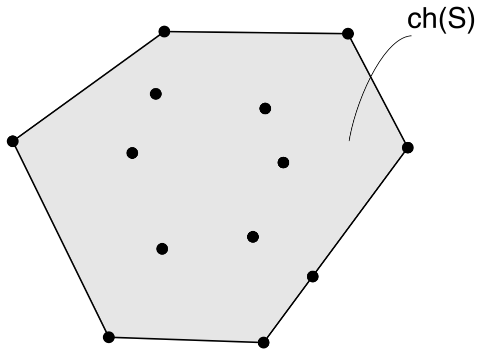
Durchschnitte

Elmar Langetepe
University of Bonn

Konstruktive Definition!

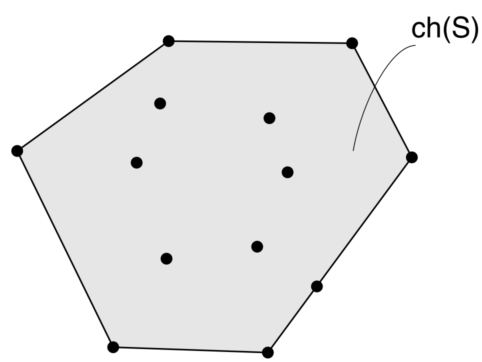
$$ch(S) = \bigcap_{\substack{K \supseteq A \\ K \text{ konvex}}} K$$



Konstruktive Definition!

Lemma 4.1 Sei S eine Menge von n Punkten in der Ebene, dann ist *ch*(S) ein konvexes Polygon mit Eckpunkten aus S.

$$\mathit{ch}(S) = \bigcap_{\substack{K \supseteq A \\ K \text{ konvex}}} K$$

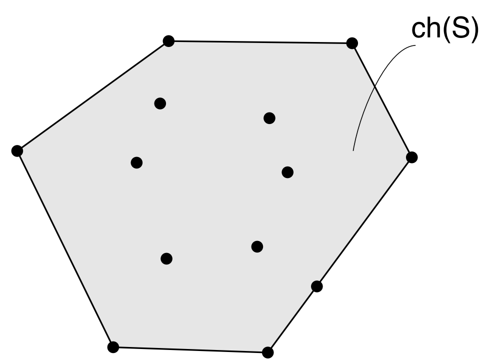


Konstruktive Definition!

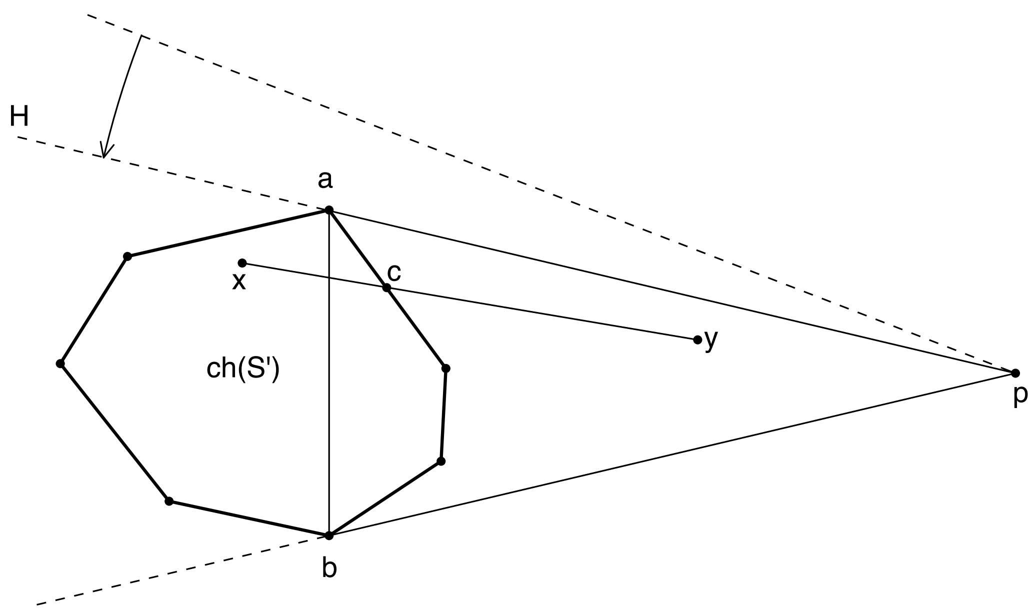
Lemma 4.1 Sei S eine Menge von n Punkten in der Ebene, dann ist *ch*(S) ein konvexes Polygon mit Eckpunkten aus S.

Beweis!

$$ch(S) = \bigcap_{\substack{K \supseteq A \\ K \text{ konvex}}} K$$



Induktiver Beweis! $p \notin \boldsymbol{ch}(S')$



Untere Schranke!

Untere Schranke!

- Diskrete Aufgabenstellung: Konvexes Polygon

Untere Schranke!

- Diskrete Aufgabenstellung: Konvexes Polygon
- Bestimme Rand mit Knoten sortiert in CW (CCW) Ordnung!
  Datenstruktur!

