

Algorithmische Geometrie

Überblick und Konvexe Hülle



Was ist das überhaupt?

Wikipedia

- Als Algorithmische Geometrie (englisch Computational Geometry) bezeichnet man ein Teilgebiet der Informatik, das sich mit der algorithmischen Lösung geometrisch formulierter Probleme beschäftigt.

Was ist das überhaupt?

Wikipedia

- Als Algorithmische Geometrie (englisch Computational Geometry) bezeichnet man ein Teilgebiet der Informatik, das sich mit der algorithmischen Lösung geometrisch formulierter Probleme beschäftigt.
- Ein zentrales Problem ist dabei die Speicherung und Verarbeitung geometrischer Daten.

Was ist das überhaupt?

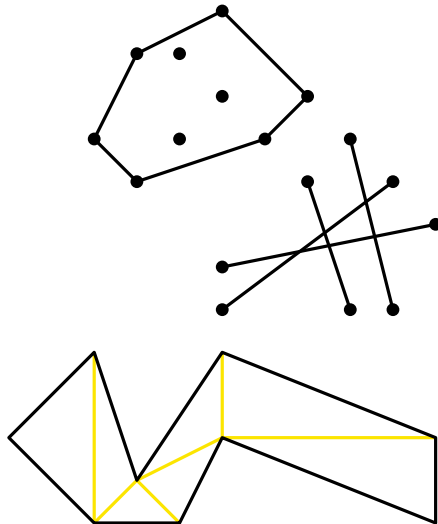
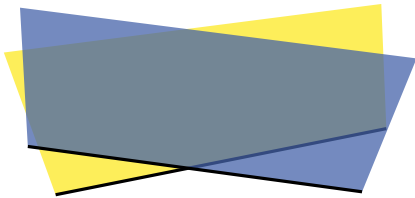
Wikipedia

- Als Algorithmische Geometrie (englisch Computational Geometry) bezeichnet man ein Teilgebiet der Informatik, das sich mit der algorithmischen Lösung geometrisch formulierter Probleme beschäftigt.
- Ein zentrales Problem ist dabei die Speicherung und Verarbeitung geometrischer Daten.
- Im Gegensatz zur Bildbearbeitung, deren Grundelemente Bildpunkte (Pixel) sind, arbeitet die algorithmische Geometrie mit geometrischen Strukturelementen wie Punkten, Linien, Kreisen, Polygonen und Körpern.

Was heißt das jetzt konkret?

Basic Toolbox

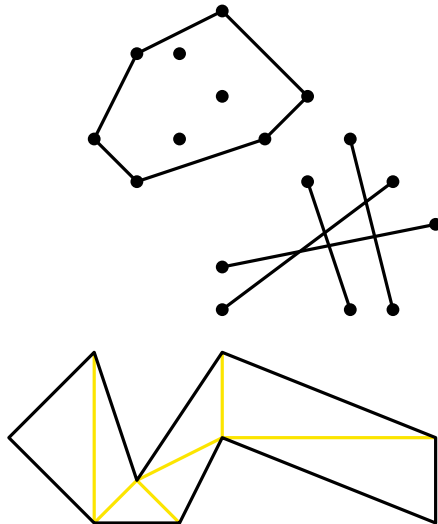
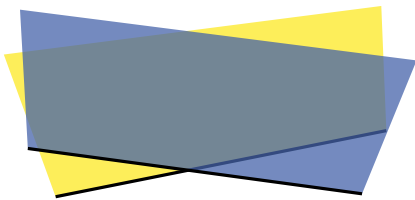
- konvexe Hülle
- Linienschnitt
- Triangulierung
- Ebenenschnitt



Was heißt das jetzt konkret?

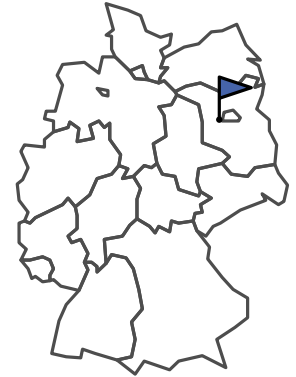
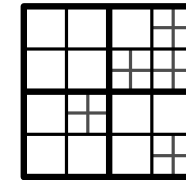
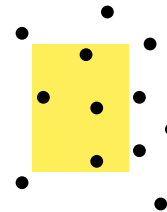
Basic Toolbox

- konvexe Hülle
- Linienschnitt
- Triangulierung
- Ebenenschnitt



Geometrische Datenstrukturen

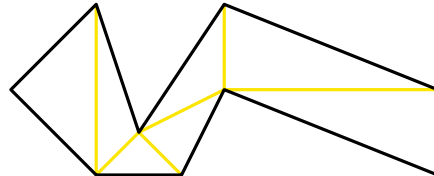
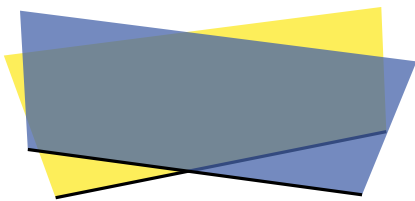
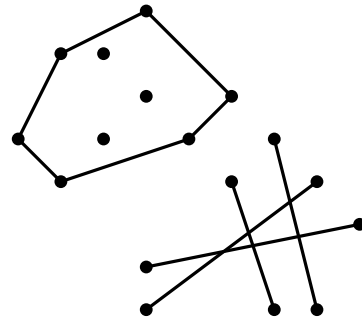
- orthogonal range searching
- Raum-Partitionierung
- Punkt-Lokalisierung



Was heißt das jetzt konkret?

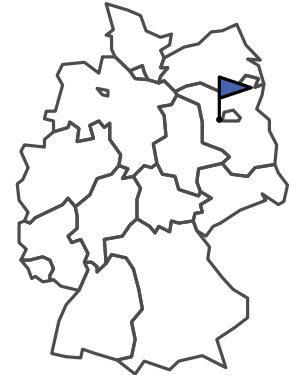
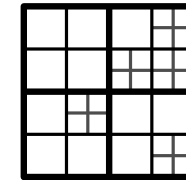
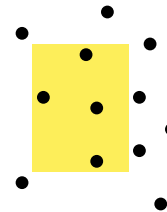
Basic Toolbox

- konvexe Hülle
- Linienschnitt
- Triangulierung
- Ebenenschnitt



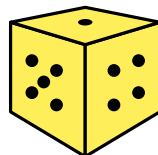
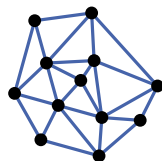
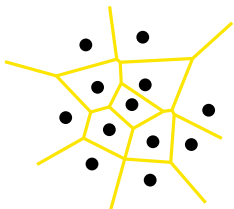
Geometrische Datenstrukturen

- orthogonal range searching
- Raum-Partitionierung
- Punkt-Lokalisierung



Erweiterte Toolbox

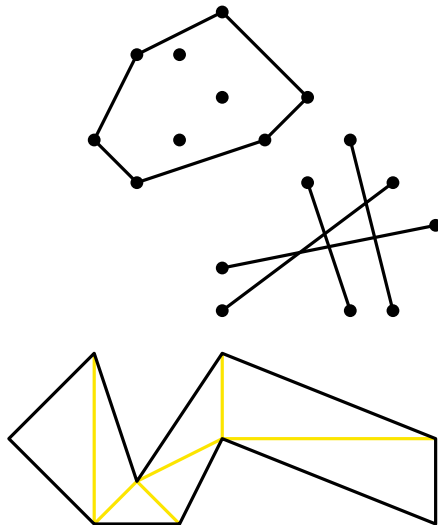
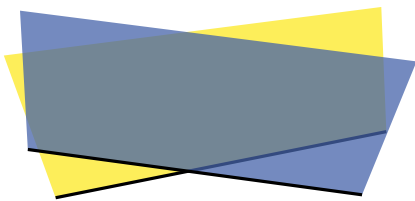
- Voronoi-Diagramme
- Delaunay-Triangulierung
- Randomisierte Algorithmen
- Komplexität



Was heißt das jetzt konkret?

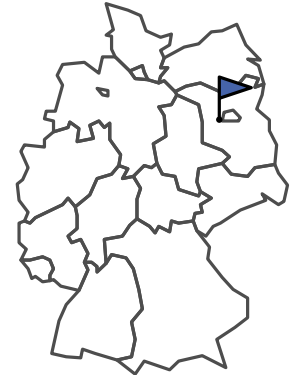
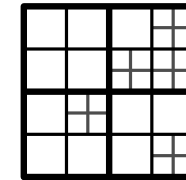
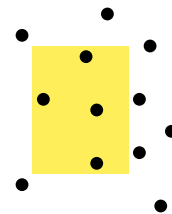
Basic Toolbox

- konvexe Hülle
- Linienschnitt
- Triangulierung
- Ebenenschnitt



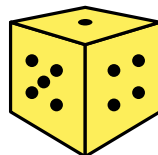
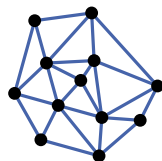
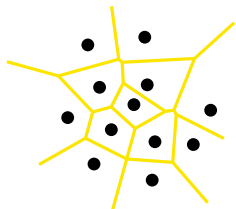
Geometrische Datenstrukturen

- orthogonal range searching
- Raum-Partitionierung
- Punkt-Lokalisierung



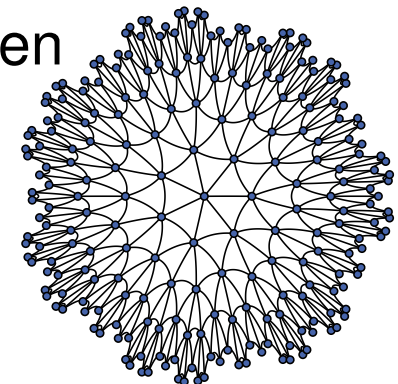
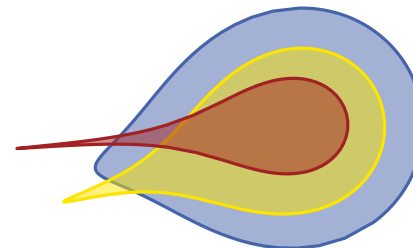
Erweiterte Toolbox

- Voronoi-Diagramme
- Delaunay-Triangulierung
- Randomisierte Algorithmen
- Komplexität



Verwandte Themen

- Was ist Geometrie überhaupt?
- hyperbolische Geometrie
- geometrische Graphen



Organisatorisches

Woche i							Woche $i + 1$							Woche $i + 2$							Woche $i + 3$							$(i \text{ gerade})$
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	
Übungsblatt $\frac{i}{2}$														Übungsblatt $\frac{i+2}{2}$														

Vorlesung

- Vorlesung mit Folien

Organisatorisches

Woche i							Woche $i + 1$							Woche $i + 2$							Woche $i + 3$							$(i \text{ gerade})$
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	
Übungsblatt $\frac{i}{2}$														Übungsblatt $\frac{i+2}{2}$														

Vorlesung

- Vorlesung mit Folien

Übung (Woche $i + 1$)

- gehalten von Marcus
- Besprechung Blatt $\frac{i}{2} - 1$
- Diskussion zu Blatt $\frac{i}{2}$

Organisatorisches

Woche i							Woche $i + 1$							Woche $i + 2$							Woche $i + 3$							$(i \text{ gerade})$
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	
Übungsblatt $\frac{i}{2}$														Übungsblatt $\frac{i+2}{2}$														

Vorlesung

- Vorlesung mit Folien

Übung (Woche $i + 1$)

- gehalten von Marcus
- Besprechung Blatt $\frac{i}{2} - 1$
- Diskussion zu Blatt $\frac{i}{2}$

Aktiv-Session

- Erarbeitung weiterführender Aspekte
- Kuriositäten
- gemeinsames Papier-Lesen

Organisatorisches

Woche i							Woche $i + 1$							Woche $i + 2$							Woche $i + 3$							$(i \text{ gerade})$
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	
Übungsblatt $\frac{i}{2}$														Übungsblatt $\frac{i+2}{2}$														

Vorlesung

- Vorlesung mit Folien

Übung (Woche $i + 1$)

- gehalten von Marcus
- Besprechung Blatt $\frac{i}{2} - 1$
- Diskussion zu Blatt $\frac{i}{2}$

Aktiv-Session

- Erarbeitung weiterführender Aspekte
- Kuriositäten
- gemeinsames Papier-Lesen

Materialien

- Vorlesungsfolien

Organisatorisches

Woche i							Woche $i + 1$							Woche $i + 2$							Woche $i + 3$							$(i \text{ gerade})$
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	
Übungsblatt $\frac{i}{2}$														Übungsblatt $\frac{i+2}{2}$														

Vorlesung

- Vorlesung mit Folien

Übung (Woche $i + 1$)

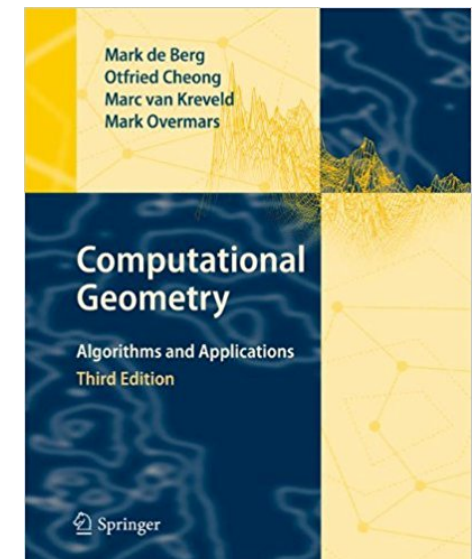
- gehalten von Marcus
- Besprechung Blatt $\frac{i}{2} - 1$
- Diskussion zu Blatt $\frac{i}{2}$

Aktiv-Session

- Erarbeitung weiterführender Aspekte
- Kuriositäten
- gemeinsames Papier-Lesen

Materialien

- Vorlesungsfolien
- Buch *Computational Geometry*



Organisatorisches

Woche i							Woche $i + 1$							Woche $i + 2$							Woche $i + 3$							$(i \text{ gerade})$
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	
Übungsblatt $\frac{i}{2}$														Übungsblatt $\frac{i+2}{2}$														

Vorlesung

- Vorlesung mit Folien

Übung (Woche $i + 1$)

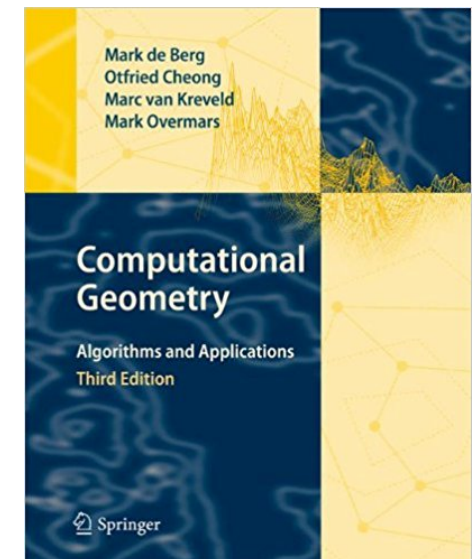
- gehalten von Marcus
- Besprechung Blatt $\frac{i}{2} - 1$
- Diskussion zu Blatt $\frac{i}{2}$

Aktiv-Session

- Erarbeitung weiterführender Aspekte
- Kuriositäten
- gemeinsames Papier-Lesen

Materialien

- Vorlesungsfolien
- Buch *Computational Geometry*
- Aufzeichnung der Vorlesung von letztem Jahr



Organisatorisches

Woche i							Woche $i + 1$							Woche $i + 2$							Woche $i + 3$							$(i \text{ gerade})$
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	
Übungsblatt $\frac{i}{2}$														Übungsblatt $\frac{i+2}{2}$														

Vorlesung

- Vorlesung mit Folien

Übung (Woche $i + 1$)

- gehalten von Marcus
- Besprechung Blatt $\frac{i}{2} - 1$
- Diskussion zu Blatt $\frac{i}{2}$

Aktiv-Session

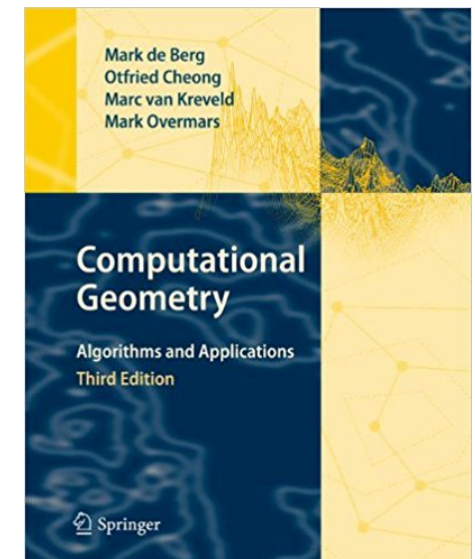
- Erarbeitung weiterführender Aspekte
- Kuriositäten
- gemeinsames Papier-Lesen

Materialien

- Vorlesungsfolien
- Buch *Computational Geometry*
- Aufzeichnung der Vorlesung von letztem Jahr

Prüfung & Übungsblätter

- mündlich
- Zulassungsvoraussetzung: 50% der Punkte bei den Übungsaufgaben (korrigiert von Carina)
- Abgabe gerne in 2er-Gruppen



Kurzvorstellung

Uni

- Was studierst du? Welches Semester?
- Was sind deine Hauptinteressen?
- Erfahrungen mit Geometrie/Algorithmen?

Und sonst so?

- Spielst du ein Instrument? Welches?
- Lieblingsspiel? Lieblingsbuch? Lieblingsfilm?
- Machst du Sport? Welche Sportart?
- Kochst du gerne oder isst du lieber nur? Lieblingsessen?
- ...

Kurzvorstellung

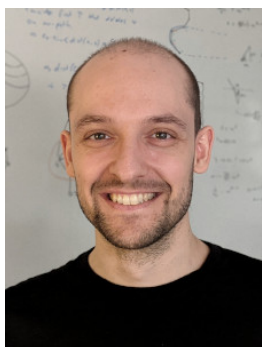
Uni

- Was studierst du? Welches Semester?
- Was sind deine Hauptinteressen?
- Erfahrungen mit Geometrie/Algorithmen?

Und sonst so?

- Spielst du ein Instrument? Welches?
- Lieblingsspiel? Lieblingsbuch? Lieblingsfilm?
- Machst du Sport? Welche Sportart?
- Kochst du gerne oder isst du lieber nur? Lieblingsessen?
- ...

Wir fangen an



Thomas
Vorlesung



Marcus
Übung



Carina
Korrektur

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B
- zwei Quellen:

	A	B
Quelle 1	35%	10%
Quelle 2	20%	16%

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B
 - zwei Quellen:
- | | A | B |
|----------|-----|-----|
| Quelle 1 | 35% | 10% |
| Quelle 2 | 20% | 16% |

Können wir 30% von A und 12% von B erreichen?

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B
- zwei Quellen:

	A	B
Quelle 1	35%	10%
Quelle 2	20%	16%

Können wir 30% von A und 12% von B erreichen?

2 : 1

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B
- zwei Quellen:

	A	B
Quelle 1	35%	10%
Quelle 2	20%	16%

Können wir 30% von A und 12% von B erreichen?

2 : 1

22% von A und 13% von B ?

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B
- zwei Quellen:

	A	B
Quelle 1	35%	10%
Quelle 2	20%	16%
- dritte Quelle:

Quelle 3	15%	7%
----------	-----	----

Können wir 30% von A und 12% von B erreichen?

2 : 1

22% von A und 13% von B ?

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B
- zwei Quellen:

	A	B
Quelle 1	35%	10%
Quelle 2	20%	16%
- dritte Quelle:

Quelle 3	15%	7%
----------	-----	----

Können wir 30% von A und 12% von B erreichen?

2 : 1

22% von A und 13% von B ?

1 : 3 : 1

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B
- zwei Quellen:

	A	B
Quelle 1	35%	10%
Quelle 2	20%	16%
- dritte Quelle:

Quelle 3	15%	7%
----------	-----	----

Können wir 30% von A und 12% von B erreichen?

2 : 1

22% von A und 13% von B ?

1 : 3 : 1

Was hat das mit Geometrie zu tun?

Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B

- zwei Quellen:

	A	B
Quelle 1	35%	10%
Quelle 2	20%	16%

- dritte Quelle:

Quelle 3	15%	7%
----------	-----	----

Können wir 30% von A und 12% von B erreichen?

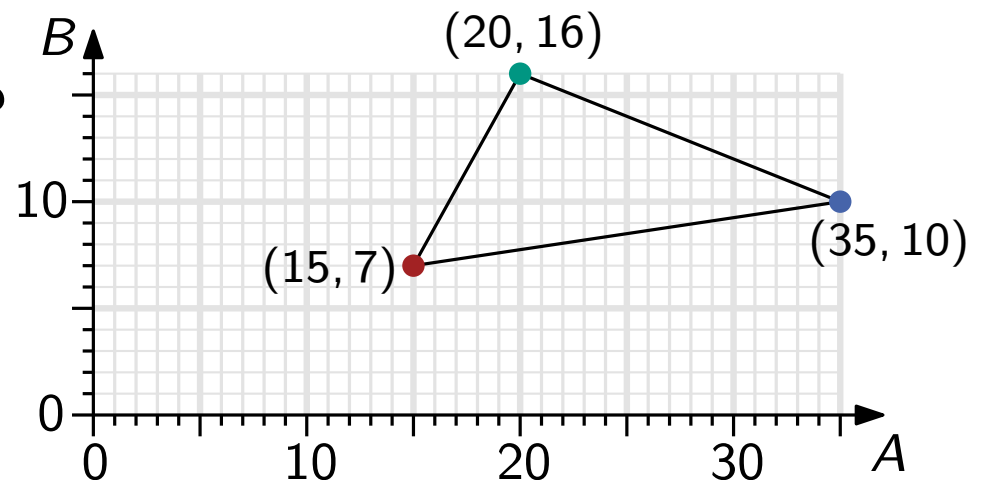
2 : 1

22% von A und 13% von B ?

1 : 3 : 1

Was hat das mit Geometrie zu tun?

