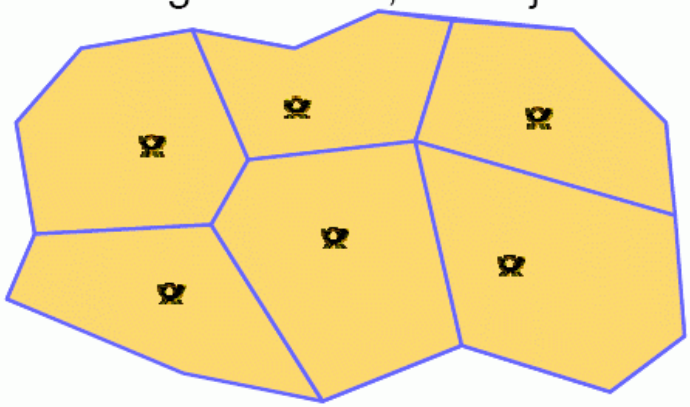
Grundlagen zur Delaunay-Triangulierung und zur konvexen Hülle

zum Begriff des *Voronoi-Diagramms*
(vgl. auch Vorlesung "Algorithmische Geometrie"):

Postproblem:

Seien n Poststellen in einer Stadt gegeben.

Welches sind die Einzugsbereiche, wenn jeder zur nächst-gelegenen Poststelle geht?

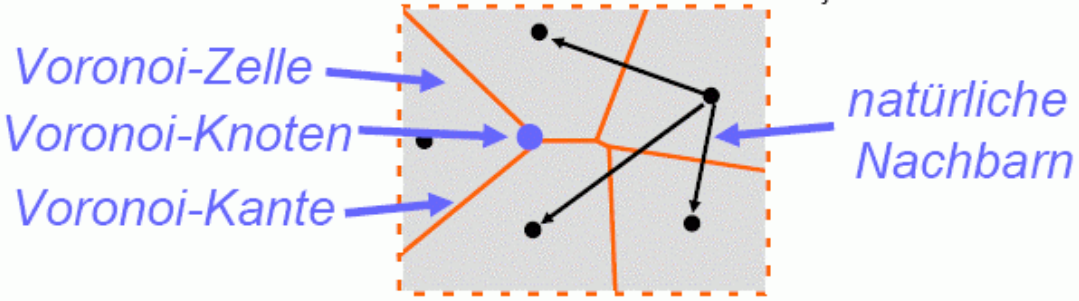


Definition Voronoi-Diagramm:

geg: eine Menge von Punkten $Q=\{\mathbf{q}_1,\ldots,\mathbf{q}_k\}$ aus $\mathbf{R}^n$ und eine Distanzfunktion dist($\mathbf{q}$,$\mathbf{r}$)

Def.: Das Voronoi-Diagramm Vor(Q) enthält für jeden Punkt $\mathbf{q}_i$ eine Zelle $V(\mathbf{q}_i)$ mit

$$V(\mathbf{q}_i)=\{\mathbf{q} \mid dist(\mathbf{q}, \mathbf{q}_i) < dist(\mathbf{q}, \mathbf{q}_j)\ \forall j \neq i\}$$

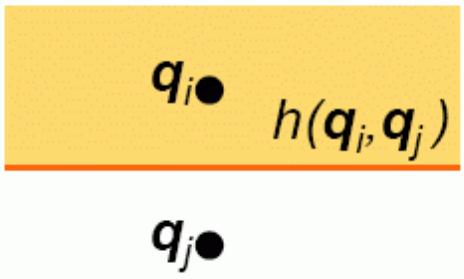


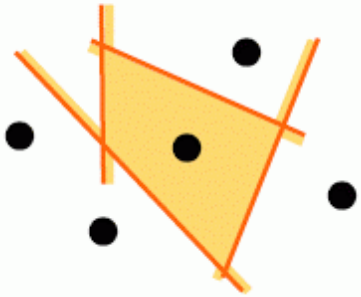
Voronoi-Zellen:

Sei $h(\mathbf{q}_i,\mathbf{q}_j)$ der Halbraum definiert durch die Mittelsenkrechte zwischen $\mathbf{q}_i$ und $\mathbf{q}_j$, indem $\mathbf{q}_i$ liegt, dann gilt:

$$V(\mathbf{q}_i) = \cap_{j\neq i} h(\mathbf{q}_i,\mathbf{q}_j)$$

➔ die Voronoi-Zellen sind konvex





Erzeugung des Voronoi-Diagramms – **Algorithmen:**

Sweep-Line-Algorithmus

- sortiere Ereignisse in eine Hauptrichtung
- bearbeite Ereignis und erzeuge neue Kanten

Einfügealgorithmus

- ordne Punkte in zufälliger Reihenfolge an
- füge Punkt für Punkt in Voronoi-Diagramm ein