Untere Schranke!

- Diskrete Aufgabenstellung: Konvexes Polygon
- Bestimme Rand mit Knoten sortiert in CW (CCW) Ordnung! Datenstruktur!

Lemma 4.2: Die Konstruktion der konvexen Hülle von n Punkten in der Ebene hat Zeitkompleximtät $\Omega(n \log n)$.

Untere Schranke!

- Diskrete Aufgabenstellung: Konvexes Polygon
- Bestimme Rand mit Knoten sortiert in CW (CCW) Ordnung! Datenstruktur!

Lemma 4.2: Die Konstruktion der konvexen Hülle von n Punkten in der Ebene hat Zeitkompleximtät $\Omega(n \log n)$.

Beweis: Sortieren auf Konvexe Hülle reduzieren!

Untere Schranke!

- Diskrete Aufgabenstellung: Konvexes Polygon
- Bestimme Rand mit Knoten sortiert in CW (CCW) Ordnung! Datenstruktur!

Lemma 4.2: Die Konstruktion der konvexen Hülle von n Punkten in der Ebene hat Zeitkompleximtät $\Omega(n \log n)$.

Beweis: Sortieren auf Konvexe Hülle reduzieren!

Sortieren $\leq_n$ Konvexe Hülle

Strukturelle Eigenschaft!

Strukturelle Eigenschaft!

Lemma 4.3: Punktmenge S gegeben. Sei W ein einfacher, geschlossener Weg, der den Rand von *ch*(S) umschließt. Dann ist der Rand von *ch*(S) höchstens so lang wir W.

Strukturelle Eigenschaft!

Lemma 4.3: Punktmenge S gegeben. Sei W ein einfacher, geschlossener Weg, der den Rand von *ch*(S) umschließt. Dann ist der Rand von *ch*(S) höchstens so lang wir W.

Beweis!

Beweis Lemma 4.3

Beweis Lemma 4.3

- Winkelhalbierende h_i bei v_i nach außen

Beweis Lemma 4.3

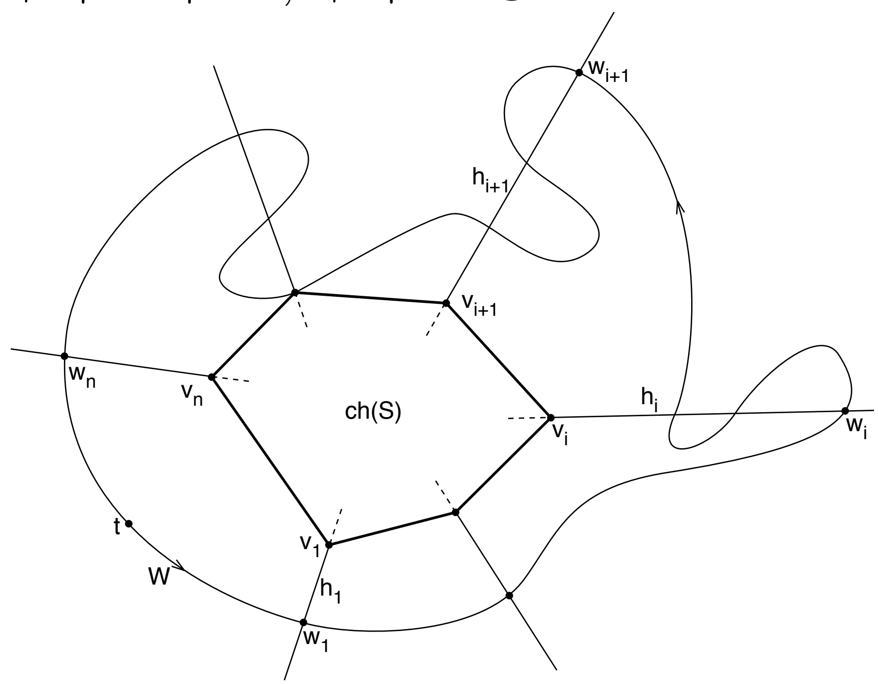
- Winkelhalbierende h_i bei v_i nach außen
- Wege von w_i nach w_{i+1}, vergleichen mit $v_i v_{i+1}$