- Mischverhältnis kann als Punkt interpretiert werden



Motivation

Öl in unterschiedlichen Zusammensetzungen

- die Zusammensetzung von Öl hängt von der Quelle ab, aus der es stammt
- Ziel: mische Öl aus verschiedenen Quellen, sodass die resultierende Zusammensetzung besonders gut zu verarbeiten ist

Beispiel

- Öl enthält die Komponenten A und B

- zwei Quellen:

	A	B
Quelle 1	35%	10%
Quelle 2	20%	16%

- dritte Quelle:

Quelle 3	15%	7%
----------	-----	----

Können wir 30% von A und 12% von B erreichen?

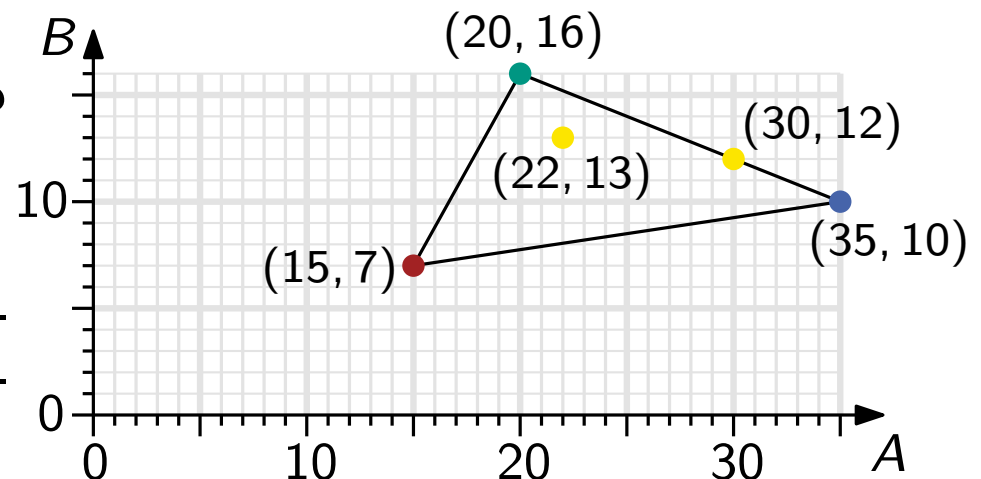
2 : 1

22% von A und 13% von B ?

1 : 3 : 1

Was hat das mit Geometrie zu tun?

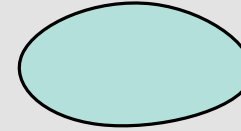
- Mischverhältnis kann als Punkt interpretiert werden
- Mischung ist möglich \Leftrightarrow zugehöriger Punkt liegt „zwischen“ den verfügbaren Punkten



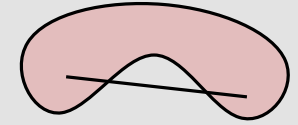
Konvexe Hülle

Definition

Eine Punktmenge $P \subseteq \mathbb{R}^d$ ist **konvex** wenn für je zwei Punkte $p, q \in P$ auch die Strecke pq in P liegt.



konvex

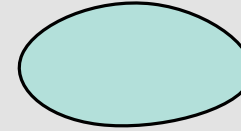


nicht-konvex

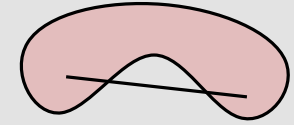
Konvexe Hülle

Definition

Eine Punktmenge $P \subseteq \mathbb{R}^d$ ist **konvex** wenn für je zwei Punkte $p, q \in P$ auch die Strecke pq in P liegt.



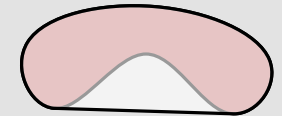
konvex



nicht-konvex

Definition

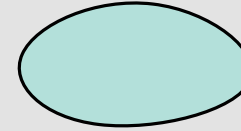
Für $P \subseteq \mathbb{R}^d$ ist die **Konvexe Hülle** $\mathcal{CH}(P)$ die inklusionsminimale Teilmenge von \mathbb{R}^d , sodass $\mathcal{CH}(P)$ konvex ist und $P \subseteq \mathcal{CH}(P)$.



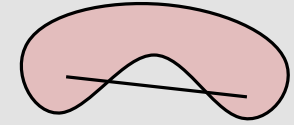
Konvexe Hülle

Definition

Eine Punktmenge $P \subseteq \mathbb{R}^d$ ist **konvex** wenn für je zwei Punkte $p, q \in P$ auch die Strecke pq in P liegt.



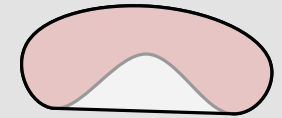
konvex



nicht-konvex

Definition

Für $P \subseteq \mathbb{R}^d$ ist die **Konvexe Hülle** $\mathcal{CH}(P)$ die inklusionsminimale Teilmenge von \mathbb{R}^d , sodass $\mathcal{CH}(P)$ konvex ist und $P \subseteq \mathcal{CH}(P)$.



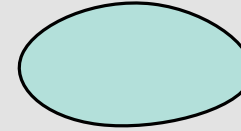
Äquivalente Definitionen

- der Schnitt aller konvexer Mengen aus \mathbb{R}^d , die P enthalten

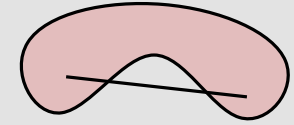
Konvexe Hülle

Definition

Eine Punktmenge $P \subseteq \mathbb{R}^d$ ist **konvex** wenn für je zwei Punkte $p, q \in P$ auch die Strecke pq in P liegt.



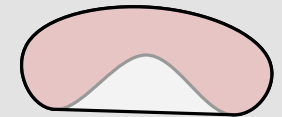
konvex



nicht-konvex

Definition

Für $P \subseteq \mathbb{R}^d$ ist die **Konvexe Hülle** $\mathcal{CH}(P)$ die inklusionsminimale Teilmenge von \mathbb{R}^d , sodass $\mathcal{CH}(P)$ konvex ist und $P \subseteq \mathcal{CH}(P)$.



Äquivalente Definitionen

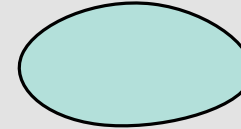
- der Schnitt aller konvexer Mengen aus \mathbb{R}^d , die P enthalten
- die Vereinigung aller Simplexe mit Eckpunkten in P



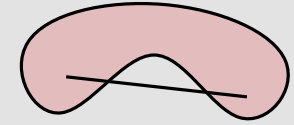
Konvexe Hülle

Definition

Eine Punktmenge $P \subseteq \mathbb{R}^d$ ist **konvex** wenn für je zwei Punkte $p, q \in P$ auch die Strecke pq in P liegt.



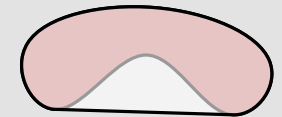
konvex



nicht-konvex

Definition

Für $P \subseteq \mathbb{R}^d$ ist die **Konvexe Hülle** $\mathcal{CH}(P)$ die inklusionsminimale Teilmenge von \mathbb{R}^d , sodass $\mathcal{CH}(P)$ konvex ist und $P \subseteq \mathcal{CH}(P)$.



Äquivalente Definitionen

- der Schnitt aller konvexer Mengen aus \mathbb{R}^d , die P enthalten
- die Vereinigung aller Simplexe mit Eckpunkten in P

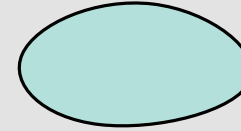


- Menge aller Punkte, die Konvexkombinationen von Punkten aus P sind

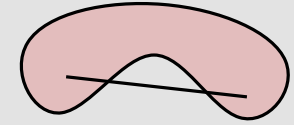
Konvexe Hülle

Definition

Eine Punktmenge $P \subseteq \mathbb{R}^d$ ist **konvex** wenn für je zwei Punkte $p, q \in P$ auch die Strecke pq in P liegt.



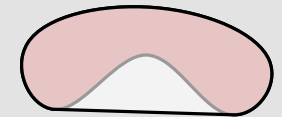
konvex



nicht-konvex

Definition

Für $P \subseteq \mathbb{R}^d$ ist die **Konvexe Hülle** $\mathcal{CH}(P)$ die inklusionsminimale Teilmenge von \mathbb{R}^d , sodass $\mathcal{CH}(P)$ konvex ist und $P \subseteq \mathcal{CH}(P)$.



Äquivalente Definitionen

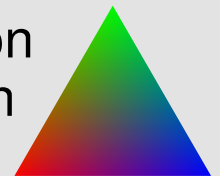
- der Schnitt aller konvexer Mengen aus \mathbb{R}^d , die P enthalten
- die Vereinigung aller Simplexe mit Eckpunkten in P



- Menge aller Punkte, die **Konvexkombinationen** von Punkten aus P sind

$$\sum_{i=1}^n a_i \cdot p_i \text{ mit } p_i \in P, 0 \leq a_i \in \mathbb{R} \text{ und } \sum_{i=1}^n a_i = 1$$

kennt ihr ggf. von
baryzentrischen
Koordinaten



Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

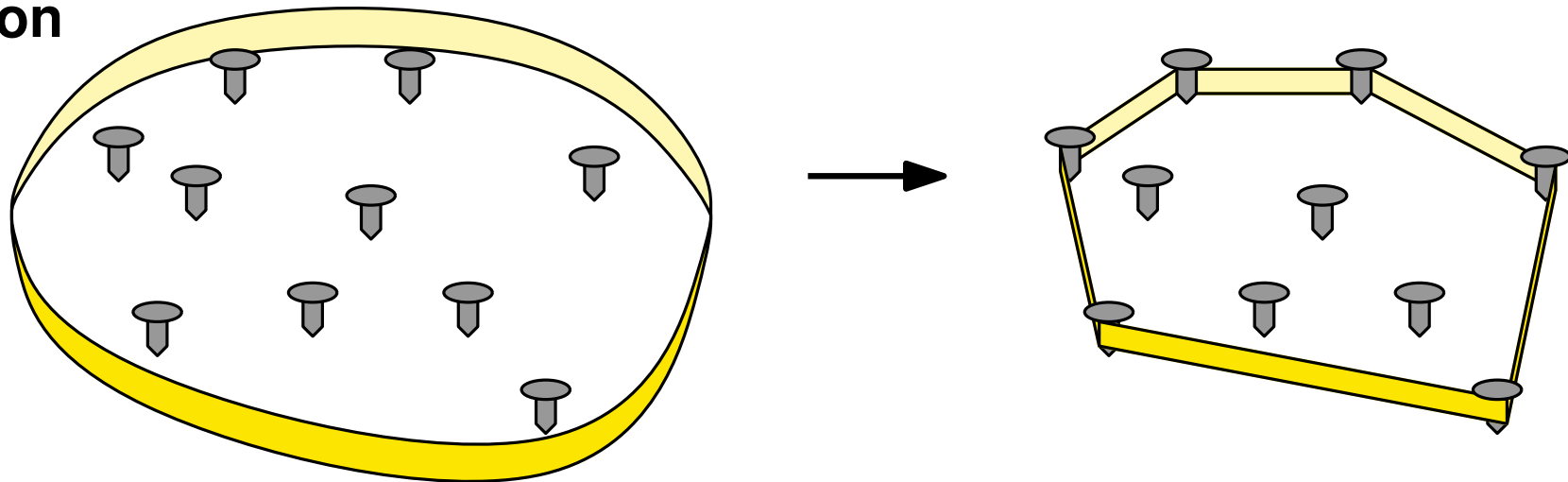
Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Intuition



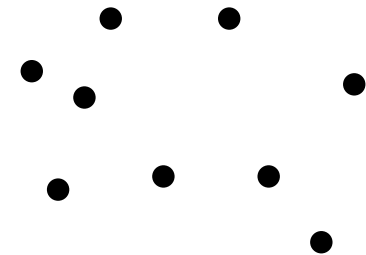
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage



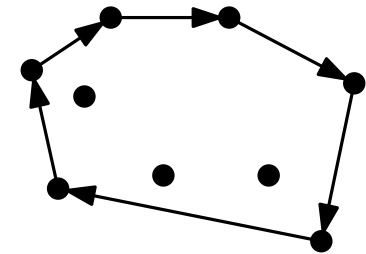
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
→ gewünschte Ausgabe ist eine Folge von Punkten



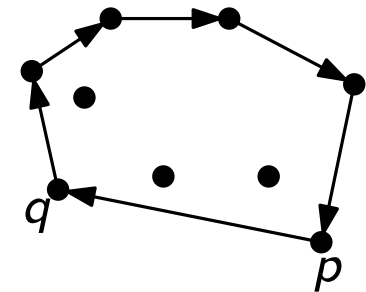
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
 → gewünschte Ausgabe ist eine Folge von Punkten
- wann ist pq eine Kante von $\mathcal{CH}(P)$?



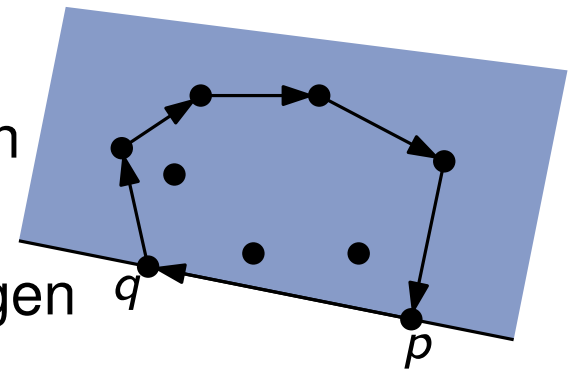
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
→ gewünschte Ausgabe ist eine Folge von Punkten
- wann ist pq eine Kante von $\mathcal{CH}(P)$?
 - wenn alle Punkte im Halbraum rechts von pq liegen



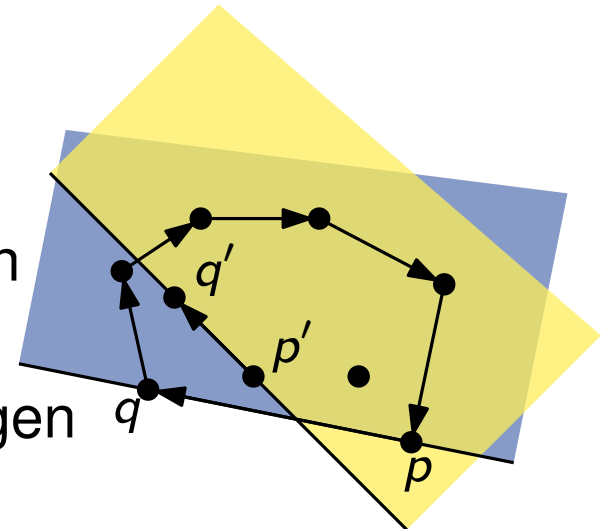
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
→ gewünschte Ausgabe ist eine Folge von Punkten
- wann ist pq eine Kante von $\mathcal{CH}(P)$?
 - wenn alle Punkte im Halbraum rechts von pq liegen



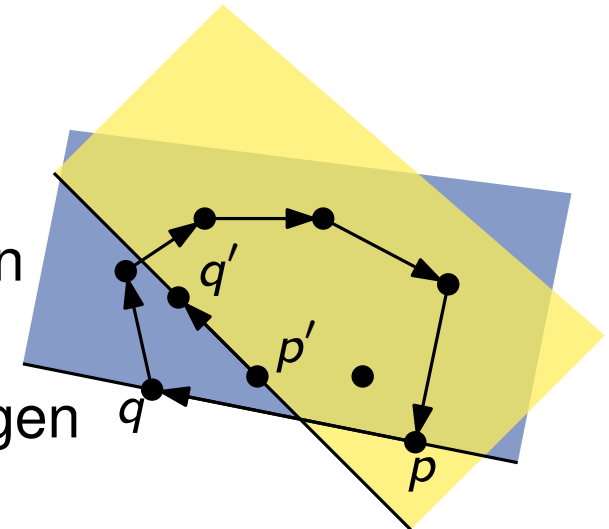
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
→ gewünschte Ausgabe ist eine Folge von Punkten
- wann ist pq eine Kante von $\mathcal{CH}(P)$?
 - wenn alle Punkte im Halbraum rechts von pq liegen



Trivialer Algorithmus

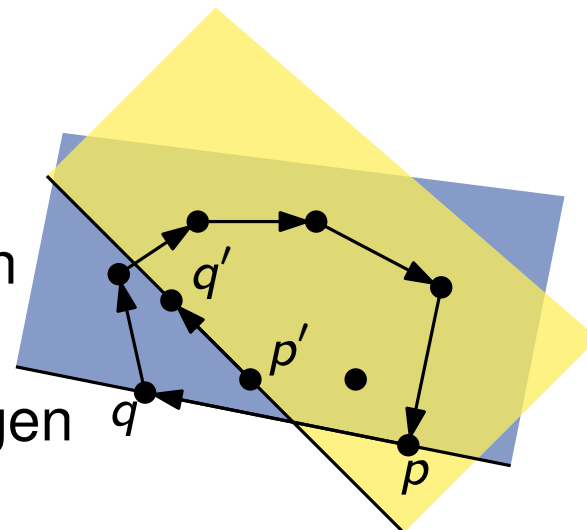
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
→ gewünschte Ausgabe ist eine Folge von Punkten
- wann ist pq eine Kante von $\mathcal{CH}(P)$?
 - wenn alle Punkte im Halbraum rechts von pq liegen



Trivialer Algorithmus

- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq

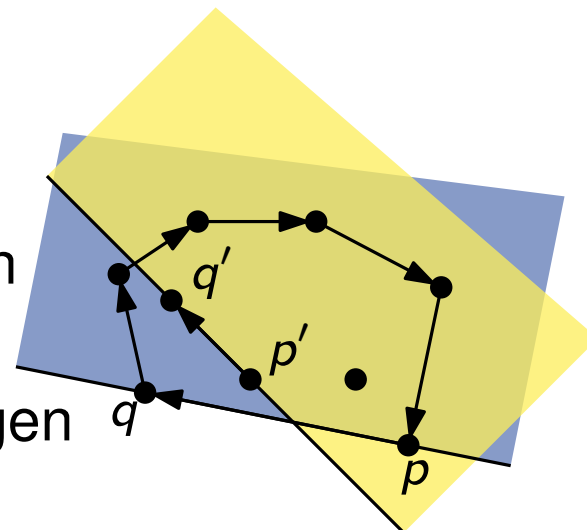
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
→ gewünschte Ausgabe ist eine Folge von Punkten
- wann ist pq eine Kante von $\mathcal{CH}(P)$?
 - wenn alle Punkte im Halbraum rechts von pq liegen



Trivialer Algorithmus

- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq
- konstruiere das Polygon (als Punktfolge) aus den gespeicherten Kanten

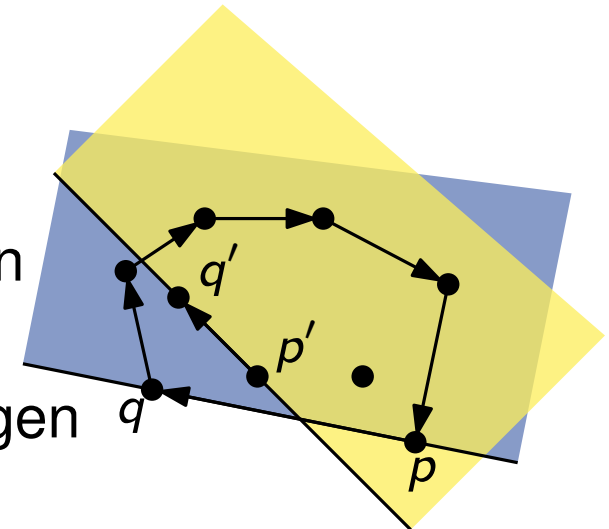
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
→ gewünschte Ausgabe ist eine Folge von Punkten
- wann ist pq eine Kante von $\mathcal{CH}(P)$?
 - wenn alle Punkte im Halbraum rechts von pq liegen



Trivialer Algorithmus

- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq
- konstruiere das Polygon (als Punktfolge) aus den gespeicherten Kanten

Laufzeit:

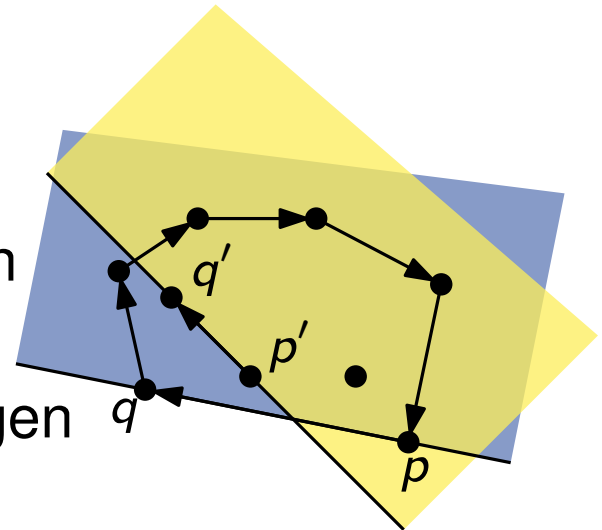
Konvexe Hülle – Trivialer Algorithmus

Problem: KONVEXE HÜLLE (2D)

Gegeben n Punkte $P \subseteq \mathbb{R}^2$, berechne die konvexe Hülle $\mathcal{CH}(P)$.

