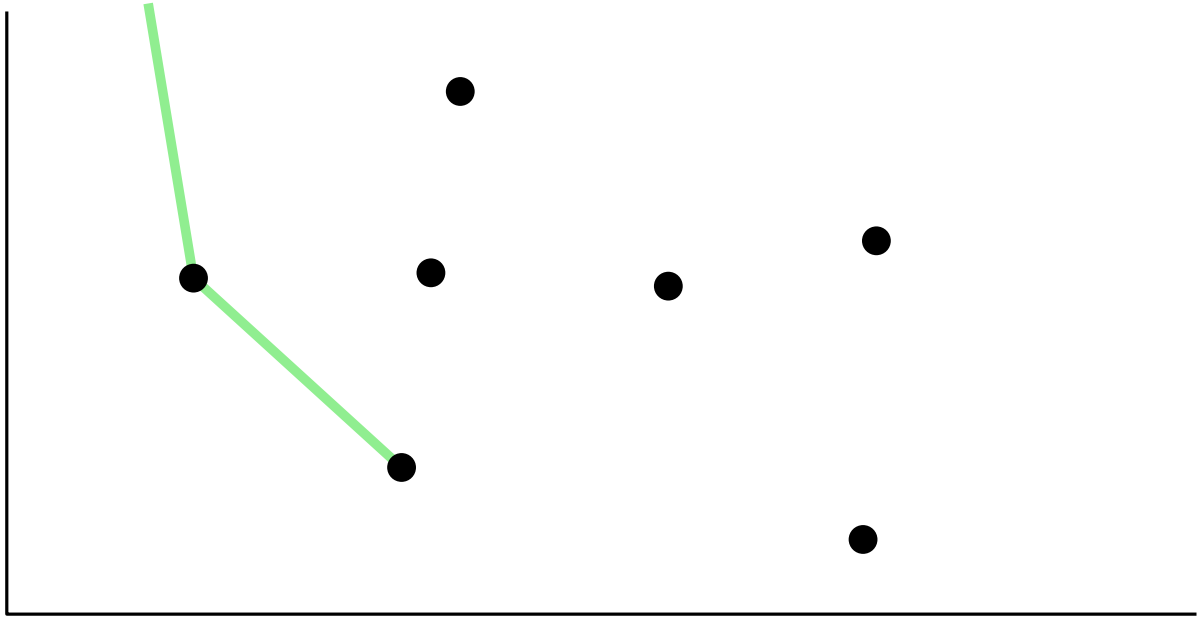
while true **do**

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

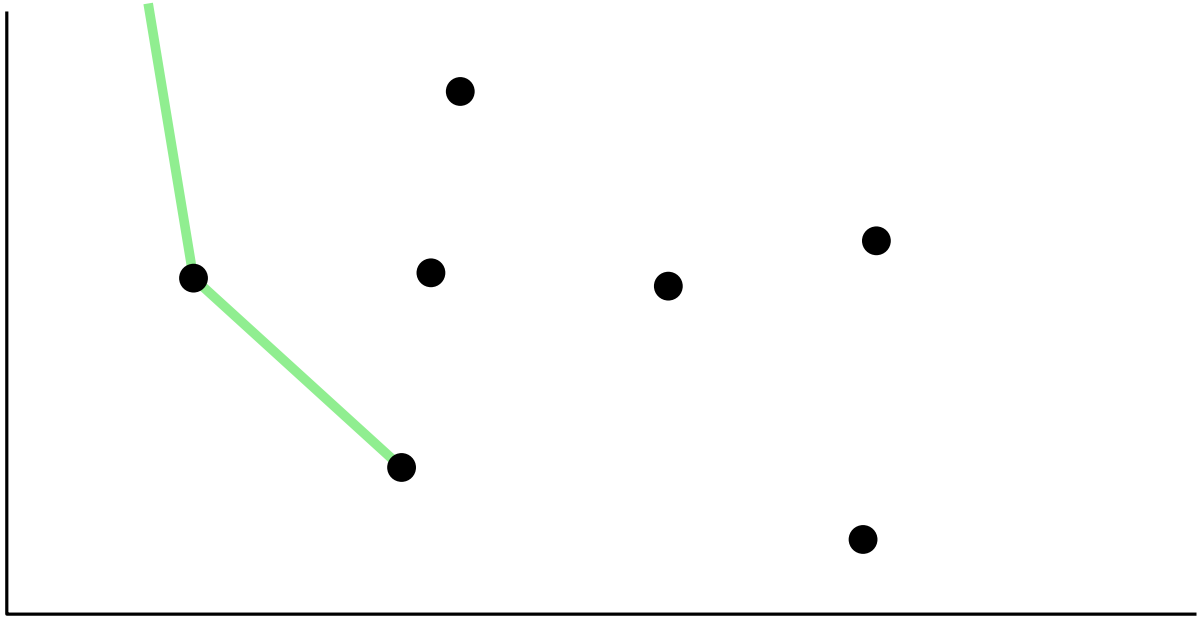
while true **do**

Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

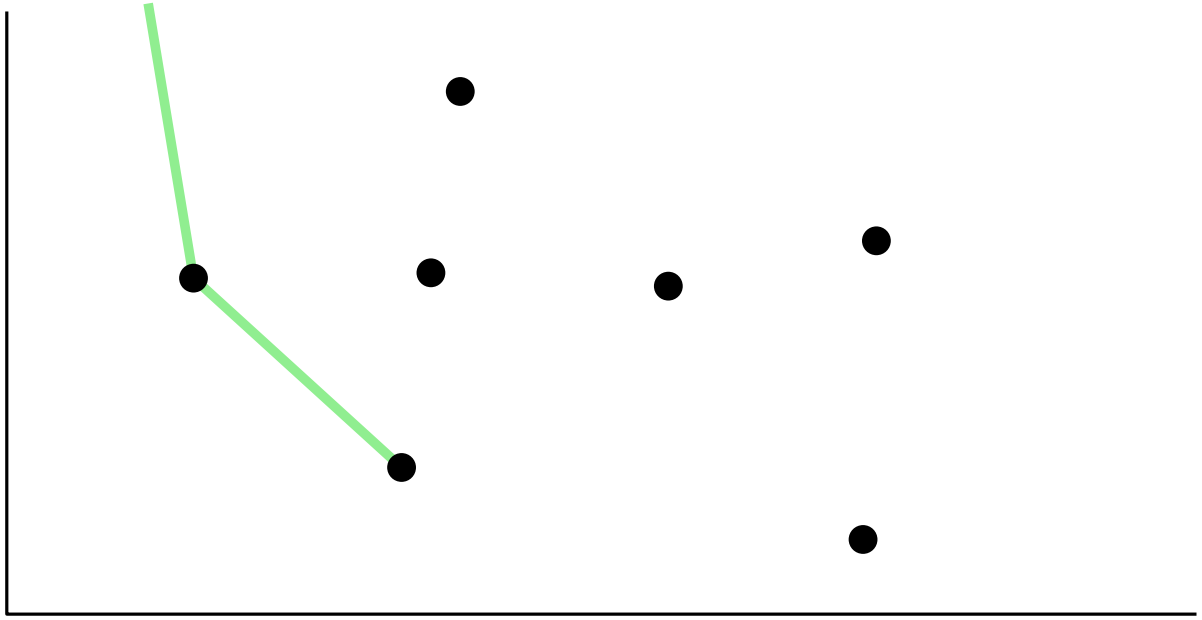
while true **do** $\mathcal{O}(n)$

  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

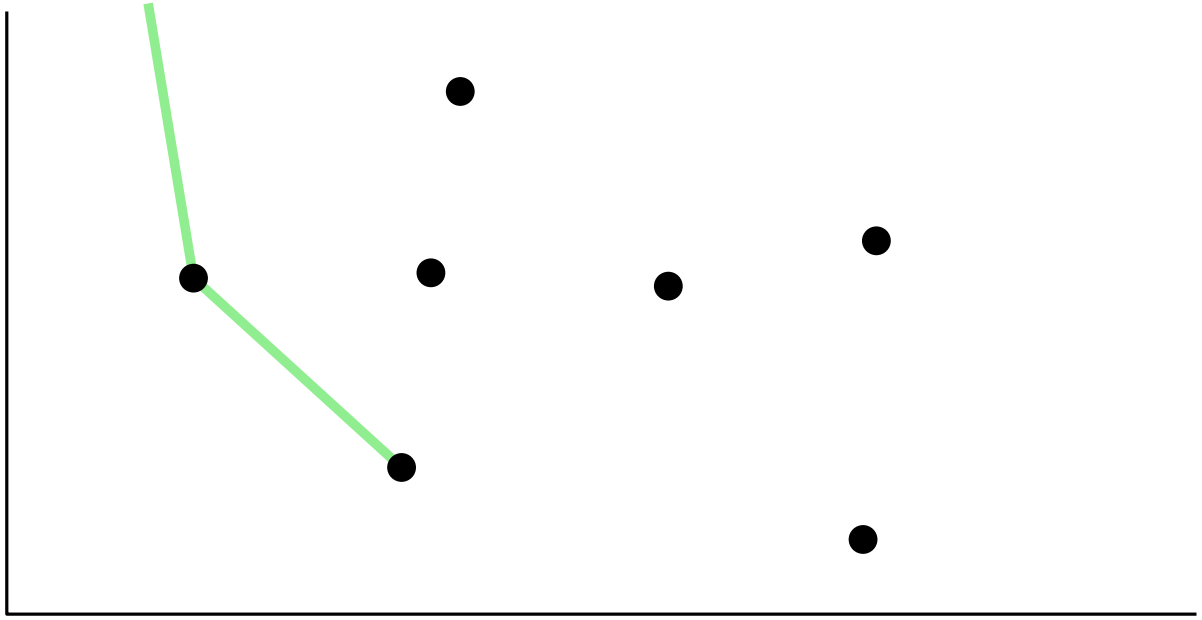