Beweis Lemma 4.3

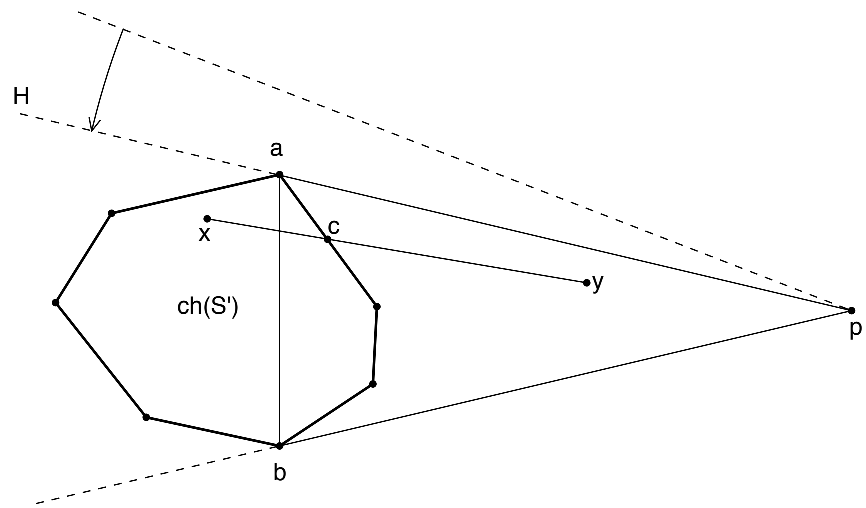
- Winkelhalbierende h_i bei v_i nach außen
- Wege von w_i nach w_{i+1}, vergleichen mit v_iv_{i+1}
- $|v_iv_{i+1}| \leq |w_iw_{i+1}| \leq |W_{i,i+1}|$ wegen Winkel!

Beweis Lemma 4.3

- Winkelhalbierende h_i bei v_i nach außen
- Wege von w_i nach w_{i+1}, vergleichen mit $v_i v_{i+1}$
- $|v_i v_{i+1}| \le |w_i w_{i+1}| \le |W_{i,i+1}|$ wegen Winkel!

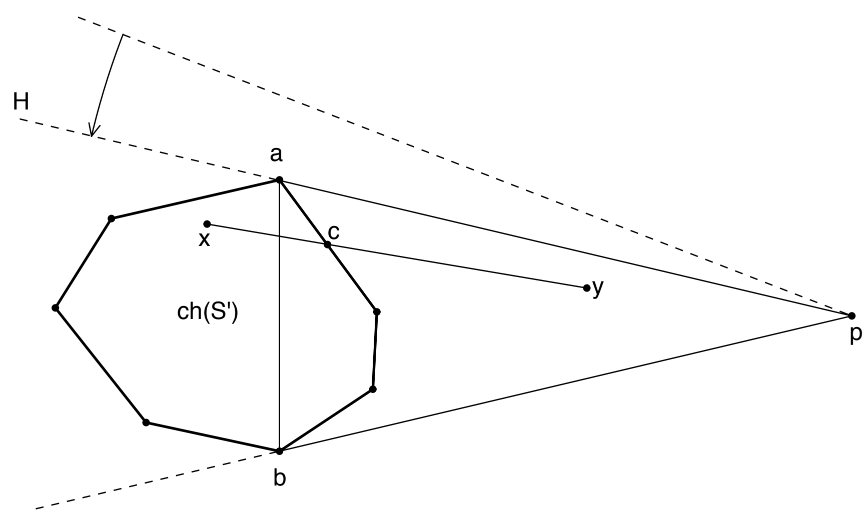


Inkrementelle Konstruktion!

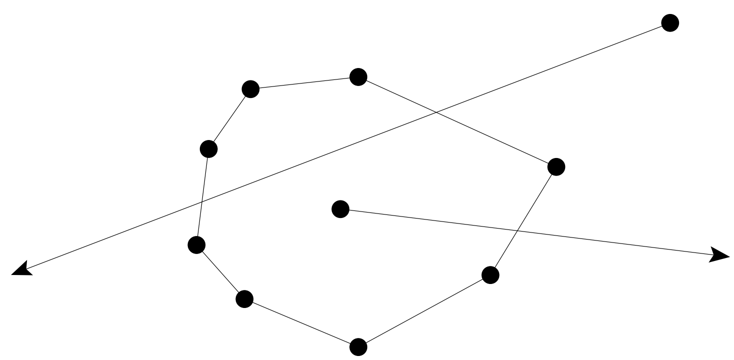


Inkrementelle Konstruktion!

- Beweis Lemma 4.1, Einfügen eines neuen Punktes
- Problem I: Innerhalb/außerhalb
- Problem II: Tangentenbestimmung, Vereinigung