Anmerkung & Beobachtung

- Annahme: die Punkte sind in allgemeiner Lage
- $\mathcal{CH}(P)$ ist durch ein Polygon begrenzt
→ gewünschte Ausgabe ist eine Folge von Punkten
- wann ist pq eine Kante von $\mathcal{CH}(P)$?
 - wenn alle Punkte im Halbraum rechts von pq liegen



Trivialer Algorithmus

- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq
- konstruiere das Polygon (als Punktfolge) aus den gespeicherten Kanten

Laufzeit: $O(n^3)$

Konvexe Hülle – Trivialer Algorithmus

Trivialer Algorithmus

- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq
- konstruiere das Polygon (als Punktfolge) aus den gespeicherten Kanten

Probleme

- der Algorithmus ist langsam

Konvexe Hülle – Trivialer Algorithmus

Trivialer Algorithmus

- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq
- konstruiere das Polygon (als Punktfolge) aus den gespeicherten Kanten

Probleme

- der Algorithmus ist langsam
- der Algorithmus ist nicht robust

Konvexe Hülle – Trivialer Algorithmus

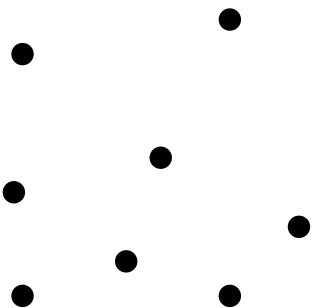
Trivialer Algorithmus

- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq
- konstruiere das Polygon (als Punktfolge) aus den gespeicherten Kanten

Probleme

- der Algorithmus ist langsam
- der Algorithmus ist nicht robust

Beispiel (Robustheit)



Konvexe Hülle – Trivialer Algorithmus

Trivialer Algorithmus

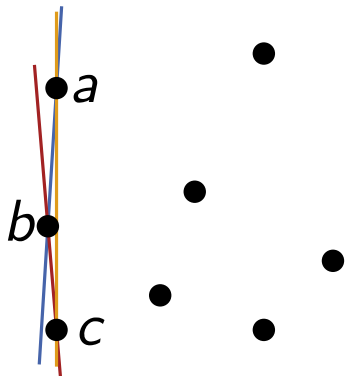
- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq
- konstruiere das Polygon (als Punktfolge) aus den gespeicherten Kanten

Probleme

- der Algorithmus ist langsam
- der Algorithmus ist nicht robust

Beispiel (Robustheit)

- drei der Entscheidungen „liegt rechts von“ sind knapp



Konvexe Hülle – Trivialer Algorithmus

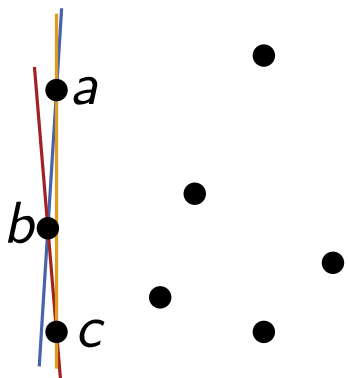
Trivialer Algorithmus

- iteriere über alle Knotenpaare $p, q \in P$ (gerichtet)
 - teste ob alle Punkte rechts von pq liegen
 - falls ja: speicher pq
- konstruiere das Polygon (als Punktfolge) aus den gespeicherten Kanten

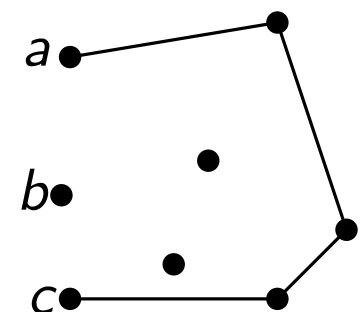
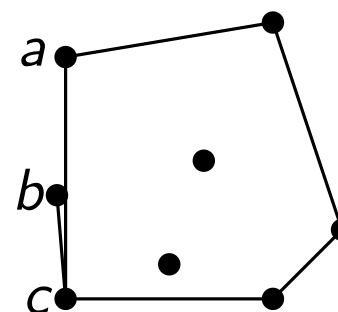
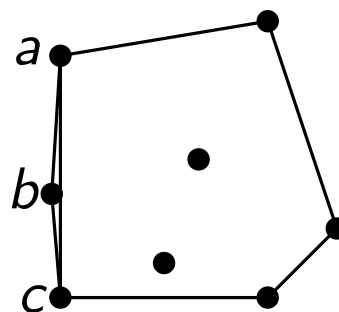
Probleme

- der Algorithmus ist langsam
- der Algorithmus ist nicht robust

Beispiel (Robustheit)



- drei der Entscheidungen „liegt rechts von“ sind knapp
- falsche Entscheidungen liefern ggf. kein Polygon

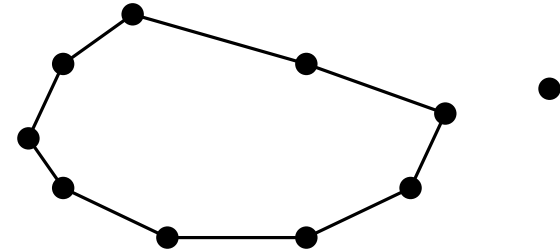


Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt

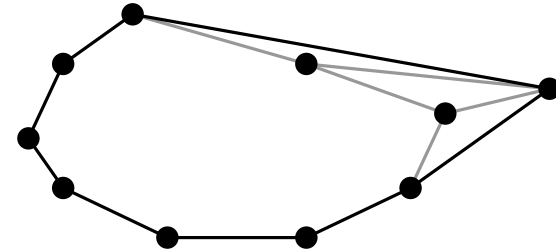


Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt

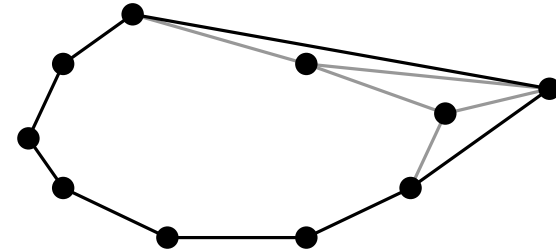


Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke

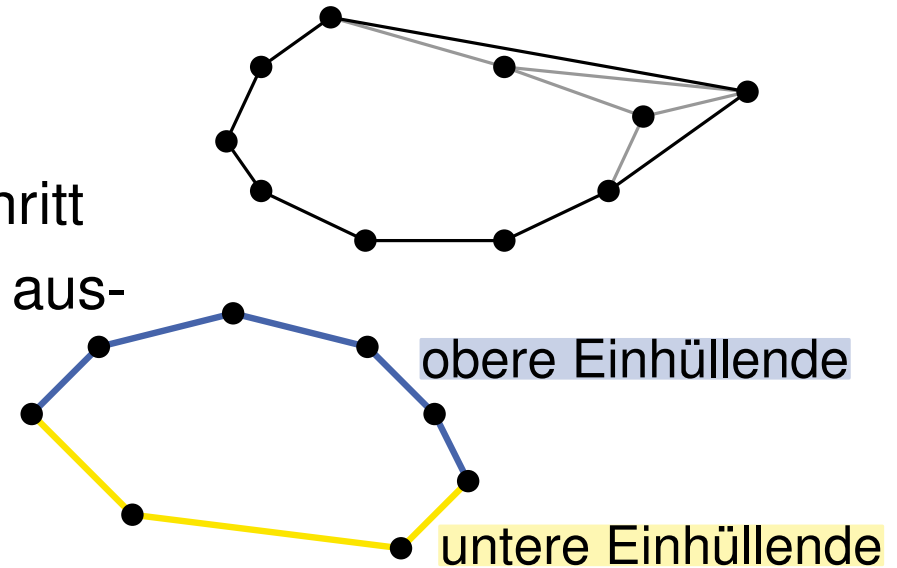


Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

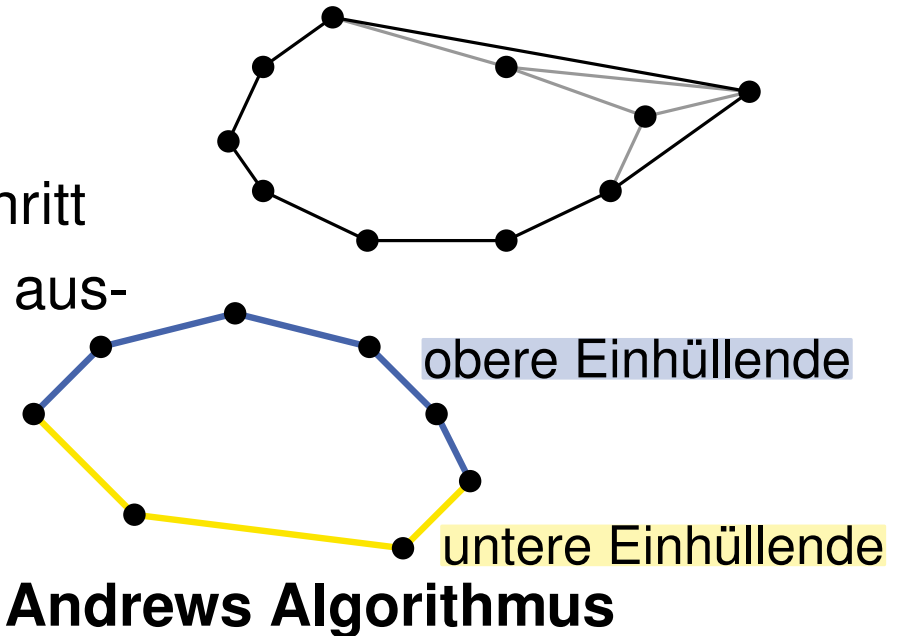


Andrews Monotone Chain Algorithmus

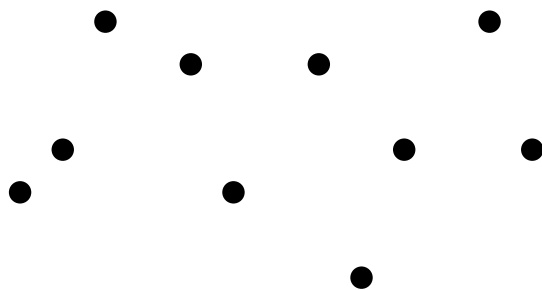
(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende



Beispiel



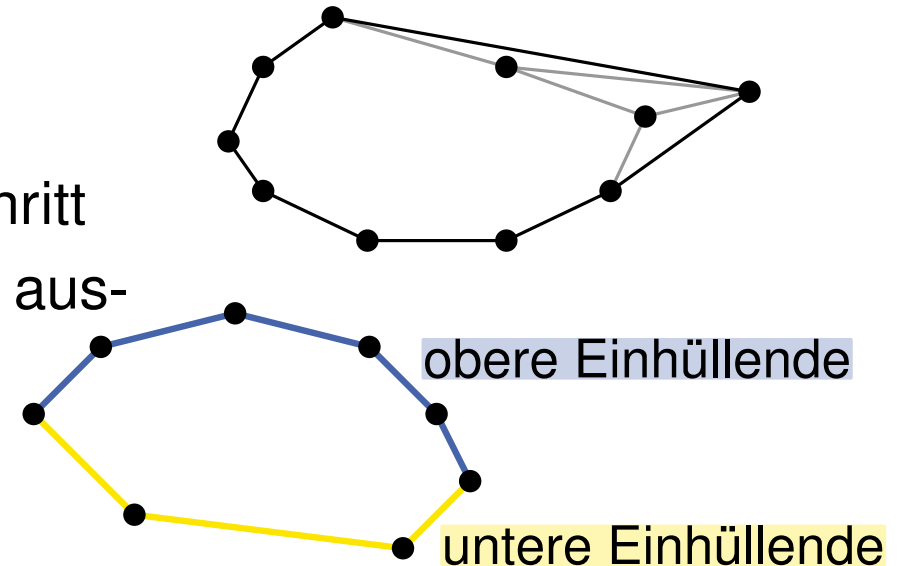
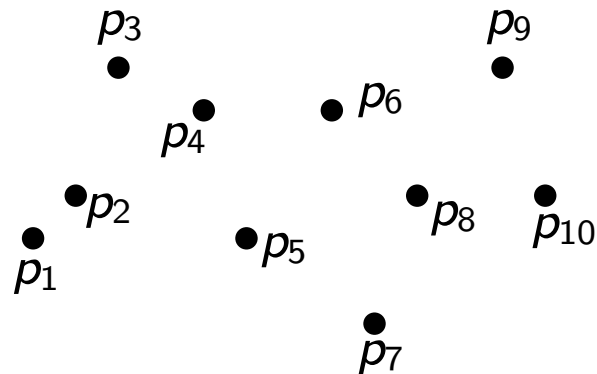
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

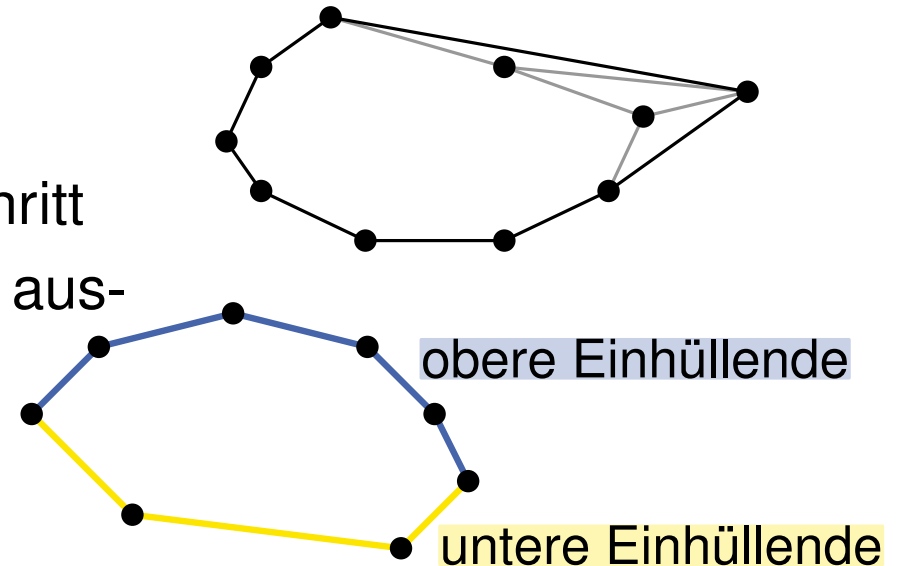
- sortiere P (bzgl. x): p_1, \dots, p_n

Andrews Monotone Chain Algorithmus

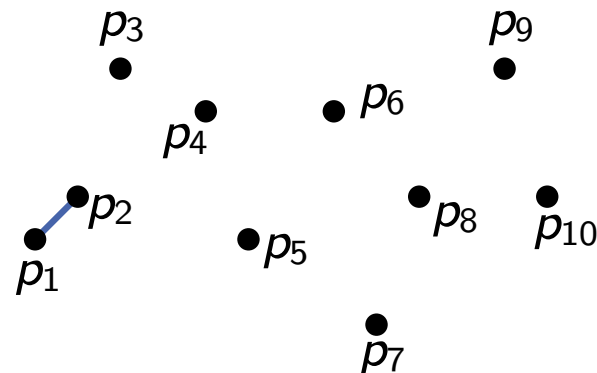
(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende



Beispiel



$L: p_1 p_2$

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein

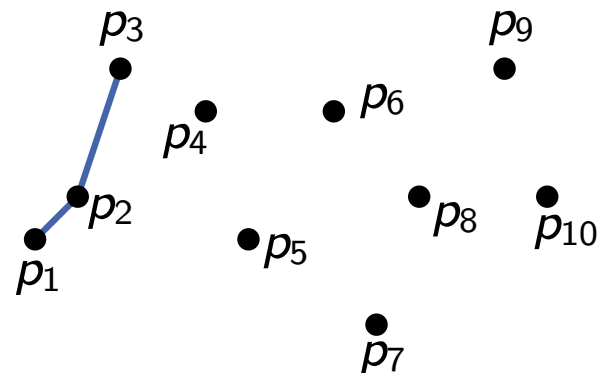
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

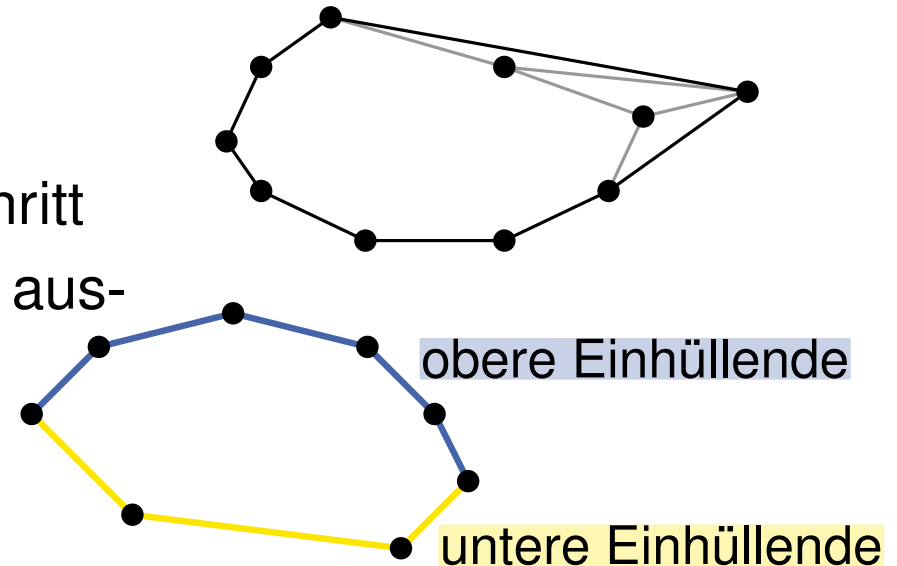
Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



$L: p_1 p_2 p_3$



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein

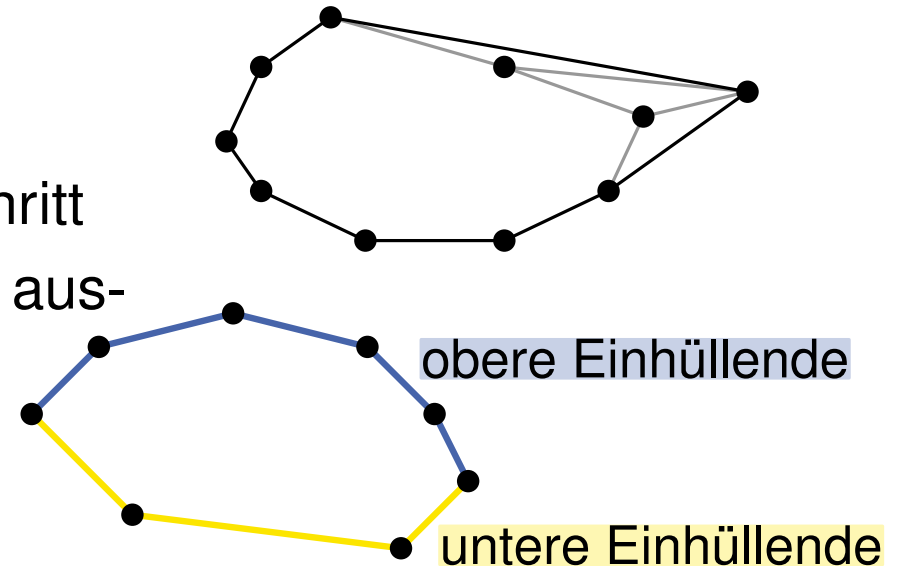
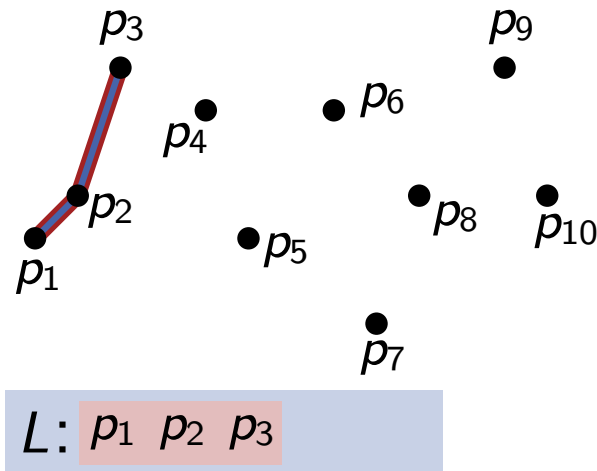
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

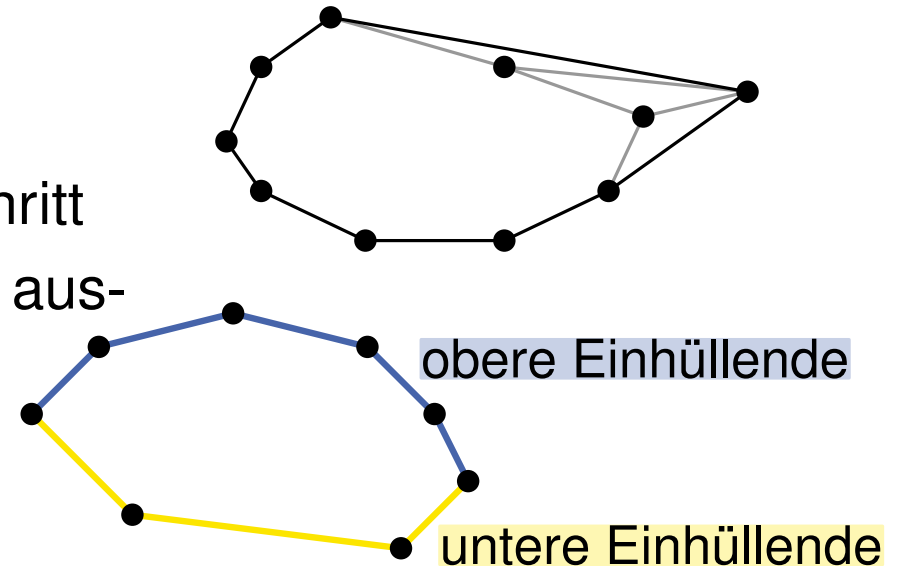
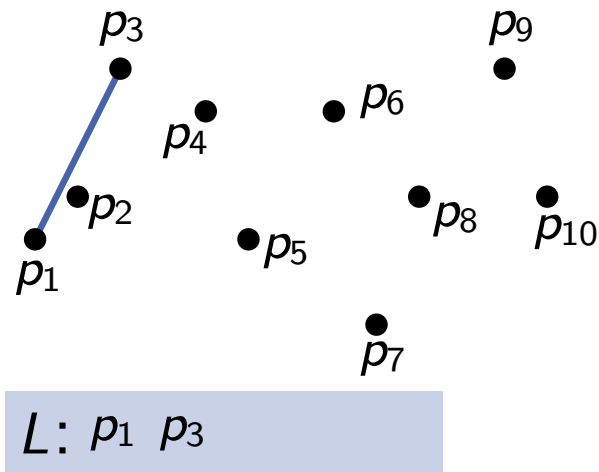
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknick
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