while true **do** $\mathcal{O}(h)$ $\mathcal{O}(n)$

Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

break

return $(p_1, \ldots, p_h)$



Kontextsensitiver Algorithmus

GiftWrapping(P)

$p_0 = (0, \infty)$, $p_1 =$ linkester Knoten in P

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

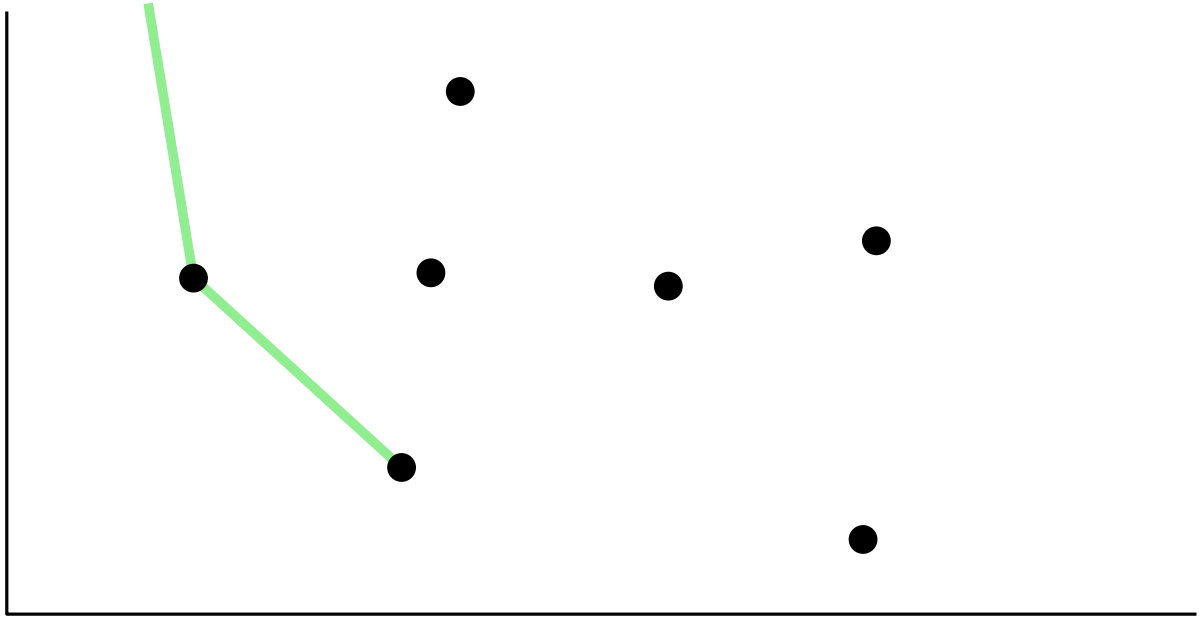
  Berechne $p_{k+1} \in P$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

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

    break

return $(p_1, \ldots, p_h)$

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



Optimalität?!

Wann ist welcher Algorithmus besser?

Optimalität?!

Wann ist welcher Algorithmus besser?

Viele Punkte auf $CH(P)$: $\mathcal{O}(n \log n)$ Graham Scan

Wenige Punkte auf $CH(P)$: $\mathcal{O}(nh)$ Gift Wrapping

Optimalität?!