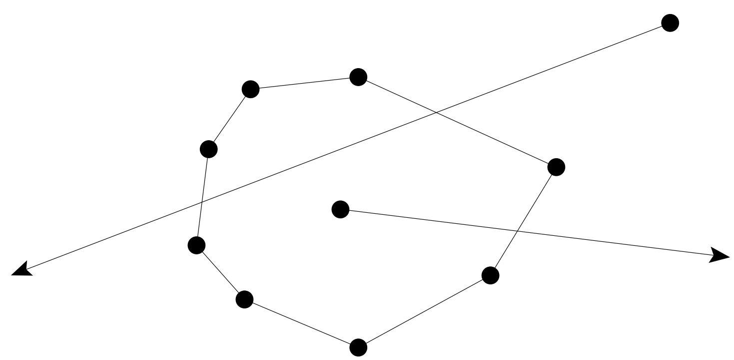


Problem I: Innerhalb/außerhalb



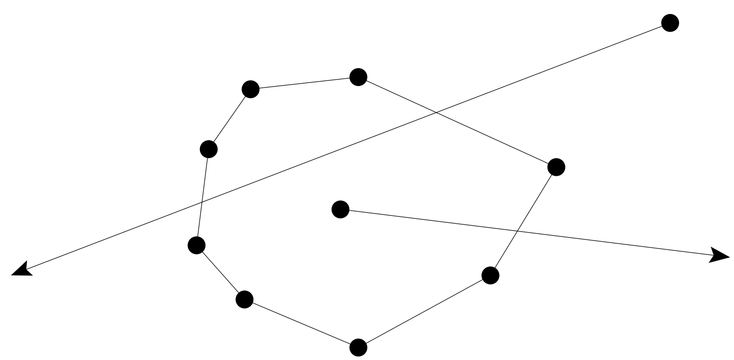
Problem I: Innerhalb/außerhalb

- Punkt p, bilde Strahl in eine Richtung, teste alle Segmente auf Schnitt



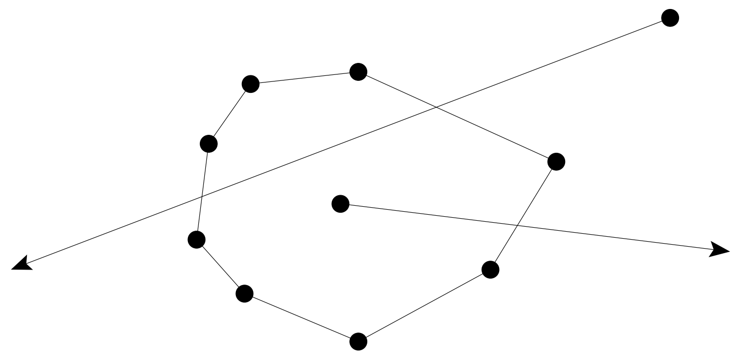
Problem I: Innerhalb/außerhalb

- Punkt p, bilde Strahl in eine Richtung, teste alle Segmente auf Schnitt
- Zwei Schnitte/kein Schnitt (außerhalb), ein Schnitt (innerhalb)

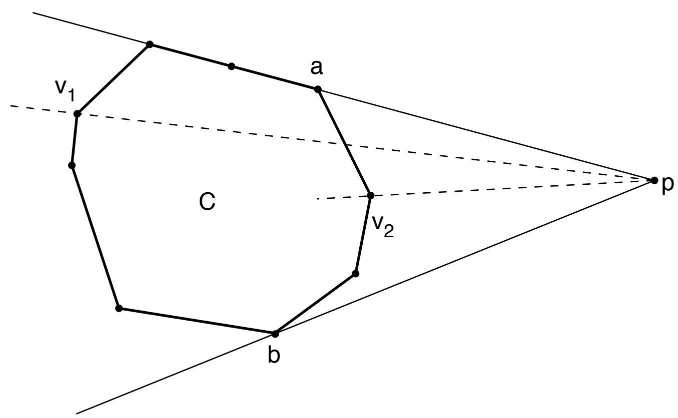


Problem I: Innerhalb/außerhalb

- Punkt p, bilde Strahl in eine Richtung, teste alle Segmente auf Schnitt
- Zwei Schnitte/kein Schnitt (außerhalb), ein Schnitt (innerhalb)
- In Zeit $O(n)$ lösbar!

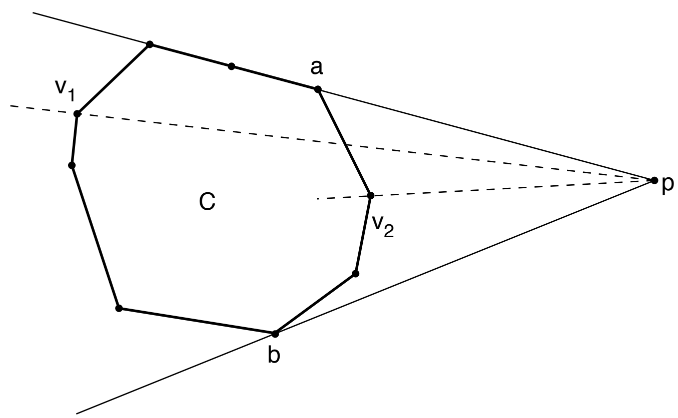


Problem II: Tangententest



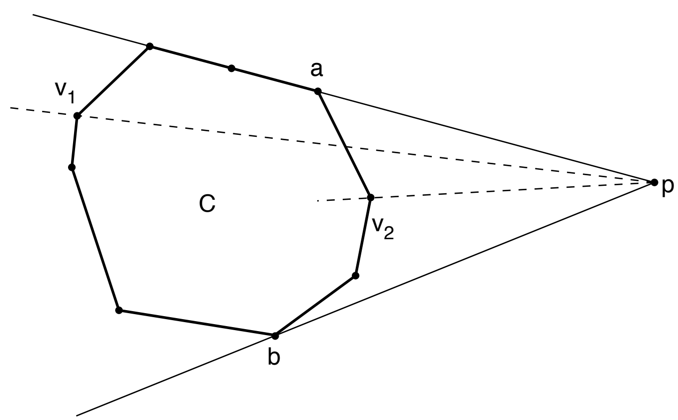
Problem II: Tangententest

- Von p aus sukzessive auf Tangente testen (Orientationtest)