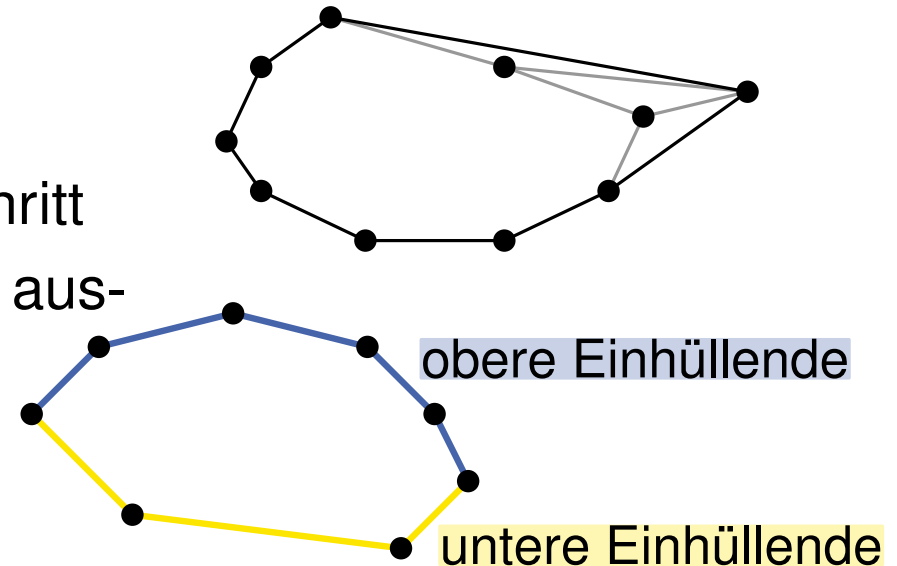
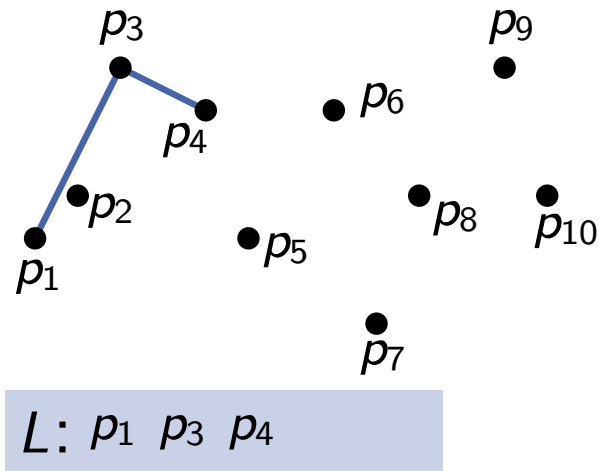
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknick
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

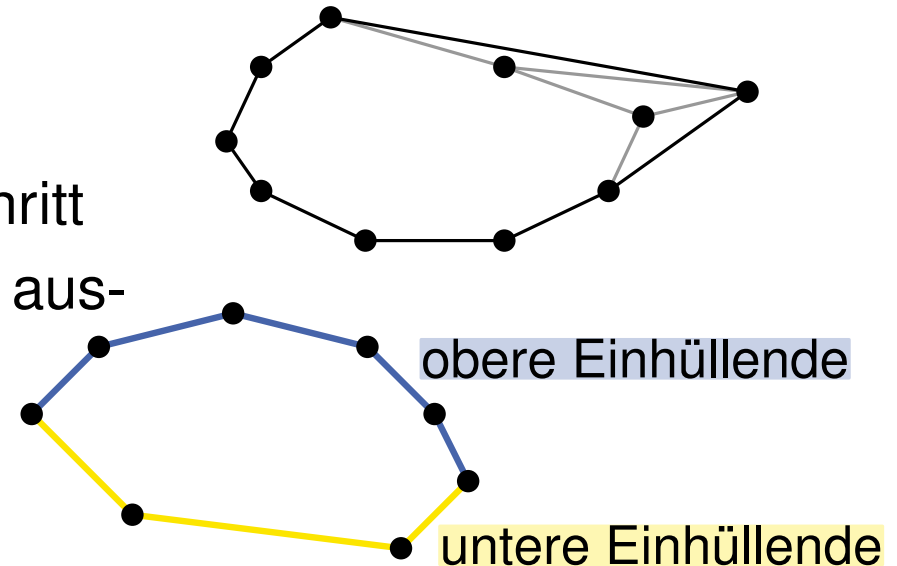
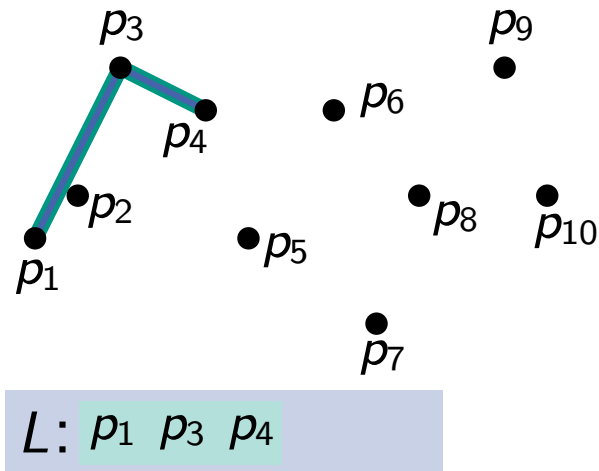
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

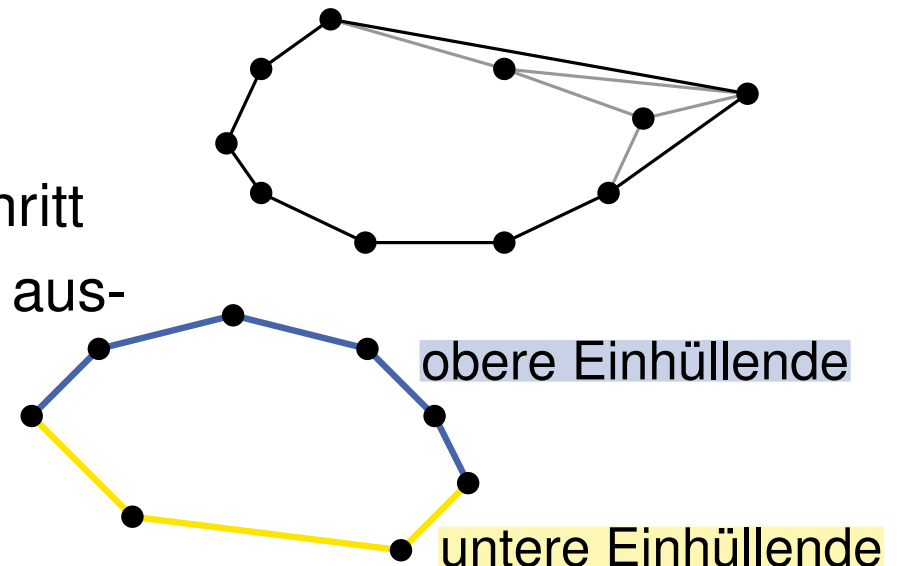
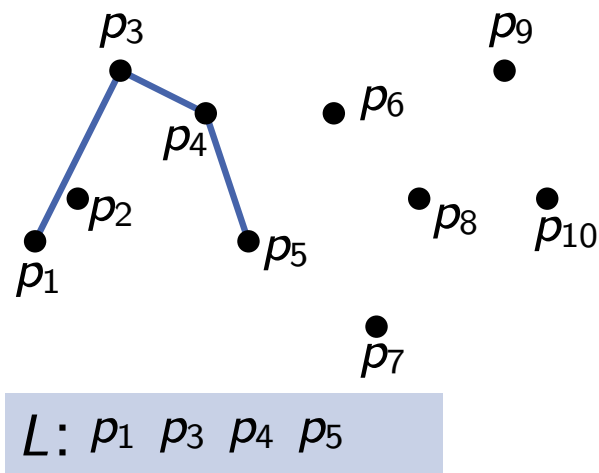
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

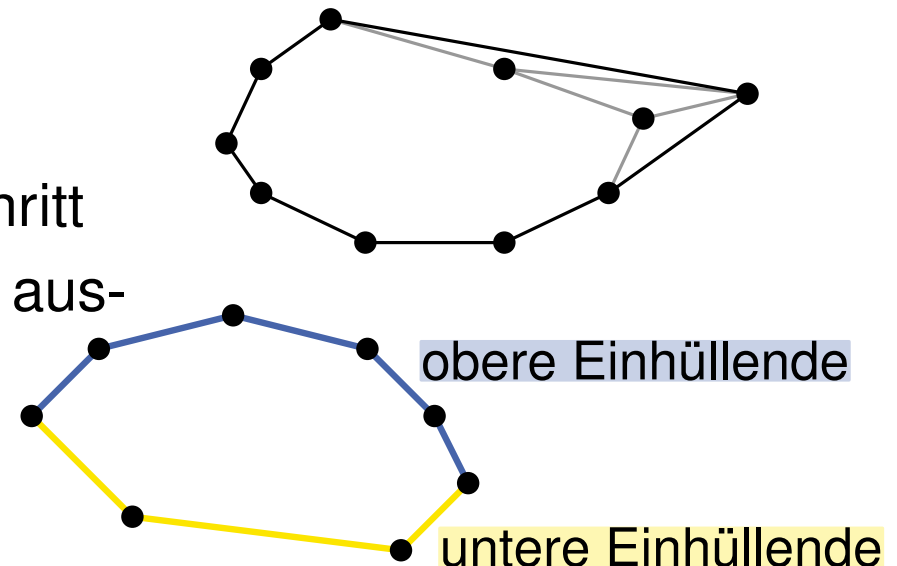
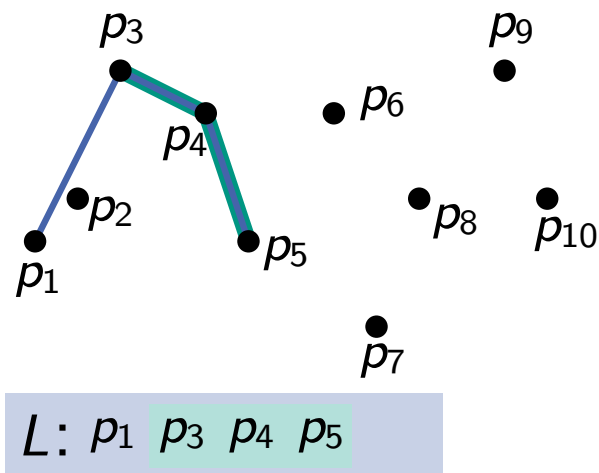
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

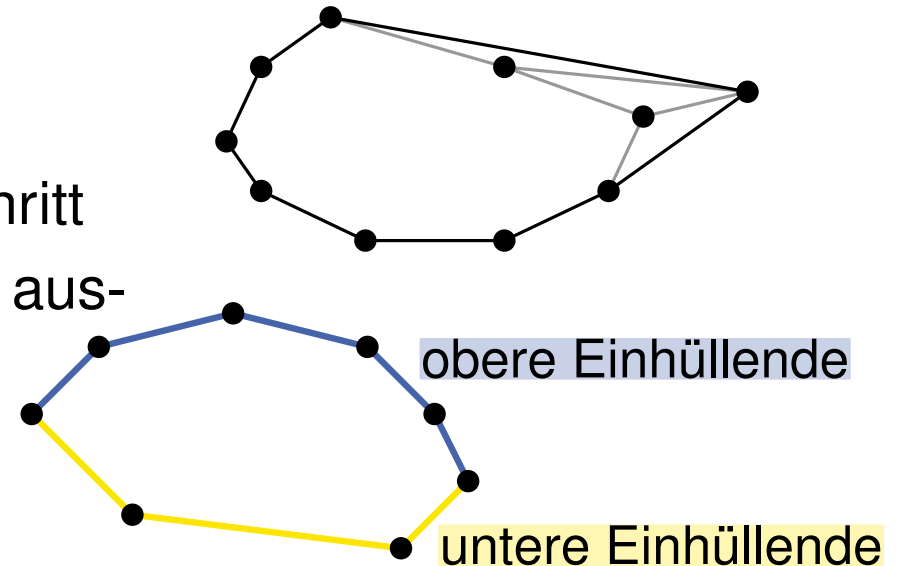
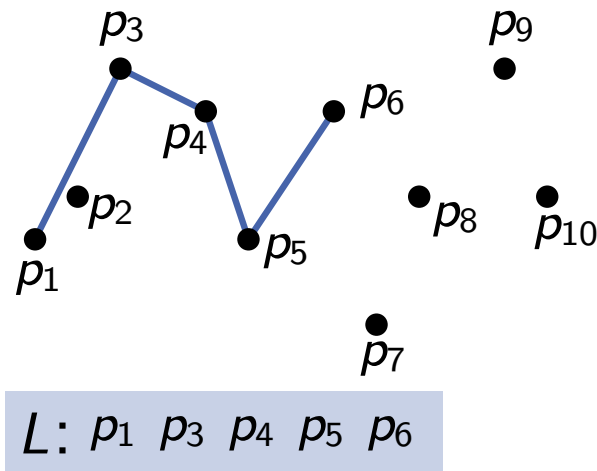
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

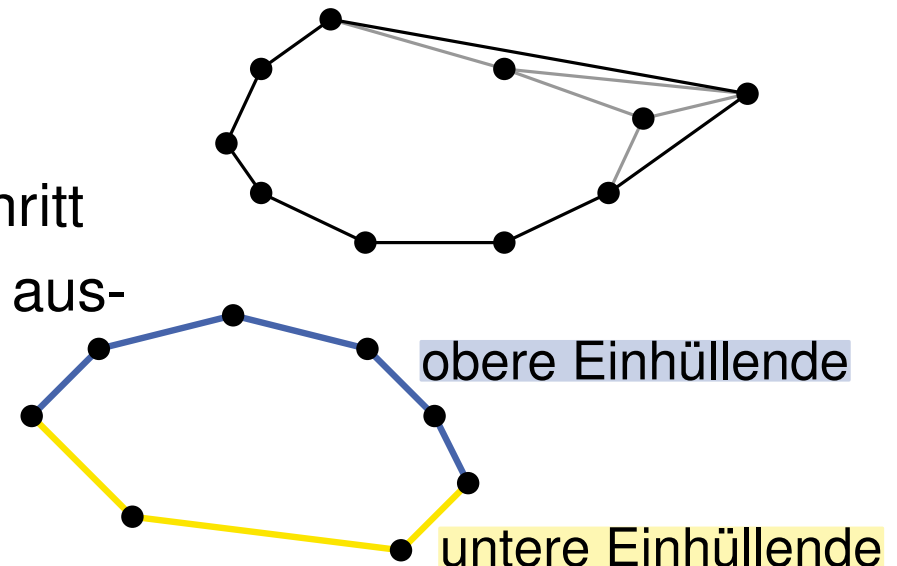
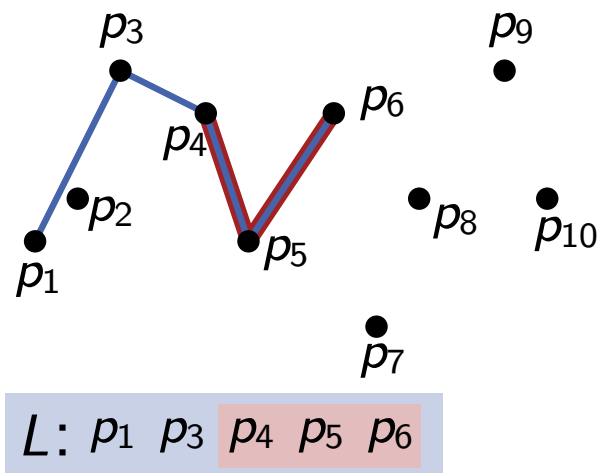
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

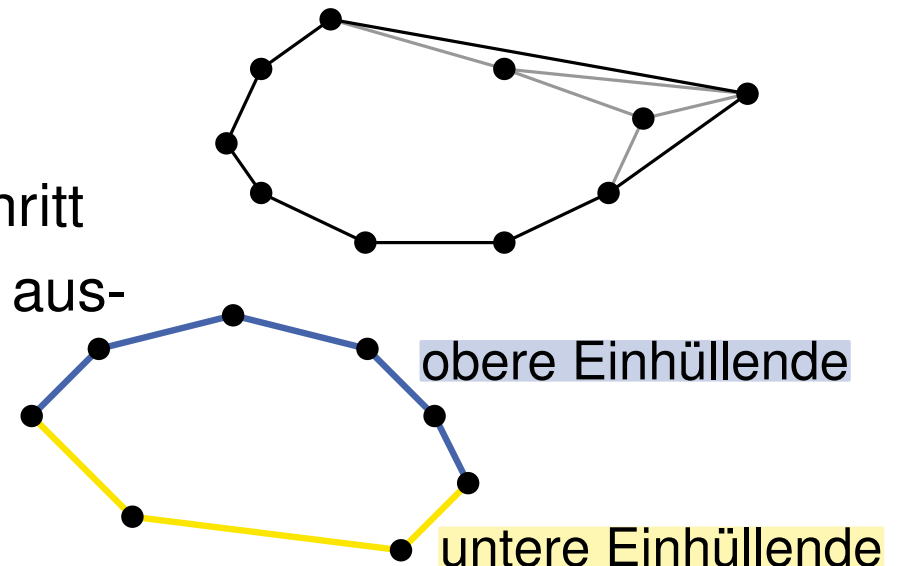
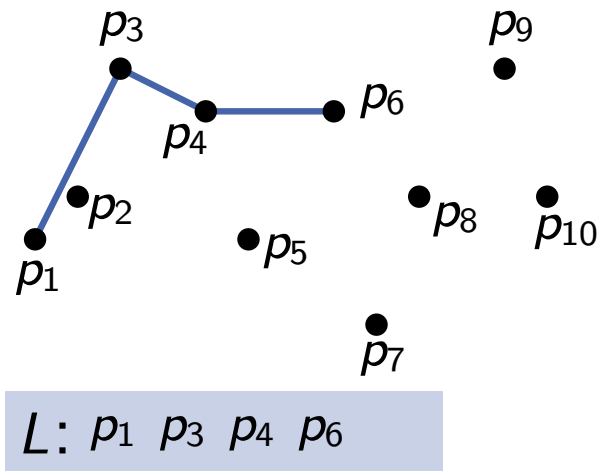
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

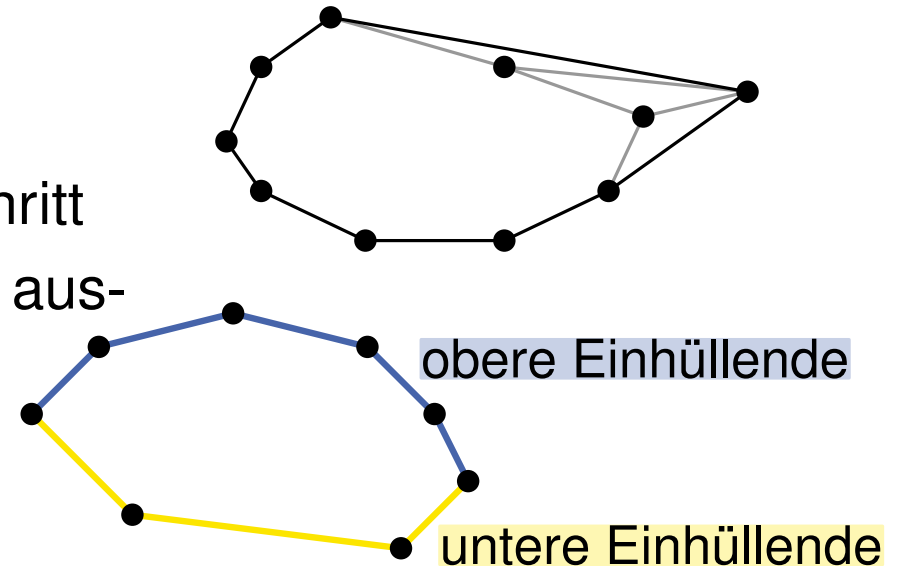
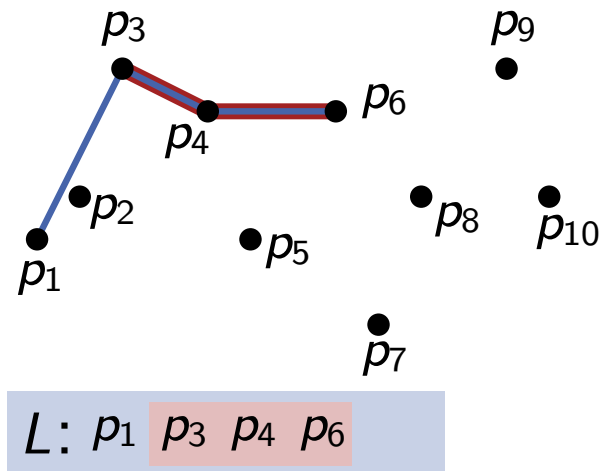
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

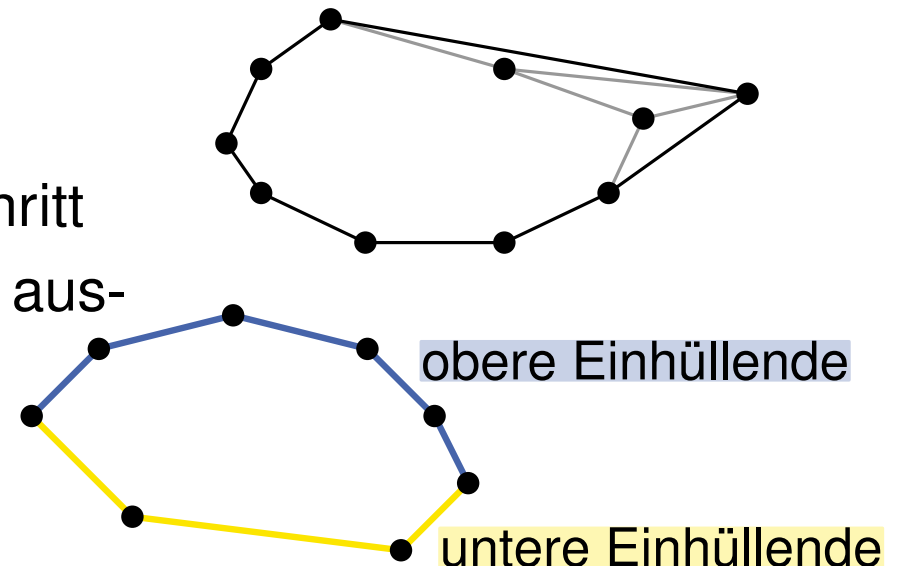
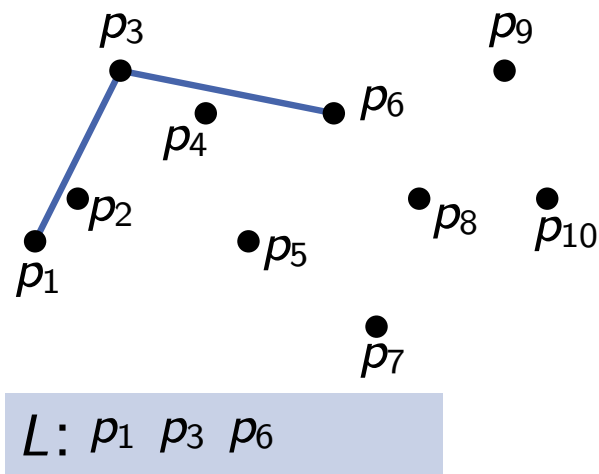
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