Wann ist welcher Algorithmus besser?

Viele Punkte auf $CH(P)$: $\mathcal{O}(n \log n)$ Graham Scan

Wenige Punkte auf $CH(P)$: $\mathcal{O}(nh)$ Gift Wrapping

Geht es vielleicht noch besser?

Optimalität?!

Wann ist welcher Algorithmus besser?

Viele Punkte auf $CH(P)$: $\mathcal{O}(n \log n)$ Graham Scan

Wenige Punkte auf $CH(P)$: $\mathcal{O}(nh)$ Gift Wrapping

Geht es vielleicht noch besser?

ja

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

- Unterteile P in Mengen P_i mit je h Knoten
- **for** i von 1 bis n/h **do**
  - Berechne per GrahamScan $CH(P_i)$
- $p_0 = (0, \infty)$
- $p_1 =$ linkester Knoten in P
- **for** j von 1 bis h **do**
  - **for** i von 1 bis n/h **do**
    - Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert.
  - $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$
- **return** $(p_1, \ldots, p_h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

  Berechne per GrahamScan $CH(P_i)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**
  Berechne per GrahamScan $CH(P_i)$

GrahamScan

$p_0 = (0, \infty)$
$p_1 =$ linkester Knoten in P
for j von 1 bis h **do**
  for i von 1 bis n/h **do**
    Berechne Punkt $q_i \in P_i$ der den Winkel
    $p_{j-1}p_jq_i$ maximiert.
  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

Gift Wrapping

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

  Berechne per GrahamScan $CH(P_i)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do**

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert.

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do** $\mathcal{O}(n/h)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel

    $p_{j-1}p_jq_i$ maximiert.

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do** $\mathcal{O}(n/h)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel

    $p_{j-1}p_jq_i$ maximiert.

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

return $(p_1, \ldots, p_h)$

$\mathcal{O}(n \log h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do** $\mathcal{O}(n/h)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log m)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

return $(p_1, \ldots, p_h)$

$\mathcal{O}(n \log h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do** $\mathcal{O}(n/h)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$\mathcal{O}(n \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log m)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Karlsruhe Institute of Technology

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do** $\mathcal{O}(n/h)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$\mathcal{O}(n \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log m)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

$\mathcal{O}(n \log h)$

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Angenommen, wir kennen h:

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do** $\mathcal{O}(n/h)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$\mathcal{O}(n \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log m)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

$\mathcal{O}(n \log h)$

return $(p_1, \ldots, p_h)$

$$\mathcal{O}(n \log h)$$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P,h)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do** $\mathcal{O}(n/h)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$\mathcal{O}(n \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log m)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

$\mathcal{O}(n \log h)$

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P, ~~h~~ m)

Unterteile P in Mengen P_i mit je h Knoten

for i von 1 bis n/h **do** $\mathcal{O}(n/h)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(h \log h)$

$\mathcal{O}(n \log h)$

$p_0 = (0, \infty)$

$p_1 =$ linkester Knoten in P $\mathcal{O}(h) \cdot \mathcal{O}(n/h) = \mathcal{O}(n)$

for j von 1 bis h **do**

  for i von 1 bis n/h **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log m)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/h}\}$

$\mathcal{O}(n \log h)$

return $(p_1, \ldots, p_h)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P,m)

Unterteile P in Mengen P_i mit je m Knoten

for i von 1 bis n/m **do** $\mathcal{O}(n/m)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(m \log m)$

$\mathcal{O}(n \log m)$

$p_0 = (0, -\infty)$

$p_1 =$ linkester Knoten au P $\mathcal{O}(m) \cdot \mathcal{O}(n/m) = \mathcal{O}(n)$

for j von 1 bis m **do**

  for i von 1 bis n/m **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log m)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/m}\}$

$\mathcal{O}(n \log m)$

return $(p_1, \ldots, p_m)$

$\mathcal{O}(n \log m)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P,m)

Unterteile P in Mengen P_i mit je m Knoten

for i von 1 bis n/m **do** $\mathcal{O}(n/m)$

  Berechne per GrahamScan $CH(P_i)$ $\mathcal{O}(m \log m)$

$\mathcal{O}(n \log m)$

$p_0 = (0, -\infty)$

$p_1 =$ linkester Knoten au P $\mathcal{O}(m) \cdot \mathcal{O}(n/m) = \mathcal{O}(n)$

for j von 1 bis m **do**

  for i von 1 bis n/m **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $\mathcal{O}(\log m)$
    $p_{j-1}p_jq_i$ maximiert.