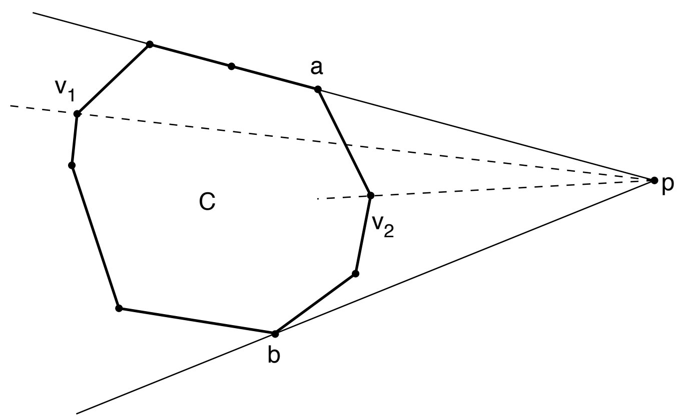
Problem II: Tangententest

- Von p aus sukzessive auf Tangente testen (Orientationtest)
- Bei beliebigem v in beide Richtungen starten



Problem II: Tangententest

- Von p aus sukzessive auf Tangente testen (Orientationtest)
- Bei beliebigem v in beide Richtungen starten
- In Zeit $O(n)$ lösbar!



Laufzeitverbesserung durch balancierte Bäume

Laufzeitverbesserung durch balancierte Bäume

Theorem 4.5 Die konvexe Hülle einer Menge von n Punkten in der Ebene zu konstruieren hat Zeitkomplexität $\Theta(n \log n)$ mit linearem Speicher.

Laufzeitverbesserung durch balancierte Bäume

Theorem 4.5 Die konvexe Hülle einer Menge von n Punkten in der Ebene zu konstruieren hat Zeitkomplexität $\Theta(n \log n)$ mit linearem Speicher.

Beweis: Verbesserung Problem I/II in $O(\log n)$ statt $O(n)$

Laufzeitverbesserung durch balancierte Bäume

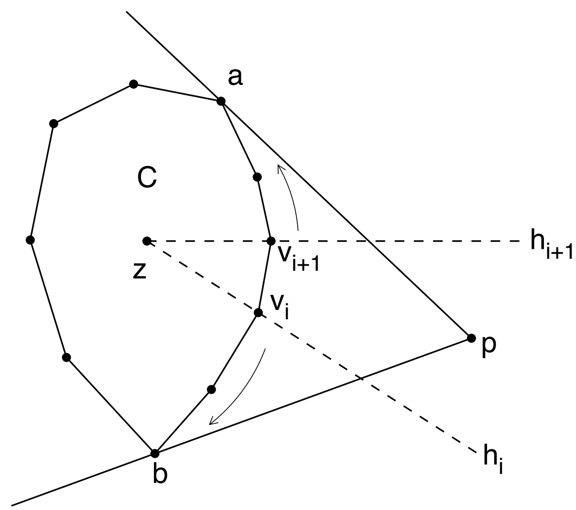
Theorem 4.5 Die konvexe Hülle einer Menge von n Punkten in der Ebene zu konstruieren hat Zeitkomplexität $\Theta(n \log n)$ mit linearem Speicher.

Beweis: Verbesserung Problem I/II in $O(\log n)$ statt $O(n)$

$O(n \log n)$ statt $O(n^2)$!

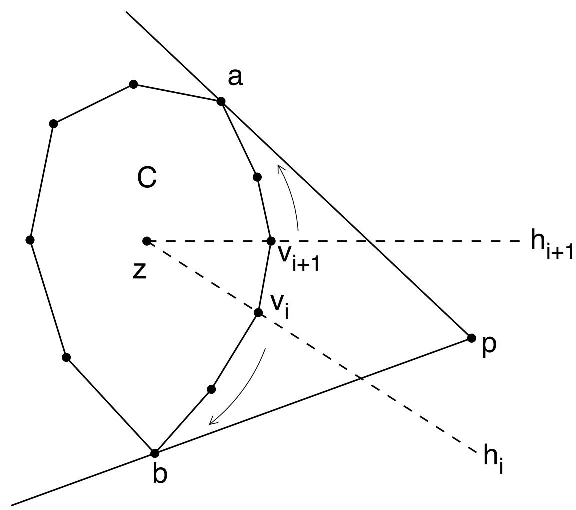
Verbesserung Problem I+II: Balancierte Bäume

- Sektoren in einem Baum abspeichern, nach Winkeln (Beispiel Tafel)