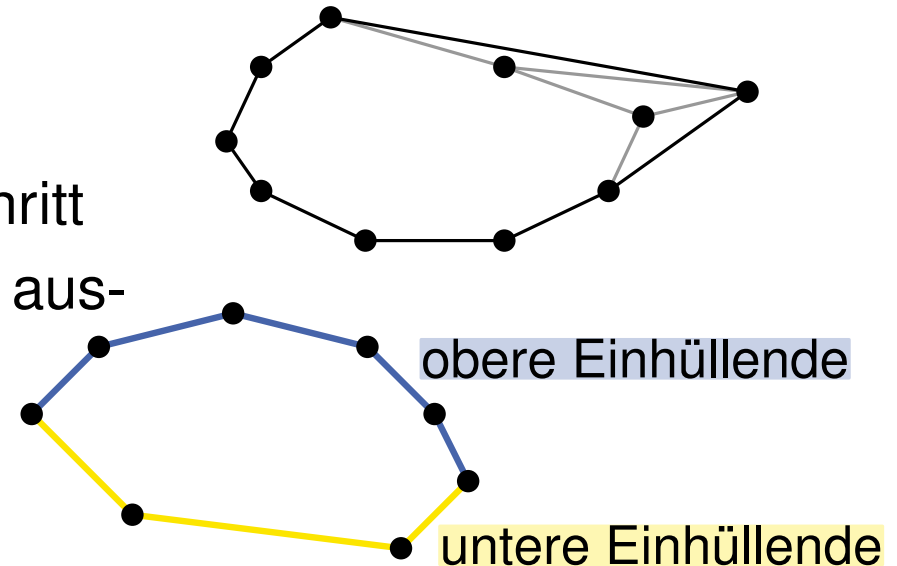
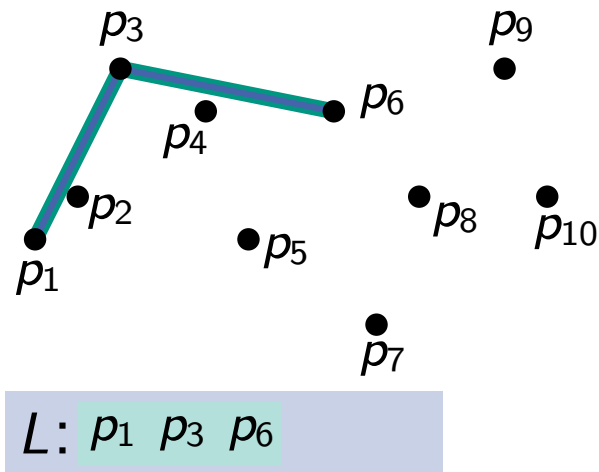
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

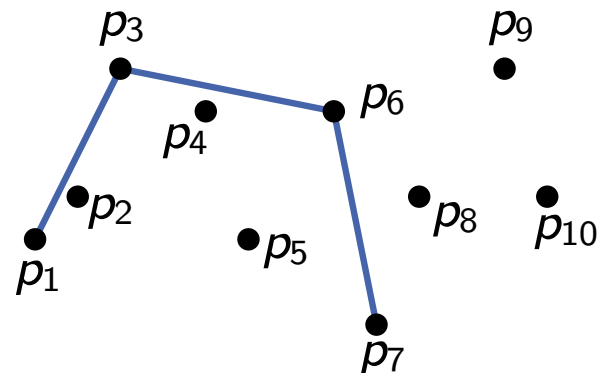
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

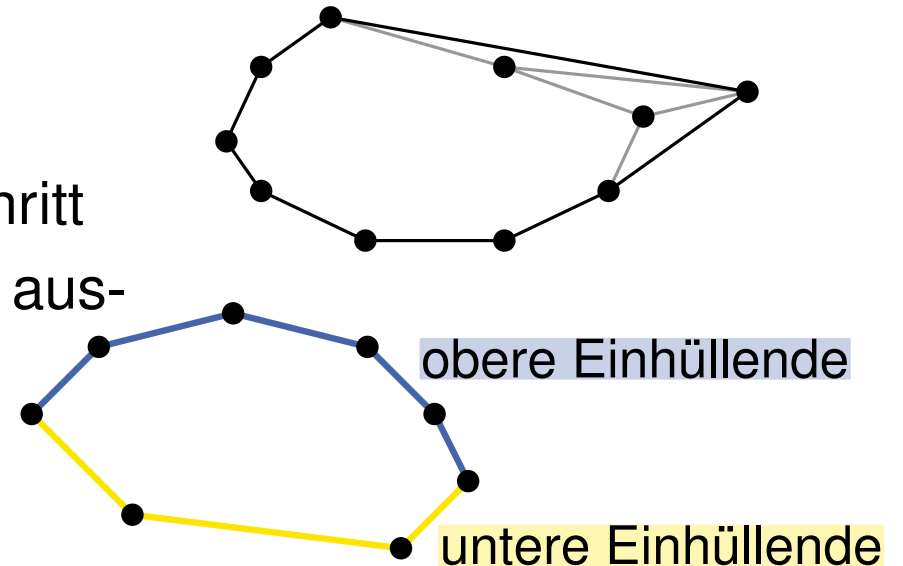
Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknick
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



$L: p_1 p_3 p_6 p_7$



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

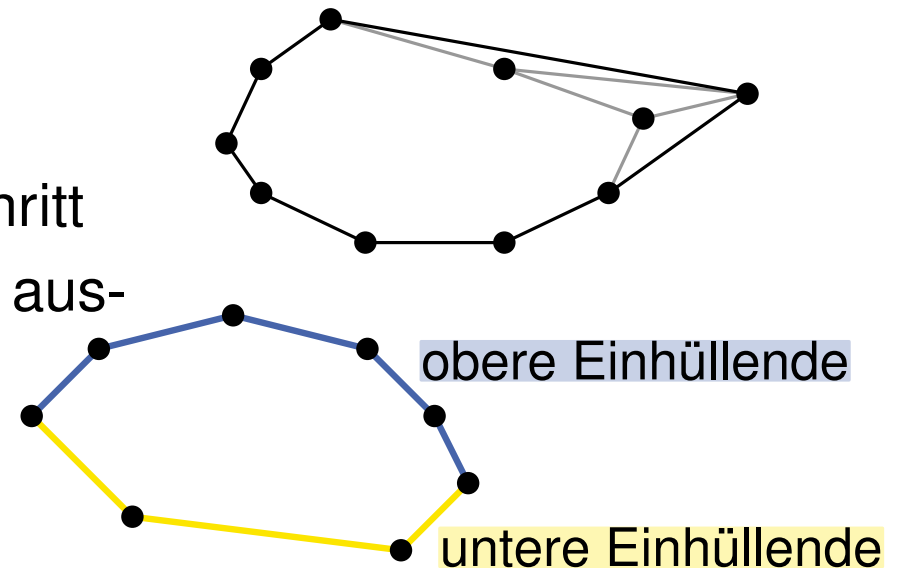
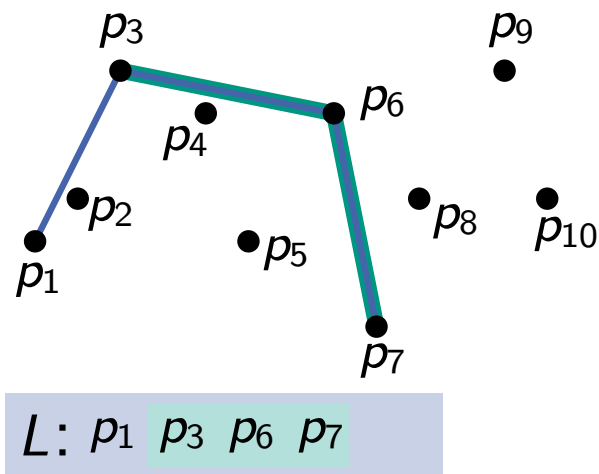
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknick
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

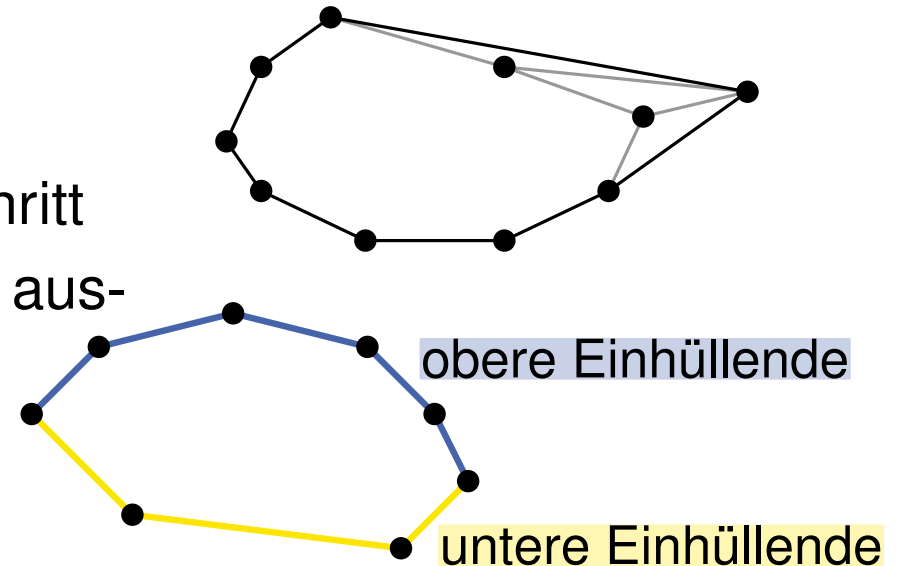
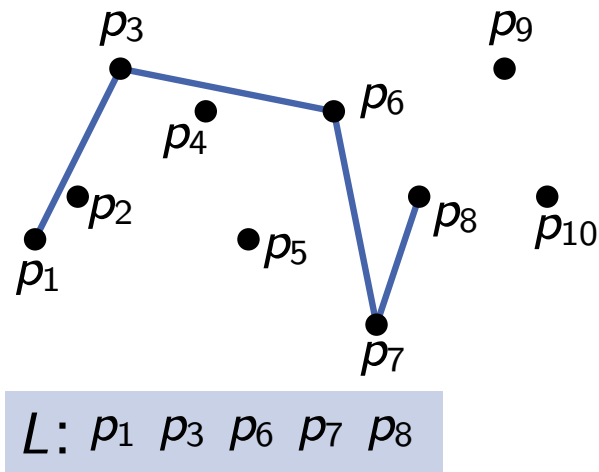
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

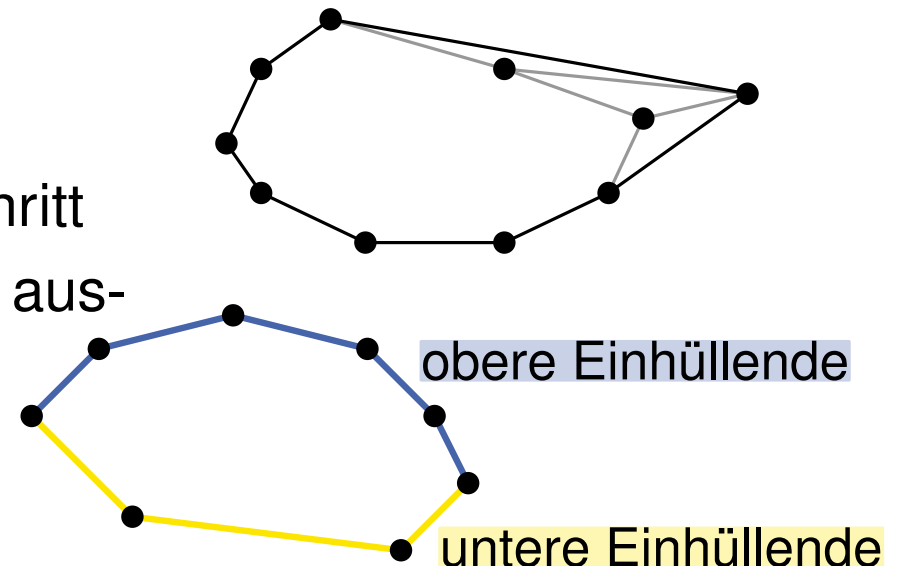
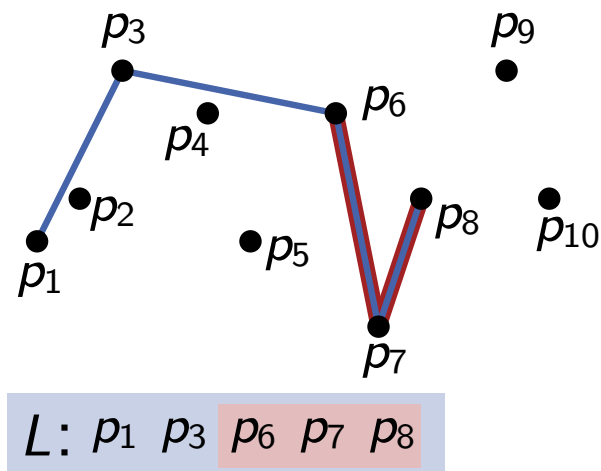
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

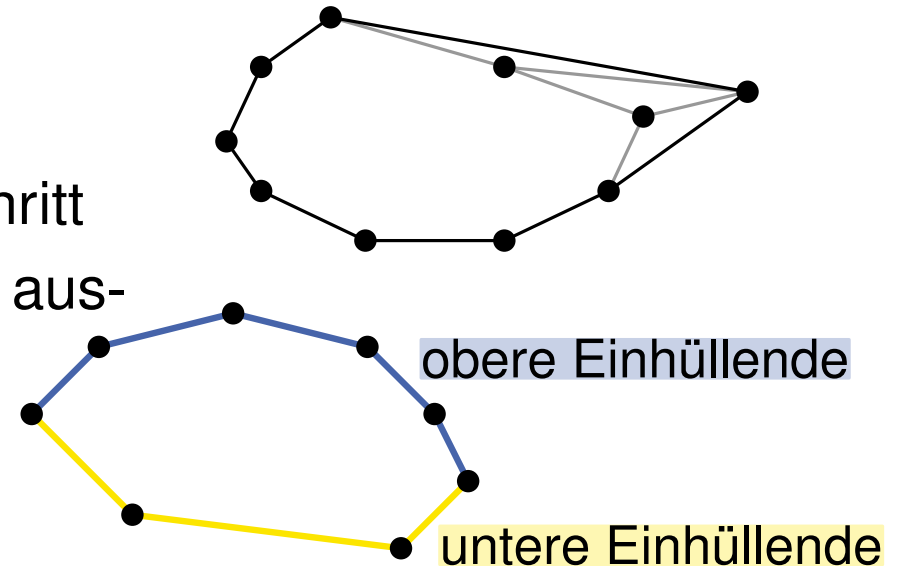
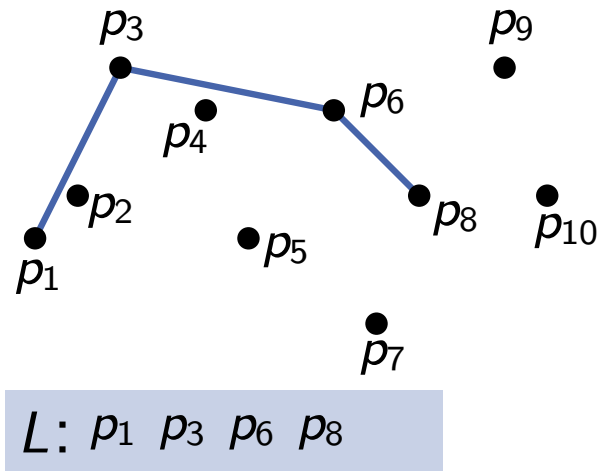
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

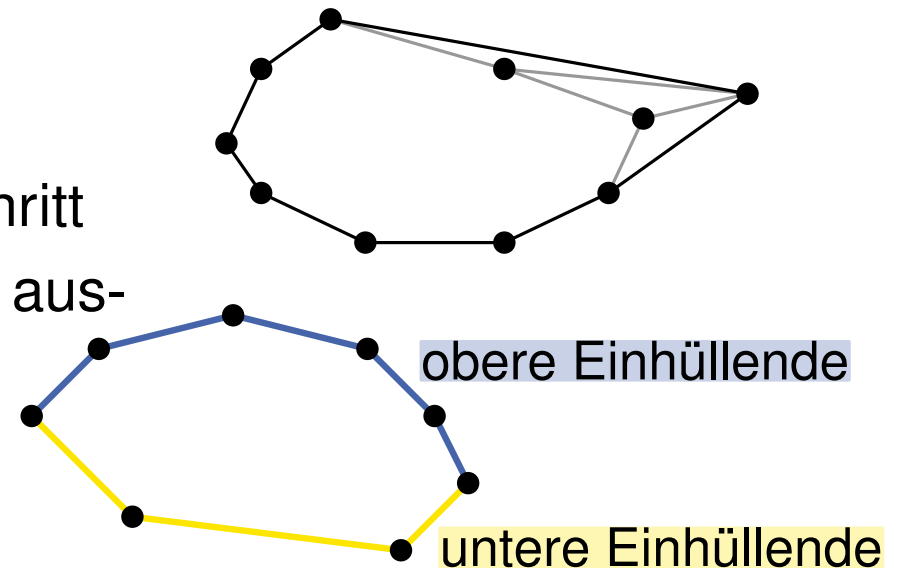
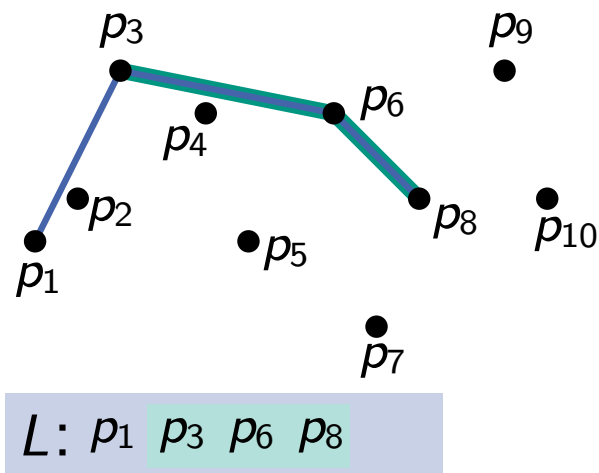
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknick
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

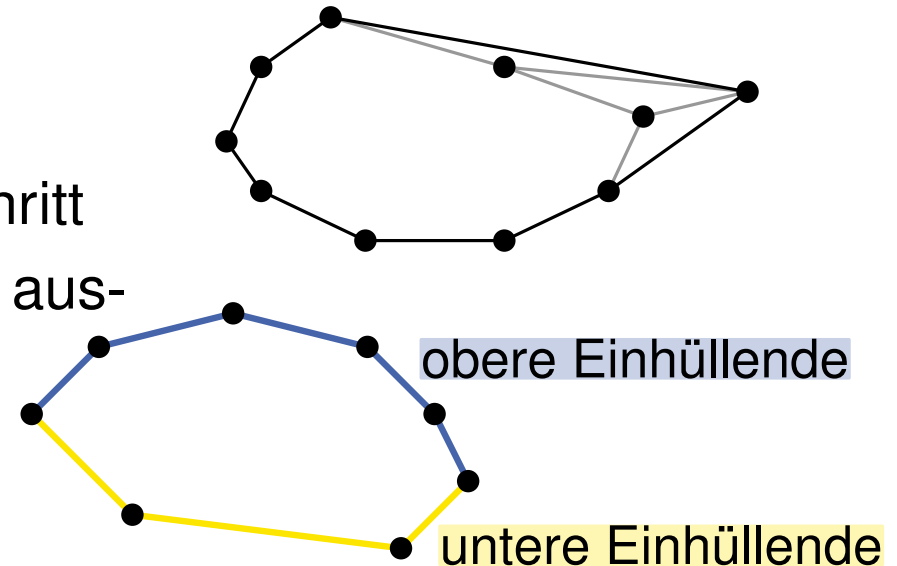
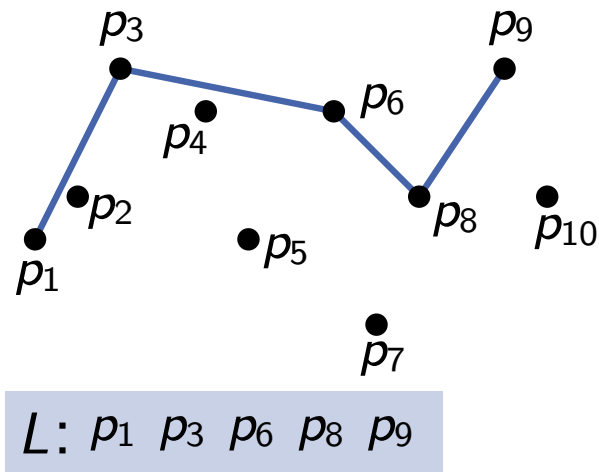
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