  $p_{j+1} = \max\{q_1, \ldots, q_{n/m}\}$

$\mathcal{O}(n \log m)$

return $(p_1, \ldots, p_m)$

$\mathcal{O}(n \log m)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P,m)

Unterteile P in Mengen P_i mit je m Knoten

...

$p_0 = (0, -\infty)$

$p_1 =$ linkester Knoten au P

for j von 1 bis m **do**

  for i von 1 bis n/m **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log h)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/m}\}$

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

    return $(p_1, \ldots, p_j)$

return failure

$\mathcal{O}(m) \cdot \mathcal{O}(n/m) = \mathcal{O}(n)$

$\mathcal{O}(n \log m)$

$\mathcal{O}(n \log m)$

Chans Algorithmus

Im Allgemeinen ist h aber unbekannt.

ChanHull(P,m)

Unterteile P in Mengen P_i mit je m Knoten

...

$p_0 = (0, -\infty)$

$p_1 =$ linkester Knoten au P

for j von 1 bis m **do**

  for i von 1 bis n/m **do**

    Berechne Punkt $q_i \in P_i$ der den Winkel $p_{j-1}p_jq_i$ maximiert. $\mathcal{O}(\log h)$

  $p_{j+1} = \max\{q_1, \ldots, q_{n/m}\}$

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

    return $(p_1, \ldots, p_j)$

return failure

$\mathcal{O}(n \log m)$

$\mathcal{O}(m) \cdot \mathcal{O}(n/m) = \mathcal{O}(n)$

$\mathcal{O}(n \log m)$

$\mathcal{O}(n \log m)$

Was ist mit m?

Vorschläge?

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**

  $m = \min\{n, 2^{2^t}\}$

  result = ChanHull(P, m)

  if result != failure **then**

    break

return result

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  result = ChanHull(P, m)
  if result != failure **then**
    break
return result

$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$

Laufzeit:

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  result = ChanHull(P, m)
  if result != failure **then**
    break
return result

$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$

Laufzeit:

$$\sum_{t=0}^{?} \mathcal{O}(n \log 2^{2^t})$$

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  result = ChanHull(P, m)
  if result != failure **then**
    break
return result

$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log \log h)} \mathcal{O}(n \log 2^{2^t})$$

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  result = ChanHull(P, m)
  if result != failure **then**
    break
return result

$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(2^t)$$

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  result = ChanHull(P, m)
  if result != failure **then**
    break
return result

$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(2^t)$$

$$= \mathcal{O}(n) \cdot \mathcal{O}(2^{\log\log h + 1})$$

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  result = ChanHull(P, m)
  if result != failure **then**
    break
return result

$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(2^t)$$

$$= \mathcal{O}(n) \cdot \mathcal{O}(2^{\log\log h + 1}) = \mathcal{O}(n) \cdot \mathcal{O}(2^{\log\log h})$$

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  result = ChanHull(P, m)
  if result != failure **then**
    break
return result

$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(2^t)$$

$$= \mathcal{O}(n) \cdot \mathcal{O}(2^{\log\log h + 1}) = \mathcal{O}(n) \cdot \mathcal{O}(2^{\log\log h})$$

$$= \mathcal{O}(n) \cdot \mathcal{O}(\log h)$$

Was ist mit m?

FullChanHull(P)

for $t = 0, 1, 2, \ldots$ **do**
  $m = \min\{n, 2^{2^t}\}$
  result = ChanHull(P, m)
  if result != failure **then**
    break
return result

$\mathcal{O}(n \log m) = \mathcal{O}(n \log 2^{2^t})$

Laufzeit:

$$\sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(n \log 2^{2^t}) = \mathcal{O}(n) \sum_{t=0}^{\mathcal{O}(\log\log h)} \mathcal{O}(2^t)$$

$$= \mathcal{O}(n) \cdot \mathcal{O}(2^{\log\log h + 1}) = \mathcal{O}(n) \cdot \mathcal{O}(2^{\log\log h})$$

$$= \mathcal{O}(n) \cdot \mathcal{O}(\log h) = \mathcal{O}(n \log h)$$

Das war's!

Karlsruhe Institute of Technology

Fragen?

Nächster Termin:
Donnerstag, 28.04, 10:15 Uhr
Segmentschnitte
Raum 131, Gebäude 50.34