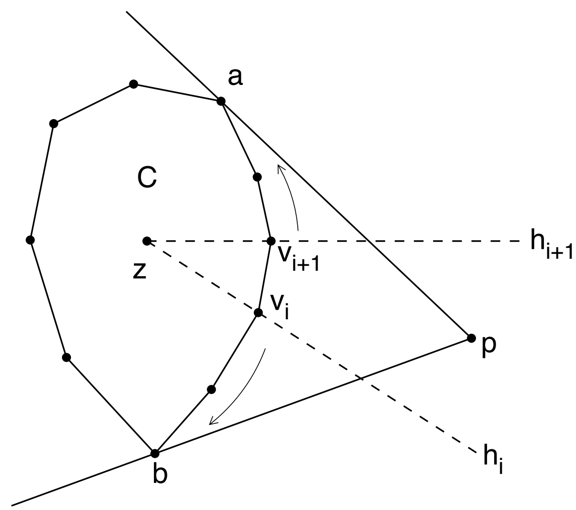
Verbesserung Problem I+II: Balancierte Bäume

- Sektoren in einem Baum abspeichern, nach Winkeln (Beispiel Tafel)
- Query Sektor: $O(\log n)$: Test innerhalb Dreieck!



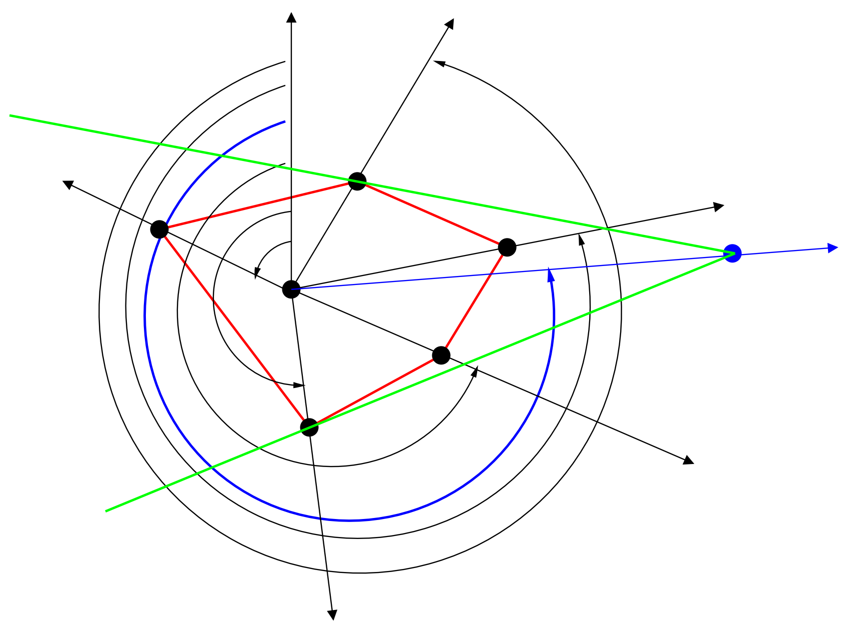
Verbesserung Problem I+II: Balancierte Bäume

- Sektoren in einem Baum abspeichern, nach Winkeln (Beispiel Tafel)
- Query Sektor: $O(\log n)$: Test innerhalb Dreieck!
- Tangenten: Entfernen aller Punkte/Winkel, Einfügen Punkt/Winkel $O(\log n)$



Verbesserung Problem I+II: Balancierte Bäume

- Sektoren in einem Baum abspeichern, nach Winkeln.
  Query Sektor, innerhalb: $O(\log n)$
- Tangenten:
  (Hülle) Punkte entfernen/Punkt einfügen $O(n)$ (amortisiert)
  (DS) Entfernen der Winkel, Einfügen Winkel, je $O(\log n)$



Ohne balancierte Bäume auskommen?

- Einfaches Verfahren, das *randomisiert* sehr gut funktioniert

Ohne balancierte Bäume auskommen?

- Einfaches Verfahren, das *randomisiert* sehr gut funktioniert
- Reihenfolge der Punkte wird zufällig gewählt

Ohne balancierte Bäume auskommen?

- Einfaches Verfahren, das *randomisiert* sehr gut funktioniert
- Reihenfolge der Punkte wird zufällig gewählt
- Randomisierte inkrementelle Konstruktion

Ohne balancierte Bäume auskommen?