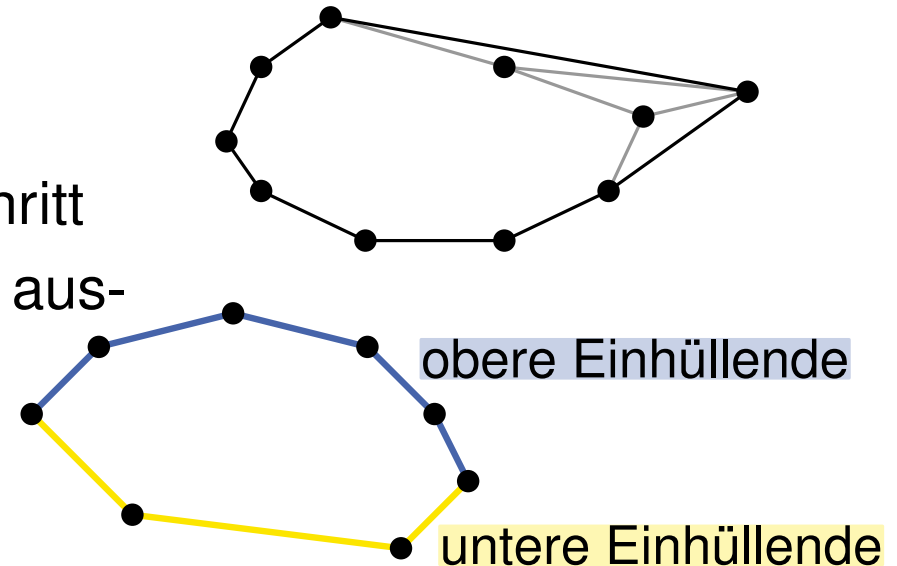
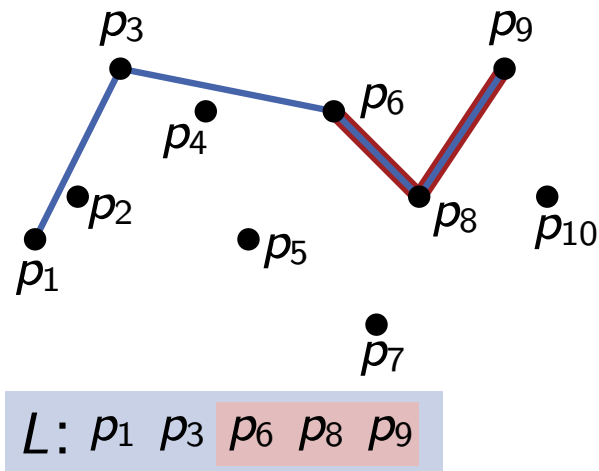
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

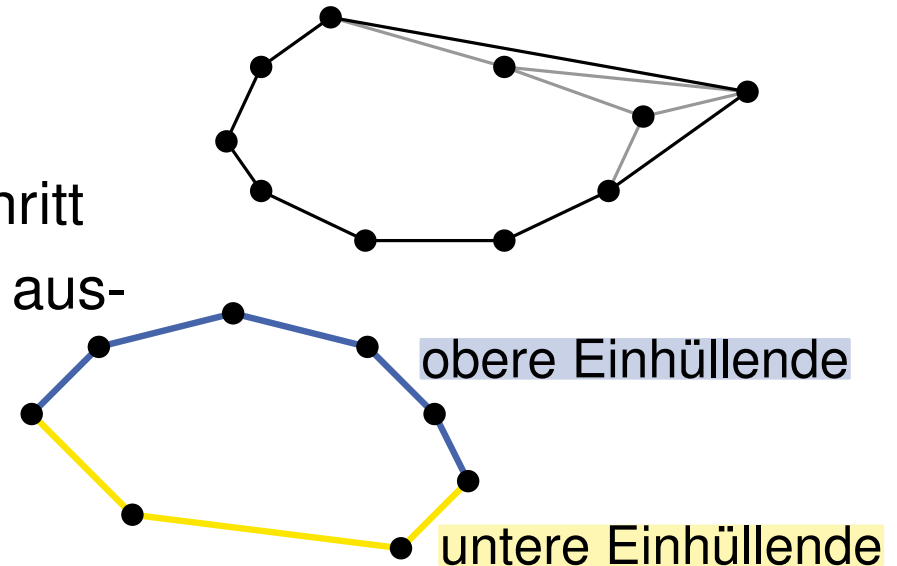
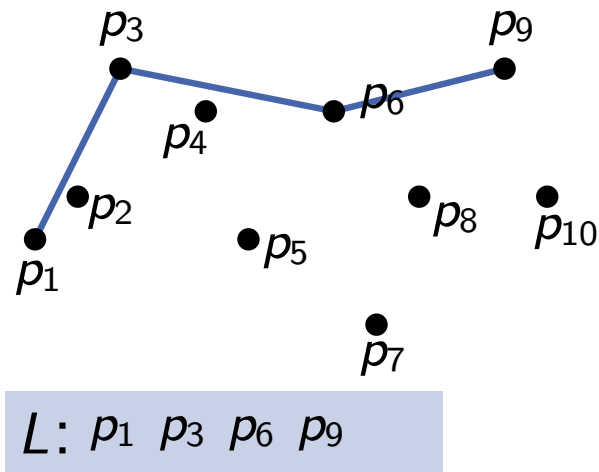
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

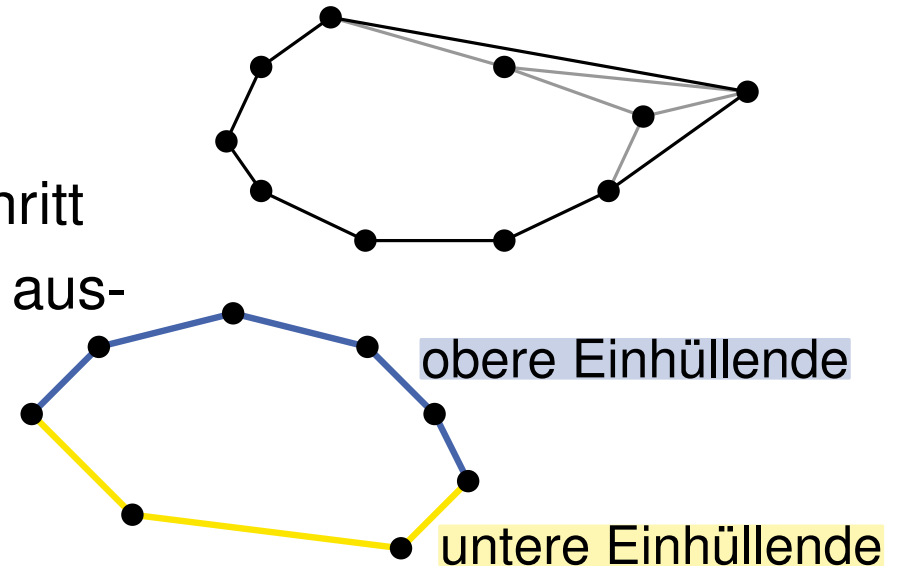
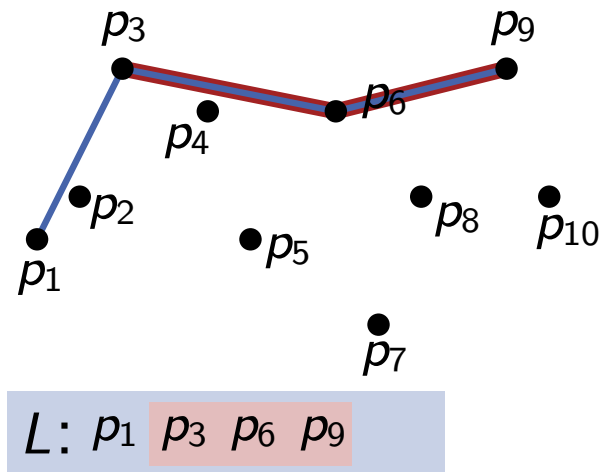
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

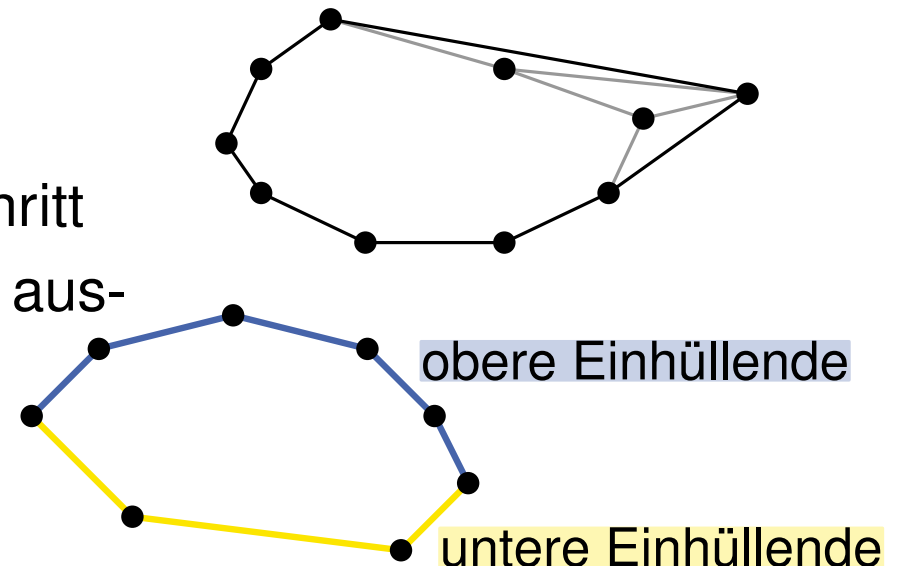
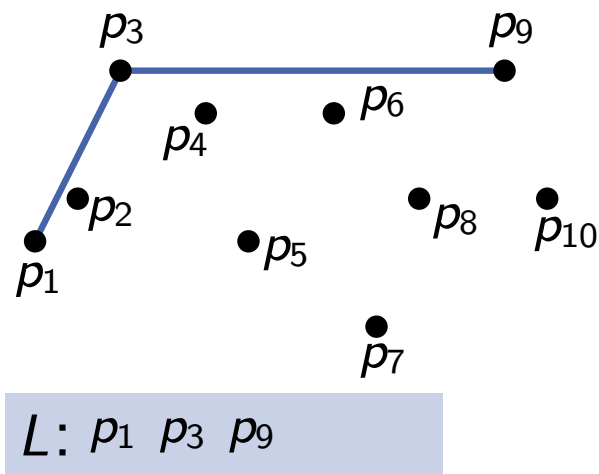
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknick
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

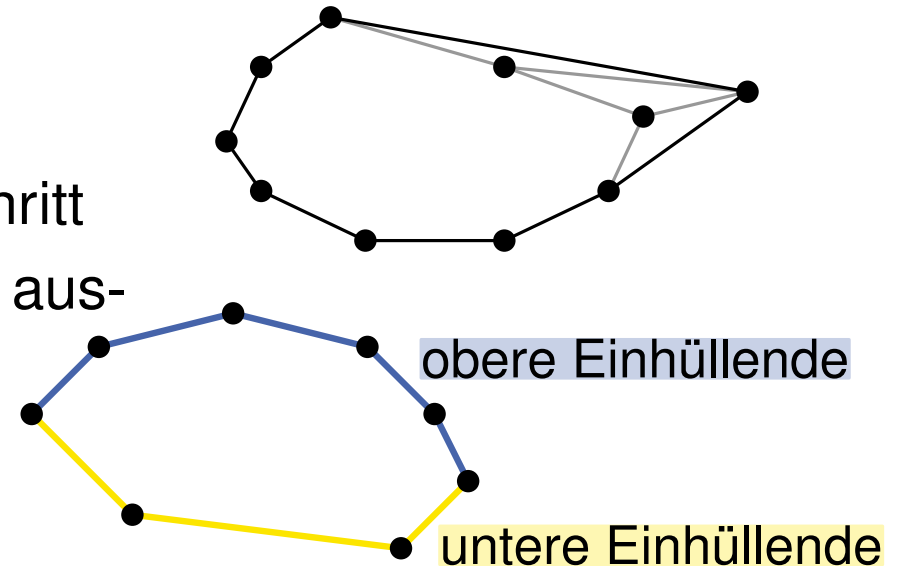
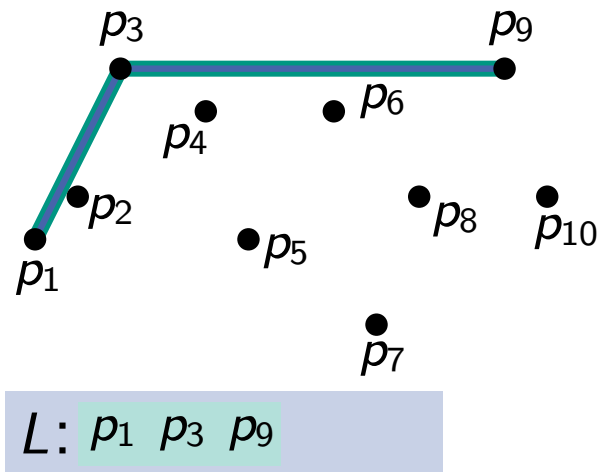
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknick
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

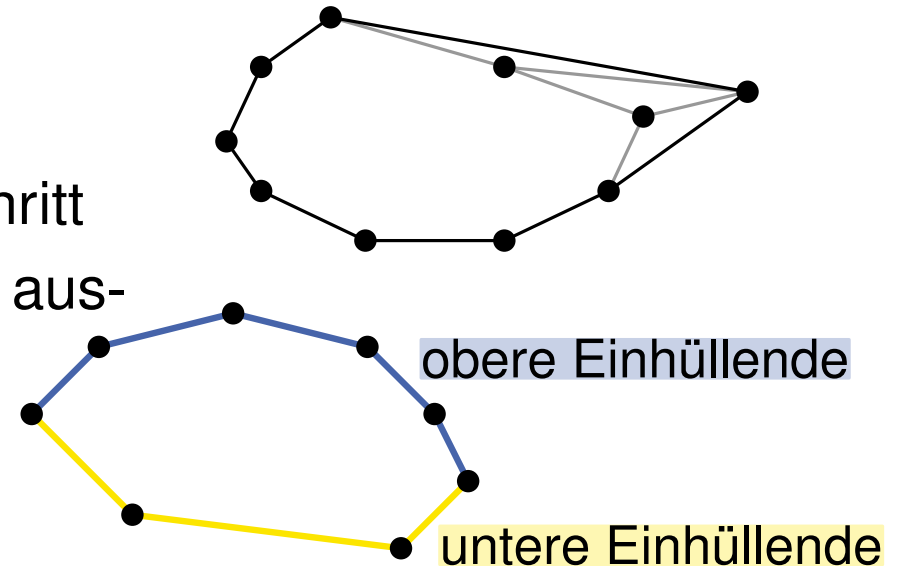
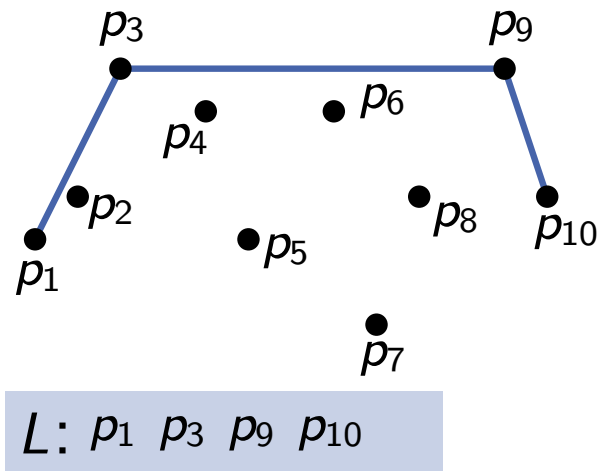
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknick
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

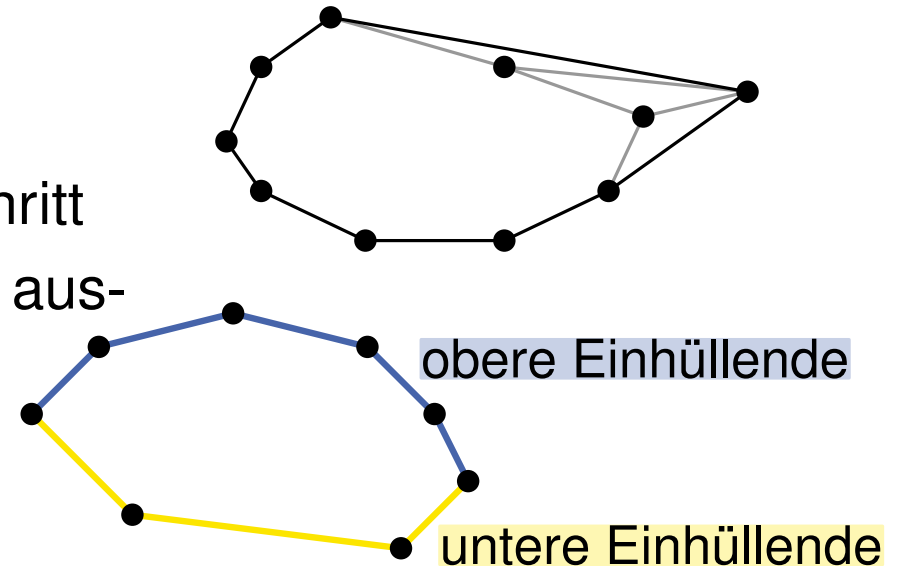
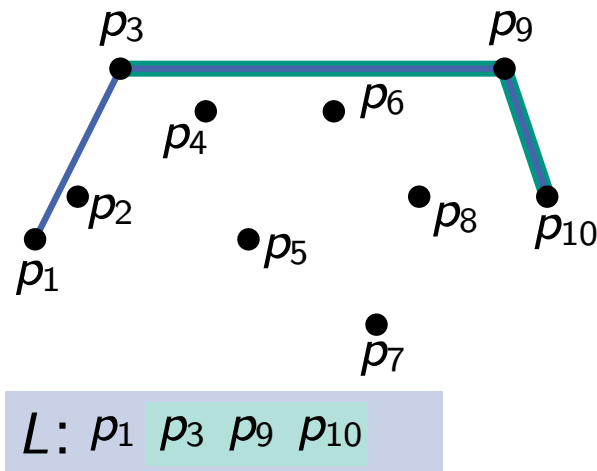
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt

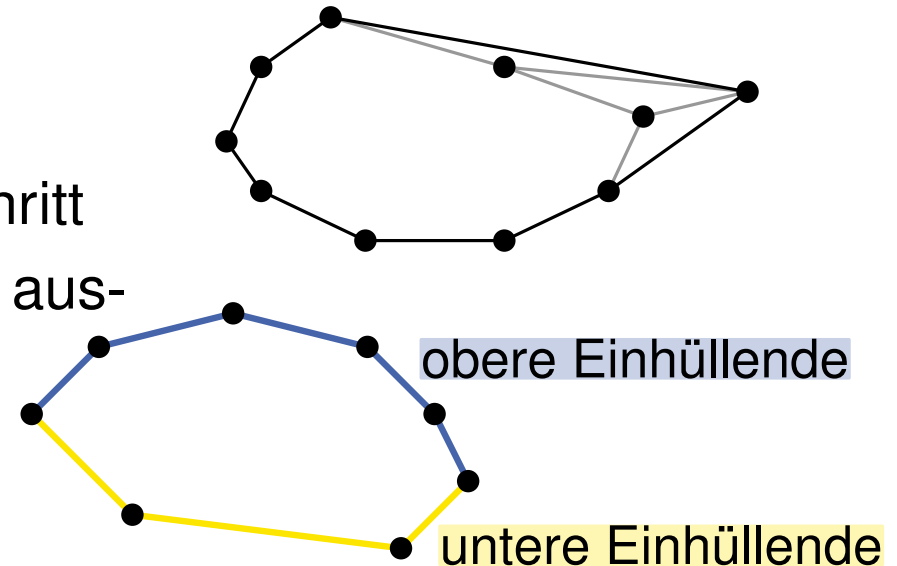
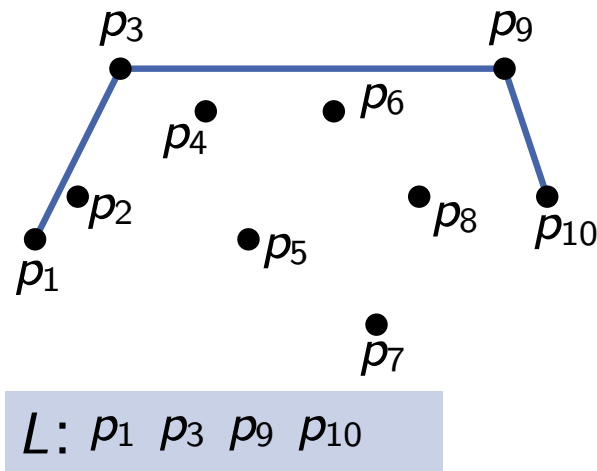
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

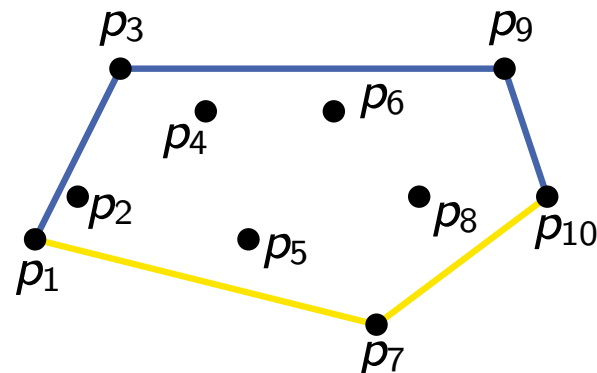
Andrews Monotone Chain Algorithmus

(Variante des Graham Scan)