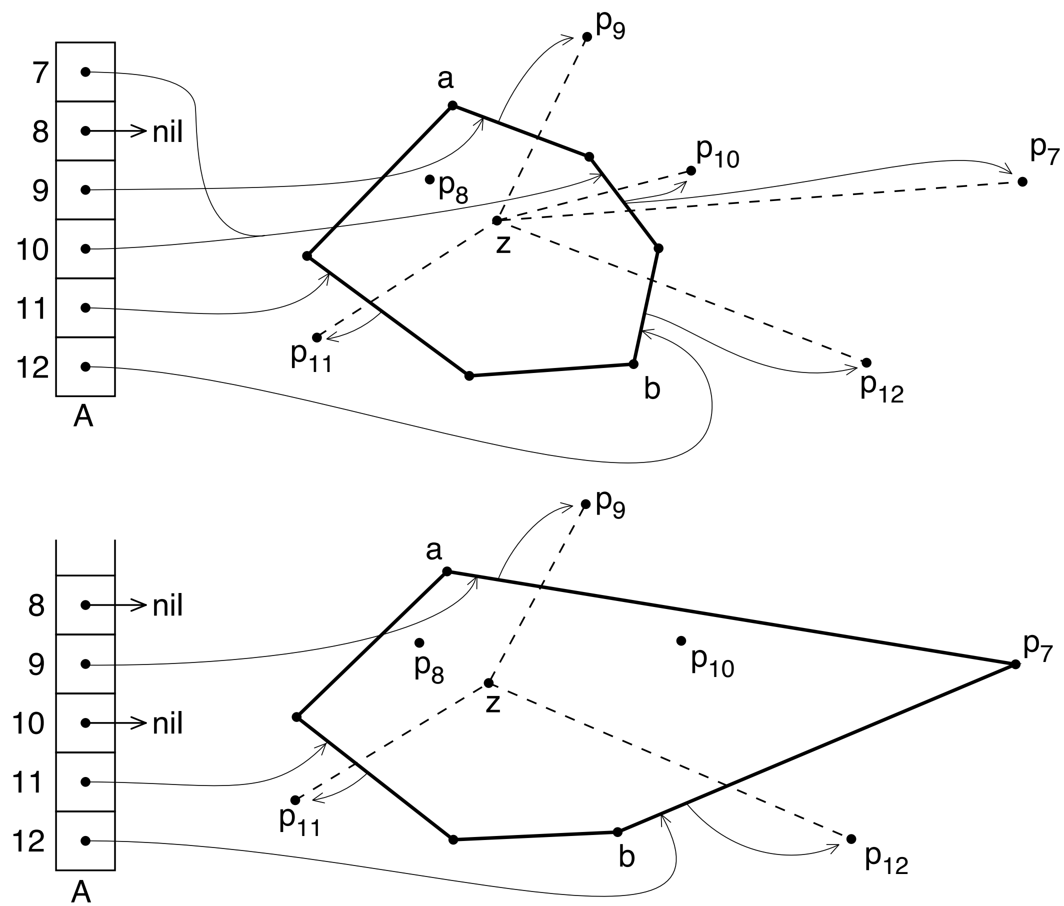
- Einfaches Verfahren, das *randomisiert* sehr gut funktioniert
- Reihenfolge der Punkte wird zufällig gewählt
- Randomisierte inkrementelle Konstruktion
- Idee: Man weiss stets, wo die Punkte liegen (Zeiger)

Ohne balancierte Bäume auskommen?

- Einfaches Verfahren, das *randomisiert* sehr gut funktioniert
- Reihenfolge der Punkte wird zufällig gewählt
- Randomisierte inkrementelle Konstruktion
- Idee: Man weiss stets, wo die Punkte liegen (Zeiger)

Theorem 4.6: Wird jede der $n!$ vielen Eingabereihenfolgen einer festen Punktmenge $S = \{p_1, p_2, \ldots, p_n\}$ mit gleicher Wahrscheinlichkeit gewählt, dann ergibt sich für die inkrementelle Konstruktion eine erwartete Anzahl von $O(n \log n)$ Operationen ($O(n \log n)$ Konfliktpaare).

Allgemein: Zeiger-Array aktualisieren (Konfliktpaar)



Aktualisierungsaufwand: Beispiel Tafel!

- Neuen Punkt einfügen. Kante erkennen, Tangentenpunkte a, b ermitteln, Kanten löschen, amortisiert insgesamt $O(n)$

Aktualisierungsaufwand: Beispiel Tafel!

- Neuen Punkt einfügen. Kante erkennen, Tangentenpunkte a, b ermitteln, Kanten löschen, amortisiert insgesamt $O(n)$
- Array aktualisieren: Einige Punkte bekommen neue Kante oder liegen nun innerhalb!

Aktualisierungsaufwand: Beispiel Tafel!

- Neuen Punkt einfügen. Kante erkennen, Tangentenpunkte a, b ermitteln, Kanten löschen, amortisiert insgesamt $O(n)$
- Array aktualisieren: Einige Punkte bekommen neue Kante oder liegen nun innerhalb!
- Konfliktpaare: p_i einfügen, p_j noch nicht bearbeitet:

Aktualisierungsaufwand: Beispiel Tafel!

- Neuen Punkt einfügen. Kante erkennen, Tangentenpunkte a, b ermitteln, Kanten löschen, amortisiert insgesamt $O(n)$
- Array aktualisieren: Einige Punkte bekommen neue Kante oder liegen nun innerhalb!
- Konfliktpaare: p_i einfügen, p_j noch nicht bearbeitet:

(p_i, p_j) in Konflikt $\iff$ zp_j schneidet eine Kante von *ch*(S_{i-1}), die beim Einfügen von p_i entfernt wird

Aktualisierungsaufwand: Beispiel Tafel!