Idee: iteratives Vorgehen

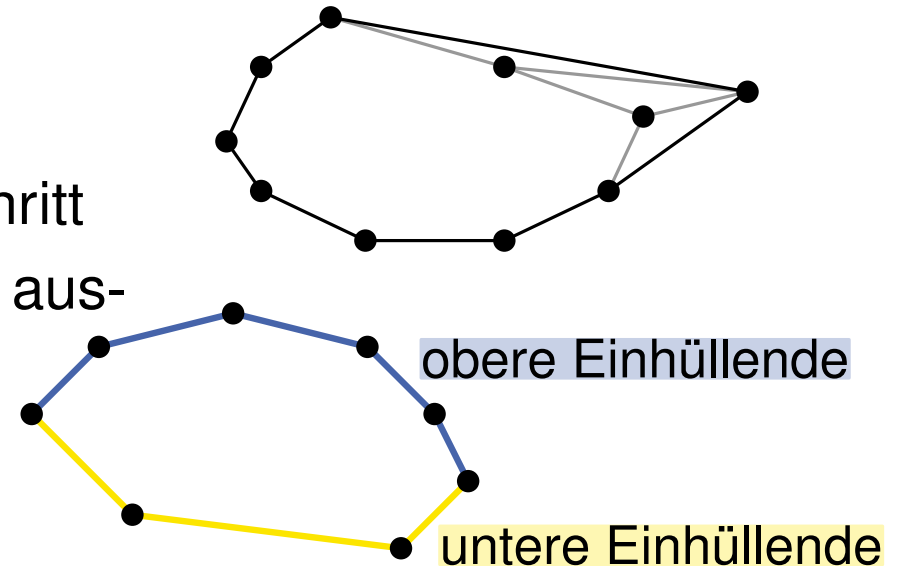
- füge nach und nach Punkte ein
- aktualisiere konvexe Hülle in jedem Schritt
- Beobachtung: die konvexe Hülle macht ausschließlich Rechtsknicke
- Reihenfolge: von links nach rechts
- zunächst: nur die obere Einhüllende

Beispiel



$L: p_1 p_3 p_9 p_{10}$

Analog: untere Einhüllende



Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Andrews Algorithmus – Analyse

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit:

Andrews Algorithmus – Analyse

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit:

$O(n \log n)$

$O(1)$

$O(??)$

$O(1)$

$O(??)$

$O(n)$

Andrews Algorithmus – Analyse

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit:

$O(n \log n)$

$O(1)$

$O(??)$

$O(1)$

$O(??)$

← kann jedem Punkt nur einmal passieren

$O(n)$

Andrews Algorithmus – Analyse

Andrews Algorithmus

- | | |
|--|--|
| ■ sortiere P (bzgl. x): p_1, \dots, p_n | Laufzeit: $O(n \log n)$ |
| ■ füge p_1 und p_2 in Liste L ein | $O(n \log n)$ |
| ■ für jeden weiteren Punkt p_i : | $O(1)$ |
| ■ füge p_i hinten in L ein | $O(n)$ |
| ■ solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt | $O(1)$ (amortisiert) |
| ■ L ist die obere Einhüllende | $O(1)$ kann jedem Punkt nur einmal passieren |
| | $O(n)$ |

Andrews Algorithmus – Analyse

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit: $O(n \log n)$

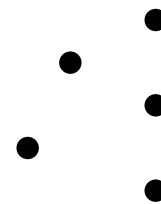
Andrews Algorithmus – Analyse

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit: $O(n \log n)$

Sonderfall: gleiche x -Koordinaten



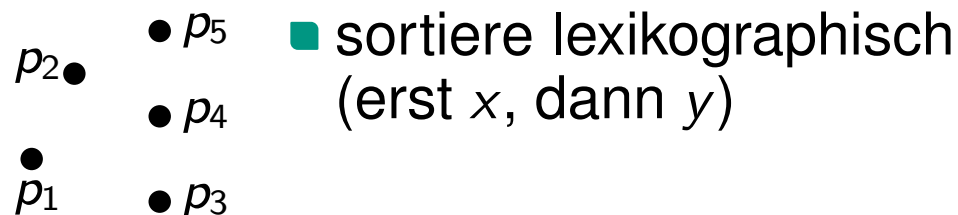
Andrews Algorithmus – Analyse

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit: $O(n \log n)$

Sonderfall: gleiche x -Koordinaten



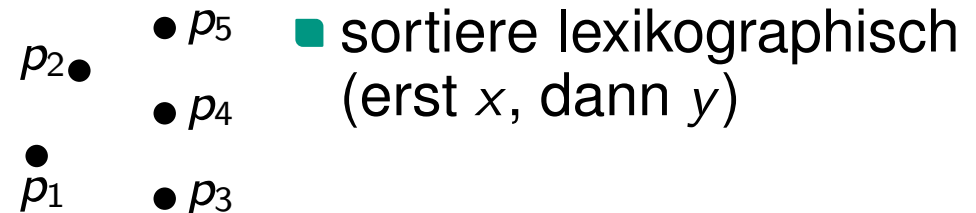
Andrews Algorithmus – Analyse

Andrews Algorithmus

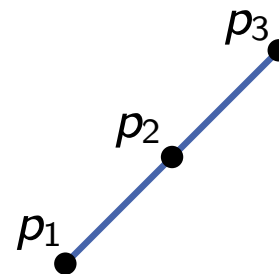
- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit: $O(n \log n)$

Sonderfall: gleiche x -Koordinaten



Sonderfall: kollineare Punkte



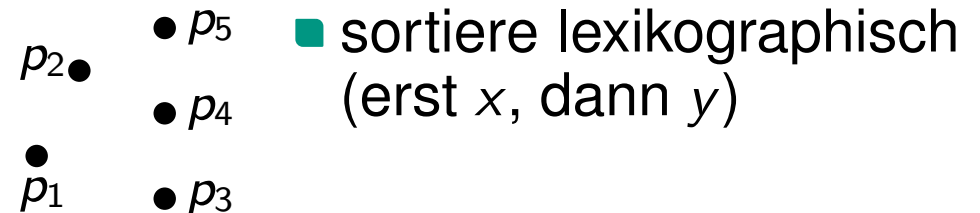
Andrews Algorithmus – Analyse

Andrews Algorithmus

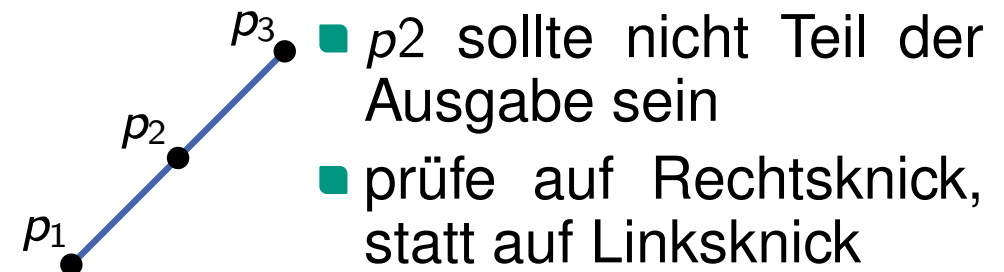
- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit: $O(n \log n)$

Sonderfall: gleiche x -Koordinaten



Sonderfall: kollineare Punkte



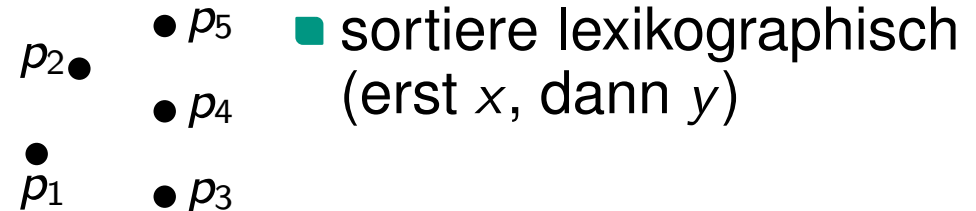
Andrews Algorithmus – Analyse

Andrews Algorithmus

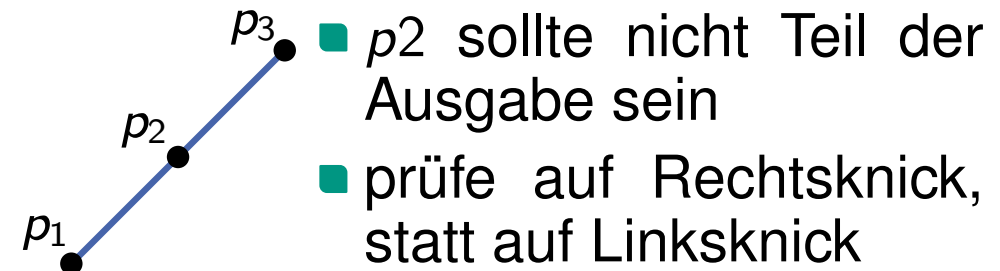
- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit: $O(n \log n)$

Sonderfall: gleiche x -Koordinaten



Sonderfall: kollineare Punkte



Robustheit

- was passiert, wenn eine Prüfung auf Links- vs. Rechtsknicke schief geht?

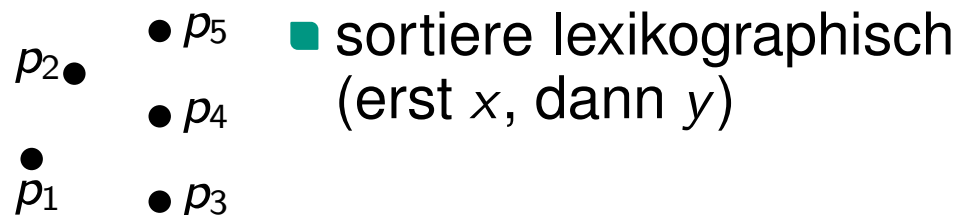
Andrews Algorithmus – Analyse

Andrews Algorithmus

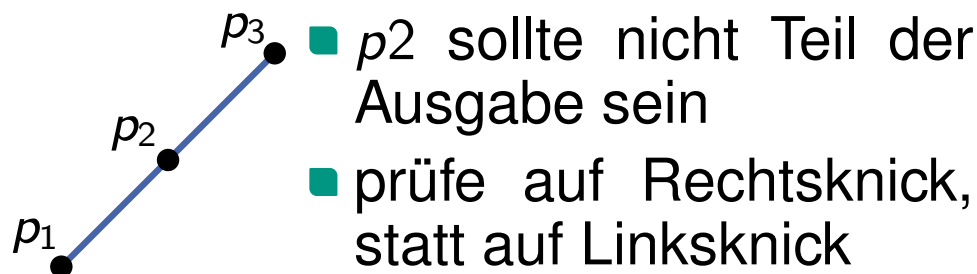
- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit: $O(n \log n)$

Sonderfall: gleiche x -Koordinaten



Sonderfall: kollineare Punkte



Robustheit

- was passiert, wenn eine Prüfung auf Links- vs. Rechtsknicke schief geht?
- das resultierende Polygon hat ggf. einen leichten Linksknicke
- ein Punkt liegt ggf. leicht außerhalb des Polygons

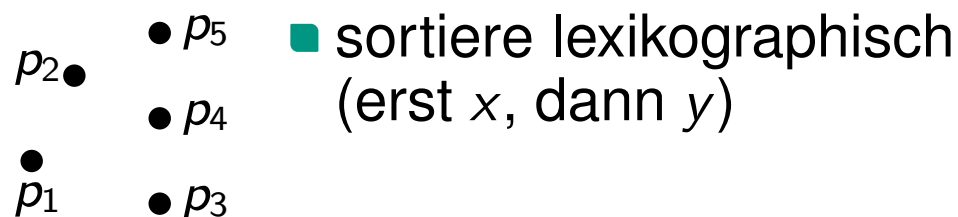
Andrews Algorithmus – Analyse

Andrews Algorithmus

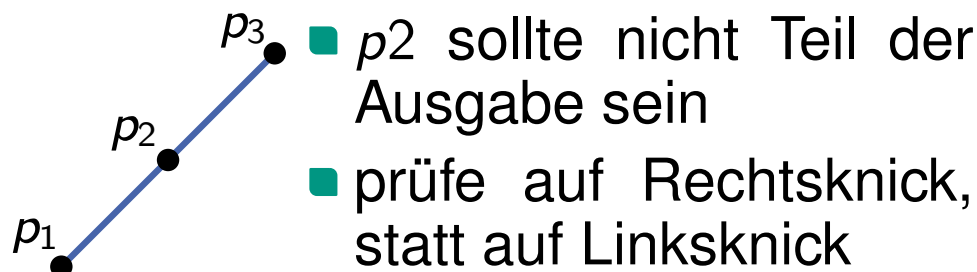
- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Laufzeit: $O(n \log n)$

Sonderfall: gleiche x -Koordinaten



Sonderfall: kollineare Punkte



Robustheit

- was passiert, wenn eine Prüfung auf Links- vs. Rechtsknicke schief geht?
- das resultierende Polygon hat ggf. einen leichten Linksknicke
- ein Punkt liegt ggf. leicht außerhalb des Polygons
- aber: das Resultat ist in jedem Fall ein Polygon, das ähnlich zu $\mathcal{CH}(P)$ ist

Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L

Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$

Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Nach Schritt i startet L bei p_1 und endet bei p_i

Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$

Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Nach Schritt i startet L bei p_1 und endet bei p_i

- offensichtlich, da erster und letzter Punkt nie gelöscht werden

Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$

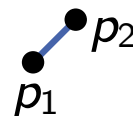
Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Nach Schritt i macht L nur Rechtsknicke

- stimmt nach der Initialisierung ($i = 2$)



Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$

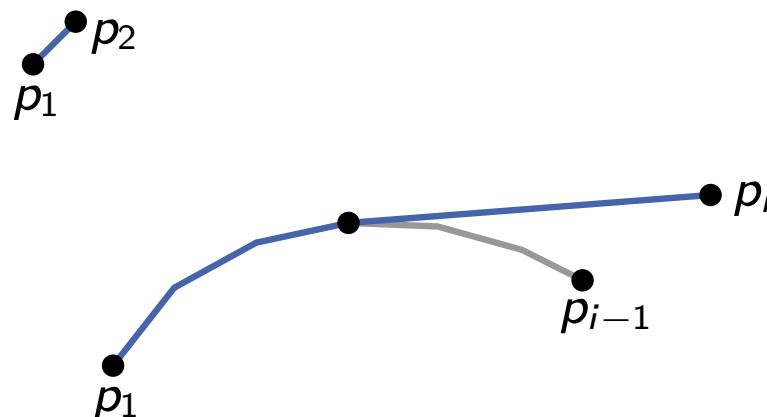
Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Nach Schritt i macht L nur Rechtsknicke

- stimmt nach der Initialisierung ($i = 2$)
- nach Schritt i besteht L aus zwei Teilen
 - Präfix des Polygons L aus dem vorherigen Schritt $i - 1$
 - Kante zu p_i



Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$

Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Nach Schritt i macht L nur Rechtsknicke

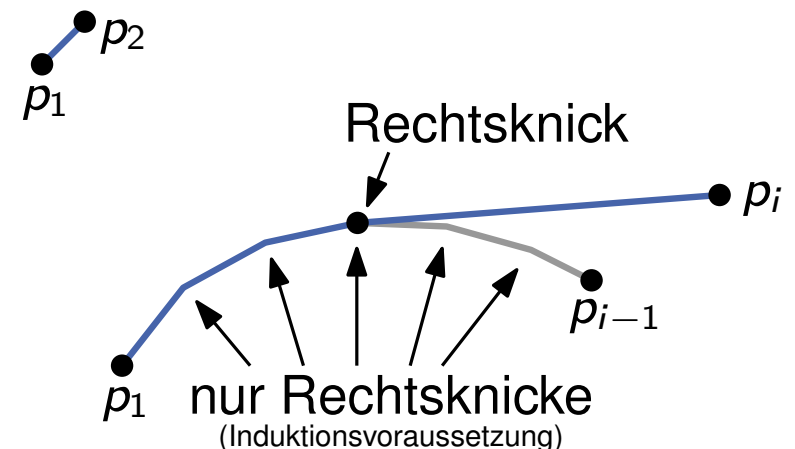
- stimmt nach der Initialisierung ($i = 2$)
- nach Schritt i besteht L aus zwei Teilen
 - Präfix des Polygons L aus dem vorherigen Schritt $i - 1$
 - Kante zu p_i

Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$



Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Nach Schritt i macht L nur Rechtsknicke

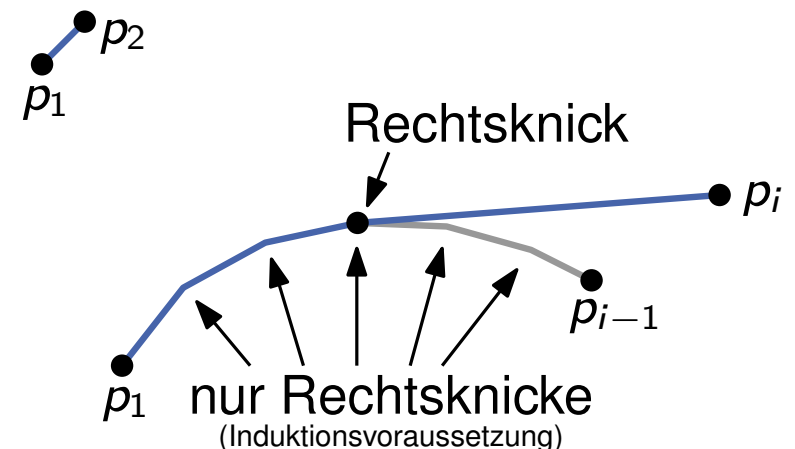
- stimmt nach der Initialisierung ($i = 2$)
 - nach Schritt i besteht L aus zwei Teilen
 - Präfix des Polygons L aus dem vorherigen Schritt $i - 1$
 - Kante zu p_i
- ⇒ nur Rechtsknicke

Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$



Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Lemma

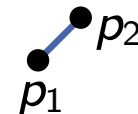
Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$

Nach Schritt i liegt jeder Punkt aus $P_i \setminus L$ unter L

- stimmt nach der Initialisierung ($i = 2$)



Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Nach Schritt i liegt jeder Punkt aus $P_i \setminus L$ unter L

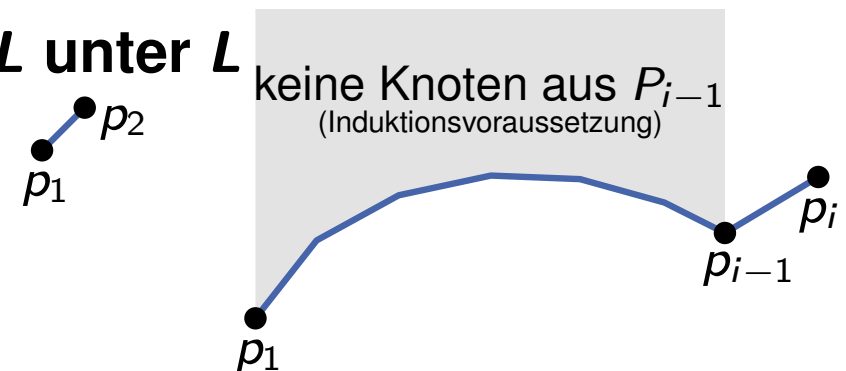
- stimmt nach der Initialisierung ($i = 2$)
- stimmt auch nach dem Einfügen von p_i

Lemma

Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$



Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Lemma

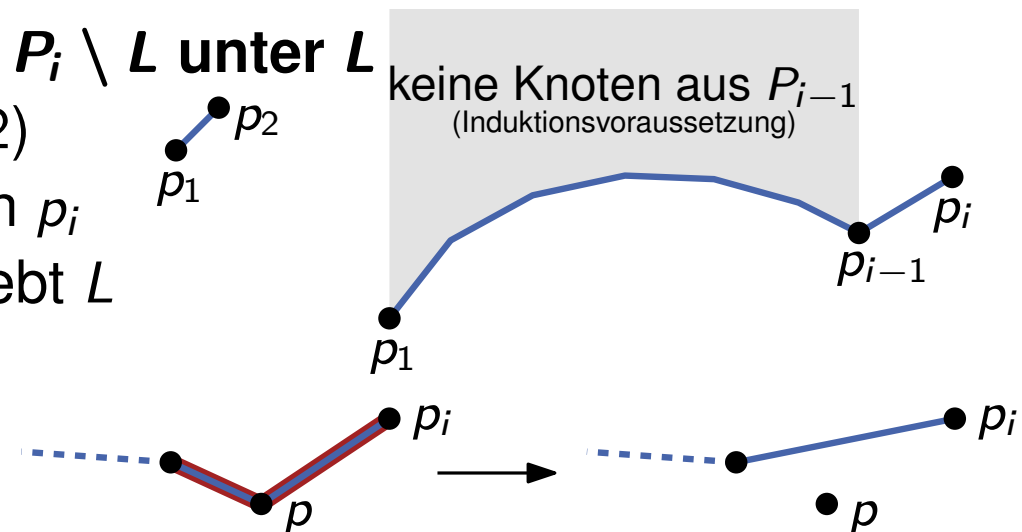
Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$

Nach Schritt i liegt jeder Punkt aus $P_i \setminus L$ unter L

- stimmt nach der Initialisierung ($i = 2$)
- stimmt auch nach dem Einfügen von p_i
- Löschen eines Punkts p aus L schiebt L nur weiter nach oben
- außerdem liegt p danach unter L



Andrews Algorithmus – Korrektheit

Andrews Algorithmus

- sortiere P (bzgl. x): p_1, \dots, p_n
- füge p_1 und p_2 in Liste L ein
- für jeden weiteren Punkt p_i :
 - füge p_i hinten in L ein
 - solange Linksknick bei letzten drei Punkten: entferne den vorletzten Punkt
- L ist die obere Einhüllende

Lemma

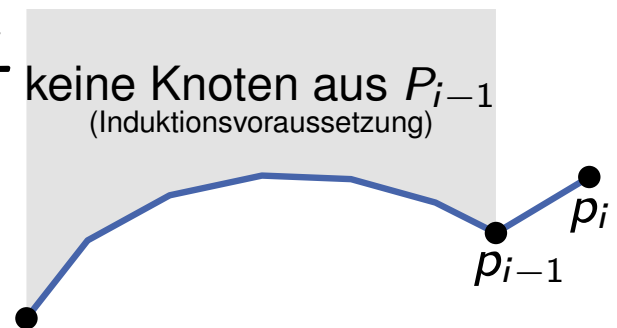
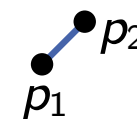
Der Algorithmus berechnet die obere Einhüllende L von P .

Beweis

- zeige: L verbindet p_1 mit p_n , sodass
 - L macht nur Rechtsknicke
 - jeder Punkt aus $P \setminus L$ liegt unter L
- zeige diese Aussagen per Induktion über i für die Punkte $P_i = \{p_1, \dots, p_i\}$

Nach Schritt i liegt jeder Punkt aus $P_i \setminus L$ unter L

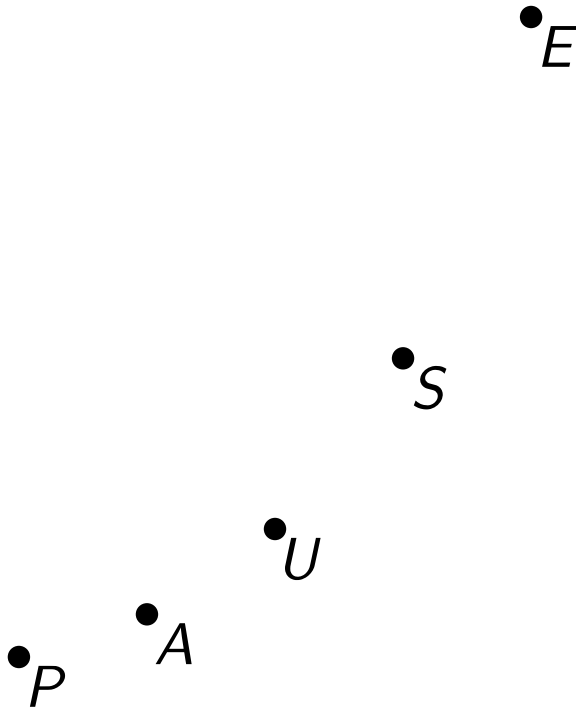
- stimmt nach der Initialisierung ($i = 2$)
- stimmt auch nach dem Einfügen von p_i
- Löschen eines Punkts p aus L schiebt L



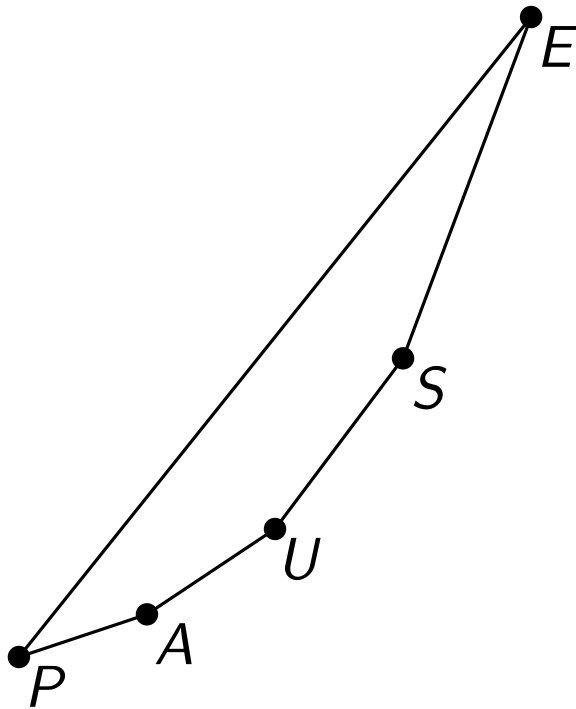
Theorem

Andrews Algo berechnet die konvexe Hülle von n Punkten in $O(n \log n)$.

Berechne die konvexe Hülle



Berechne die konvexe Hülle



Geht es schneller?

Geht es schneller?

Theorem

Wenn die konvexe Hülle von n Punkten in $f(n)$ Zeit berechnet werden kann, dann kann man auch n Zahlen in $O(f(n) + n)$ Zeit sortieren.

Beweis

Geht es schneller?

Theorem

Wenn die konvexe Hülle von n Punkten in $f(n)$ Zeit berechnet werden kann, dann kann man auch n Zahlen in $O(f(n) + n)$ Zeit sortieren.

Beweis

- gegeben: n Zahlen a_1, \dots, a_n
- konstruiere n Punkte $P = \{p_1, \dots, p_n\}$

Geht es schneller?

Theorem

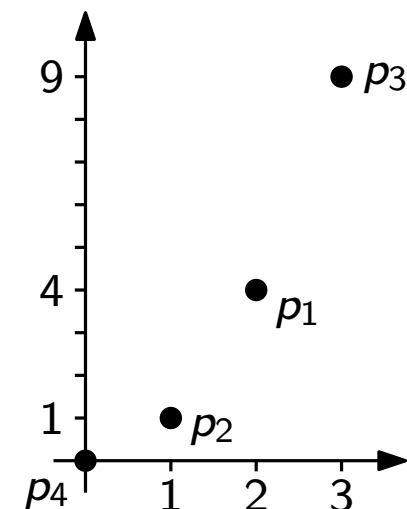
Wenn die konvexe Hülle von n Punkten in $f(n)$ Zeit berechnet werden kann, dann kann man auch n Zahlen in $O(f(n) + n)$ Zeit sortieren.

Beweis

- gegeben: n Zahlen a_1, \dots, a_n
- konstruiere n Punkte $P = \{p_1, \dots, p_n\}$ mit $p_i = (a_i, a_i^2)$

Beispiel

$$a_1 = 2, a_2 = 1, a_3 = 3, a_4 = 0$$



Geht es schneller?

Theorem

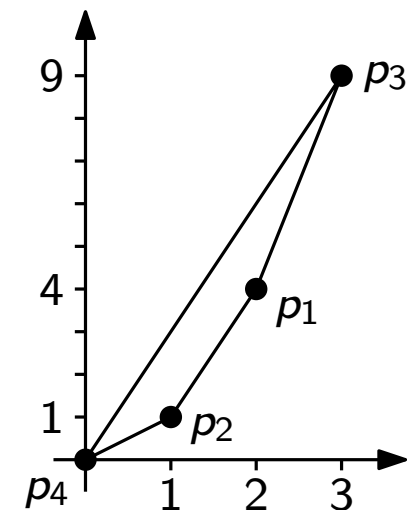
Wenn die konvexe Hülle von n Punkten in $f(n)$ Zeit berechnet werden kann, dann kann man auch n Zahlen in $O(f(n) + n)$ Zeit sortieren.

Beweis

- gegeben: n Zahlen a_1, \dots, a_n
- konstruiere n Punkte $P = \{p_1, \dots, p_n\}$ mit $p_i = (a_i, a_i^2)$
- $\mathcal{CH}(P)$ enthält die Punkte sortiert nach a_i

Beispiel

$a_1 = 2, a_2 = 1, a_3 = 3, a_4 = 0$



Geht es schneller?

Theorem

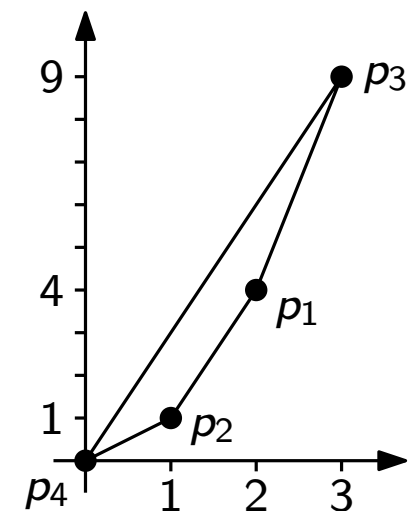
Wenn die konvexe Hülle von n Punkten in $f(n)$ Zeit berechnet werden kann, dann kann man auch n Zahlen in $O(f(n) + n)$ Zeit sortieren.

Beweis

- gegeben: n Zahlen a_1, \dots, a_n
- konstruiere n Punkte $P = \{p_1, \dots, p_n\}$ mit $p_i = (a_i, a_i^2)$
- $\mathcal{CH}(P)$ enthält die Punkte sortiert nach a_i
- Sortierung lässt sich in $O(n)$ aus $\mathcal{CH}(P)$ ablesen

Beispiel

$$a_1 = 2, a_2 = 1, a_3 = 3, a_4 = 0$$



Geht es schneller?

Theorem

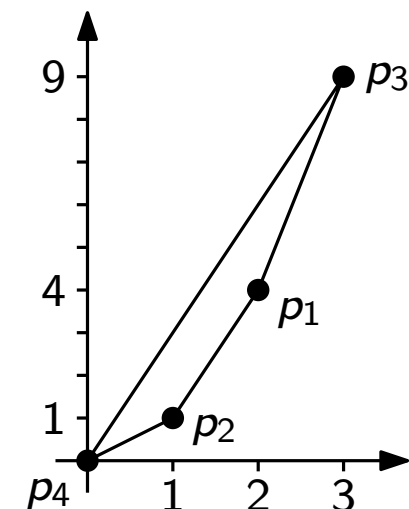
Wenn die konvexe Hülle von n Punkten in $f(n)$ Zeit berechnet werden kann, dann kann man auch n Zahlen in $O(f(n) + n)$ Zeit sortieren.

Beweis

- gegeben: n Zahlen a_1, \dots, a_n
- konstruiere n Punkte $P = \{p_1, \dots, p_n\}$ mit $p_i = (a_i, a_i^2)$
- $\mathcal{CH}(P)$ enthält die Punkte sortiert nach a_i
- Sortierung lässt sich in $O(n)$ aus $\mathcal{CH}(P)$ ablesen

Beispiel

$a_1 = 2, a_2 = 1, a_3 = 3, a_4 = 0$



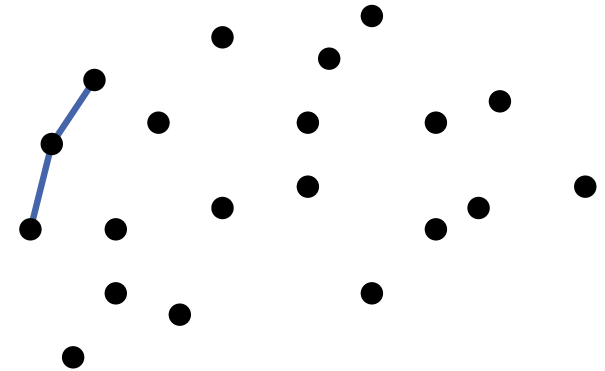
Untere Schranke

- vergleichsbasiertes Sortieren: $\Omega(n \log n)$
- Andrews Algorithmus ist also optimal

Gift Wrapping (Jarvis March)

Alternativer Ansatz

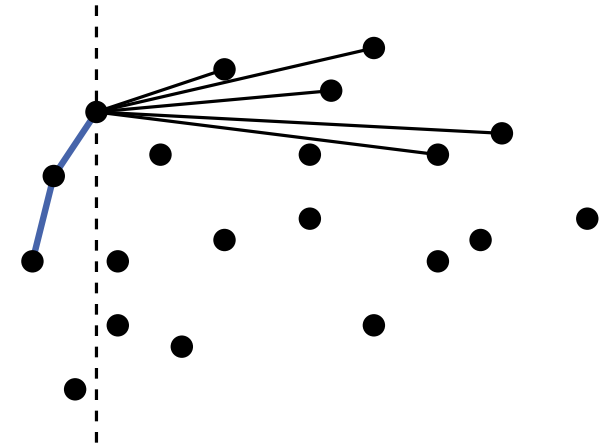
- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt



Gift Wrapping (Jarvis March)

Alternativer Ansatz

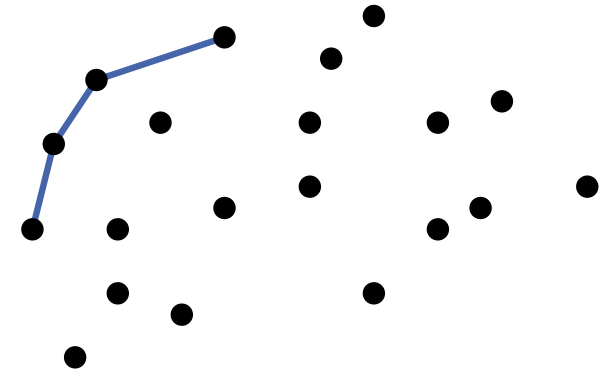
- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt
- wähle den Punkt mit dem kleinsten Winkel



Gift Wrapping (Jarvis March)

Alternativer Ansatz

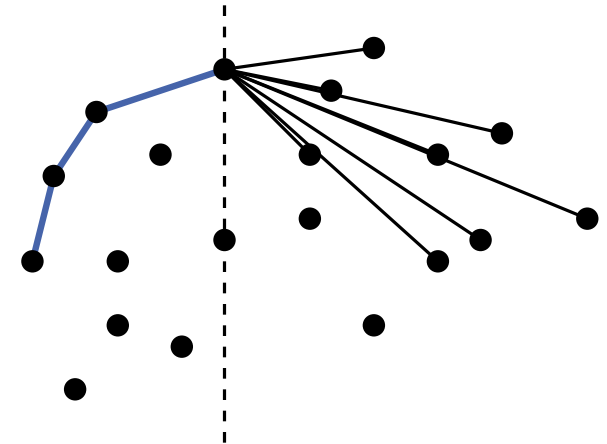
- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt
- wähle den Punkt mit dem kleinsten Winkel



Gift Wrapping (Jarvis March)

Alternativer Ansatz

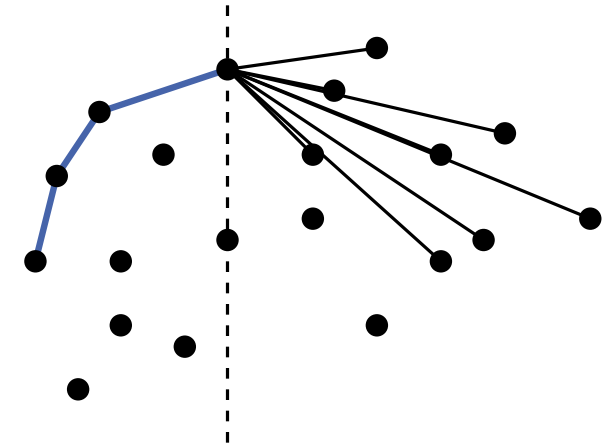
- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt
- wähle den Punkt mit dem kleinsten Winkel



Gift Wrapping (Jarvis March)

Alternativer Ansatz

- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt
- wähle den Punkt mit dem kleinsten Winkel



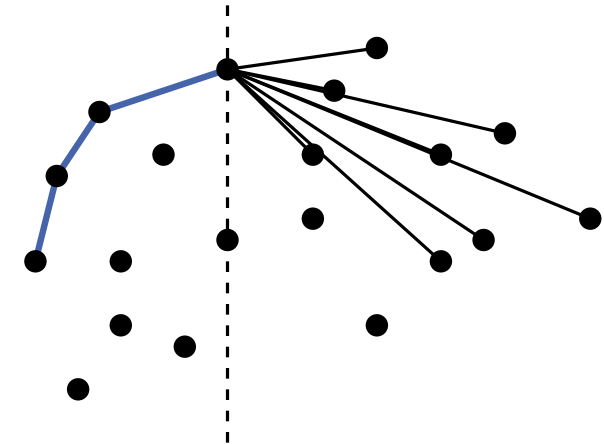
Laufzeit

- pro Schritt: Minimum suchen $\rightarrow O(n)$

Gift Wrapping (Jarvis March)

Alternativer Ansatz

- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt
- wähle den Punkt mit dem kleinsten Winkel



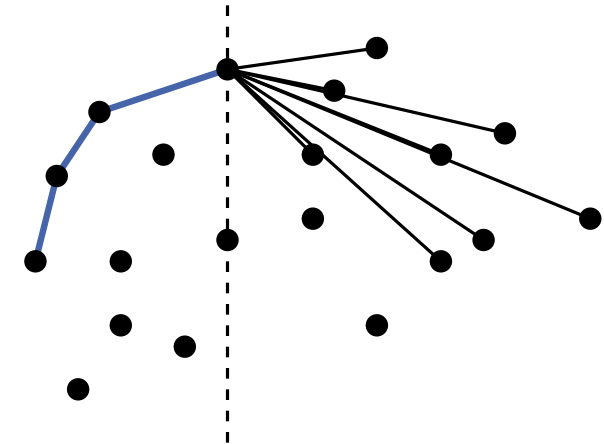
Laufzeit

- pro Schritt: Minimum suchen $\rightarrow O(n)$
- h Schritte, für $h = |\mathcal{CH}(P)|$

Gift Wrapping (Jarvis March)

Alternativer Ansatz

- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt
- wähle den Punkt mit dem kleinsten Winkel



Laufzeit

- pro Schritt: Minimum suchen $\rightarrow O(n)$
- h Schritte, für $h = |\mathcal{CH}(P)|$

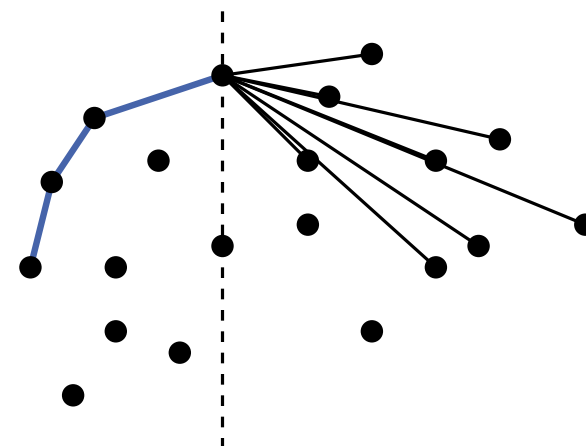
Theorem

Der Gift Wrapping Algorithmus berechnet Konvexe Hülle von n Punkten P in $O(hn)$ Zeit, wobei h die Anzahl Punkte von $\mathcal{CH}(P)$ ist.

Gift Wrapping (Jarvis March)

Alternativer Ansatz

- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt
- wähle den Punkt mit dem kleinsten Winkel



Laufzeit

- pro Schritt: Minimum suchen $\rightarrow O(n)$
- h Schritte, für $h = |\mathcal{CH}(P)|$

Theorem

Der Gift Wrapping Algorithmus berechnet Konvexe Hülle von n Punkten P in $O(hn)$ Zeit, wobei h die Anzahl Punkte von $\mathcal{CH}(P)$ ist.

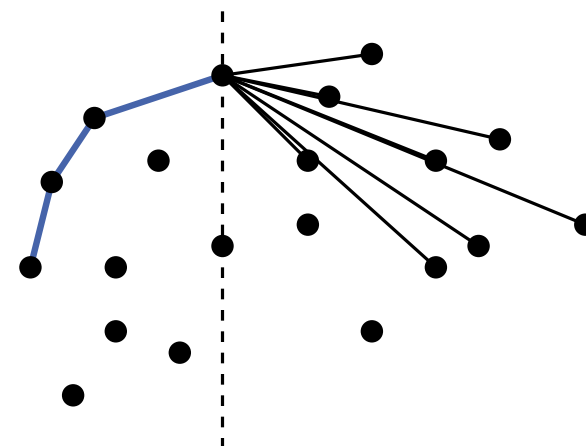
Anmerkungen

- ein solcher Algorithmus heißt **ausgabesensitiv**

Gift Wrapping (Jarvis March)

Alternativer Ansatz

- Annahme: wir haben schon einen Teil der oberen Einhüllenden
- Ziel: finde den nächsten Punkt
- wähle den Punkt mit dem kleinsten Winkel



Laufzeit

- pro Schritt: Minimum suchen $\rightarrow O(n)$
- h Schritte, für $h = |\mathcal{CH}(P)|$

Theorem

Der Gift Wrapping Algorithmus berechnet Konvexe Hülle von n Punkten P in $O(hn)$ Zeit, wobei h die Anzahl Punkte von $\mathcal{CH}(P)$ ist.

Anmerkungen

- ein solcher Algorithmus heißt **ausgabesensitiv**
- auf gewissen Instanzen (kleines h) kann er die untere Schranke schlagen

Zusammenfassung

Heute gesehen

- Algorithmus zur Berechnung der konvexen Hülle mit Laufzeit $O(n \log n)$

Zusammenfassung

Heute gesehen

- Algorithmus zur Berechnung der konvexen Hülle mit Laufzeit $O(n \log n)$
- untere Schranke von $\Omega(n \log n)$

Zusammenfassung

Heute gesehen

- Algorithmus zur Berechnung der konvexen Hülle mit Laufzeit $O(n \log n)$
- untere Schranke von $\Omega(n \log n)$
- Ausgabesensitiver Algorithmus mit Laufzeit $O(hn)$

Zusammenfassung

Heute gesehen

- Algorithmus zur Berechnung der konvexen Hülle mit Laufzeit $O(n \log n)$
- untere Schranke von $\Omega(n \log n)$
- Ausgabesensitiver Algorithmus mit Laufzeit $O(hn)$
- die Robustheit ist ein wichtiger Aspekt in der algorithmischen Geometrie

Zusammenfassung

Heute gesehen

- Algorithmus zur Berechnung der konvexen Hülle mit Laufzeit $O(n \log n)$
- untere Schranke von $\Omega(n \log n)$
- Ausgabesensitiver Algorithmus mit Laufzeit $O(hn)$
- die Robustheit ist ein wichtiger Aspekt in der algorithmischen Geometrie
- initial allgemeine Lage anzunehmen hilft beim Algorithmenentwurf

Zusammenfassung

Heute gesehen

- Algorithmus zur Berechnung der konvexen Hülle mit Laufzeit $O(n \log n)$
- untere Schranke von $\Omega(n \log n)$
- Ausgabesensitiver Algorithmus mit Laufzeit $O(hn)$
- die Robustheit ist ein wichtiger Aspekt in der algorithmischen Geometrie
- initial allgemeine Lage anzunehmen hilft beim Algorithmenentwurf

Was gibt es sonst noch?

- es geht sogar mit Laufzeit $O(n \log h)$

Zusammenfassung

Heute gesehen

- Algorithmus zur Berechnung der konvexen Hülle mit Laufzeit $O(n \log n)$
- untere Schranke von $\Omega(n \log n)$
- Ausgabesensitiver Algorithmus mit Laufzeit $O(hn)$
- die Robustheit ist ein wichtiger Aspekt in der algorithmischen Geometrie
- initial allgemeine Lage anzunehmen hilft beim Algorithmenentwurf

Was gibt es sonst noch?

- es geht sogar mit Laufzeit $O(n \log h)$
- höhere Dimensionen

Zusammenfassung

Heute gesehen

- Algorithmus zur Berechnung der konvexen Hülle mit Laufzeit $O(n \log n)$
- untere Schranke von $\Omega(n \log n)$
- Ausgabesensitiver Algorithmus mit Laufzeit $O(hn)$
- die Robustheit ist ein wichtiger Aspekt in der algorithmischen Geometrie
- initial allgemeine Lage anzunehmen hilft beim Algorithmenentwurf

Was gibt es sonst noch?

- es geht sogar mit Laufzeit $O(n \log h)$
- höhere Dimensionen
- konvexe Hülle eines einfachen Polygons geht in $O(n)$