- Neuen Punkt einfügen. Kante erkennen, Tangentenpunkte a, b ermitteln, Kanten löschen, amortisiert insgesamt $O(n)$
- Array aktualisieren: Einige Punkte bekommen neue Kante oder liegen nun innerhalb!
- Konfliktpaare: p_i einfügen, p_j noch nicht bearbeitet:

(p_i, p_j) in Konflikt $\iff$ zp_j schneidet eine Kante von *ch*(S_{i-1}), die beim Einfügen von p_i entfernt wird

- Für alle diese neuer Zeiger oder innerhalb

Aktualisierungsaufwand: Beispiel Tafel!

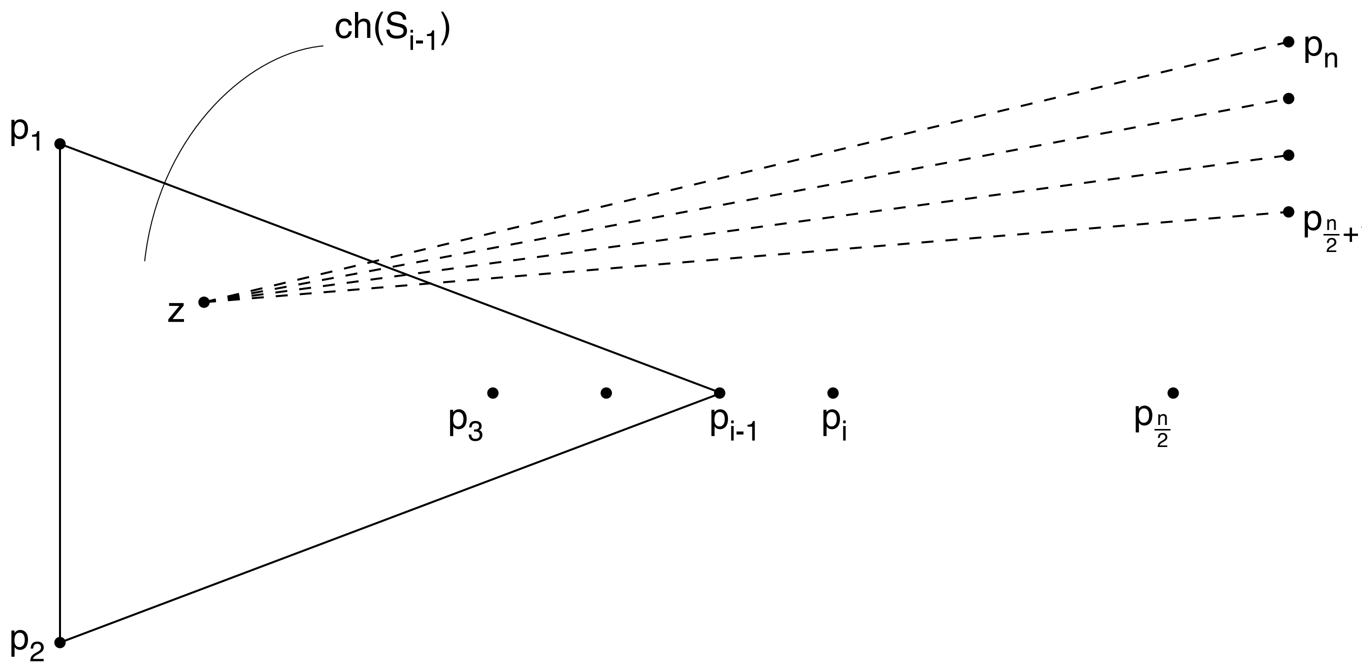
- Neuen Punkt einfügen. Kante erkennen, Tangentenpunkte a, b ermitteln, Kanten löschen, amortisiert insgesamt $O(n)$
- Array aktualisieren: Einige Punkte bekommen neue Kante oder liegen nun innerhalb!
- Konfliktpaare: p_i einfügen, p_j noch nicht bearbeitet:

(p_i, p_j) in Konflikt $\iff$ zp_j schneidet eine Kante von *ch*(S_{i-1}), die beim Einfügen von p_i entfernt wird

- Für alle diese neuer Zeiger oder innerhalb
- Laufzeit: Anzahl der Konfliktpaare, zählen

Worst-Case Anzahl Konflikte

Schlechte Reihenfolge!



Zufällige Eingabereihenfolge!

Ergebnis

Ergebnis

Theorem 4.6: Wird jede der $n!$ vielen Eingabereihenfolgen einer festen Punktmenge $S = \{p_1, p_2, \ldots, p_n\}$ mit gleicher Wahrscheinlichkeit gewählt, dann ergibt sich für die inkrementelle Konstruktion eine erwartete Anzahl von $O(n \log n)$ Operationen ($O(n \log n)$ Konfliktpaare).