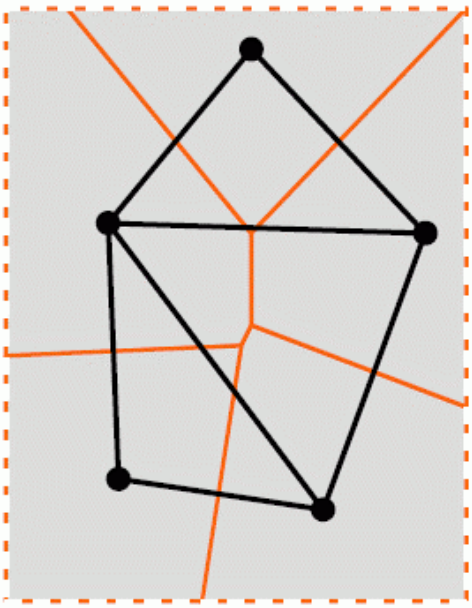
Voronoi in höheren Dimensionen:

- Größe: $\Theta(k^{\lceil n/2 \rceil})$
- Laufzeit: $O(k\cdot \log k + k^{\lceil n/2 \rceil})$

(siehe Vorlesung "Algorithmische Geometrie")

Definition Delaunay Graph:

Der Duale Graph des Voronoi-Diagramms Vor(Q) wird Delaunay-Graph Del(Q) genannt. Die Punkte Q sind die Knoten. Zwei Knoten $\mathbf{q}_i$ und $\mathbf{q}_j$ sind genau dann verbunden, wenn die Voronoi-Zellen $V(\mathbf{q}_i)$ und $V(\mathbf{q}_j)$ eine gemeinsame Kante haben.



Eigenschaften des Delaunay-Graphen:

- die Eckpunkte jeder Delaunay-Zelle liegen auf einem Kreis
- insbes. sind die Zellen konvex
- Wenn keine 4 Eckpunkte auf einem Kreis liegen ("allgemeine Position"), ist der Delaunay-Graph eine Triangulierung.

Jede Triangulierung des Delaunay-Graphen wird als *Delaunay-Triangulierung* (*DT*) bezeichnet.

Algorithmen zur Konstruktion einer Delaunay-Triangulierung zu einer Punktmenge:
- Flipping-Algorithmus
- Sweep-Line-Verfahren
- Einfügealgorithmus (inkrementell)
- divide and conquer
- und andere

(siehe unten)

Spezielle Rolle der DT unter allen Triangulierungen einer Punktmenge

Def. einer Triangulierung einer Punktmenge *S*:

Gegeben: eine endliche Menge *S* von Punkten in der Ebene

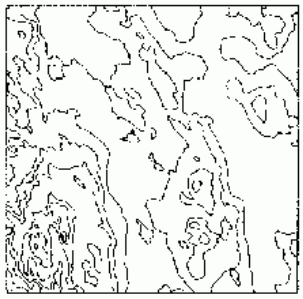
Gesucht: eine Zerlegung der konvexen Hülle von *S* in Dreiecke, so dass die Elemente von *S* die Eckpunkte der Dreiecke sind.

alternative Def.: Triangulierung von *S* = *maximale* planare Unterteilung mit Knotenmenge *S* (Maximalität: jede zusätzliche Kante zwischen 2 Punkten von *S* würde eine existierende Kante schneiden).

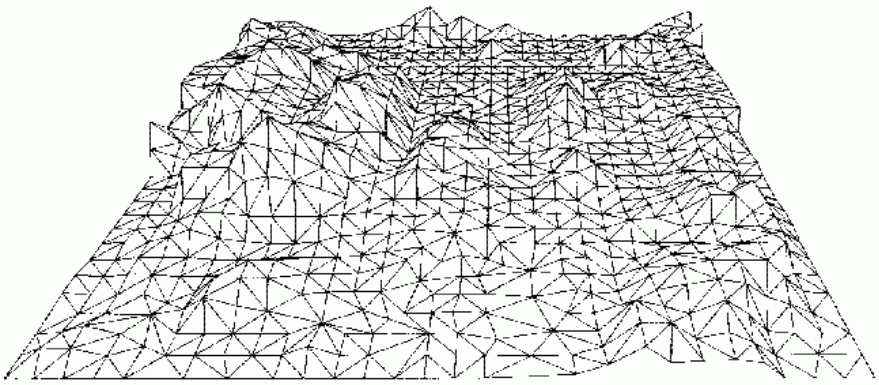
Anwendungen:
Geodäsie, Finite-Elemente-Methode (FEM).

Beispiel:

- *Terrain:*
  2-dimensionale Fläche im 3-dimensionalen Raum, so daß jede vertikale Linie die Fläche in genau einem Punkt schneidet
- Terrain ist der Graph einer Funktion f: $\mathcal{A} \subset \mathbb{R}^2 \rightarrow \mathbb{R}$, die jedem Punkt p in dem Definitionsbereich $\mathcal{A}$ des Terrains eine Höhe f(p) zuordnet
- Visualisierung eines Terrains durch Konturlinien:

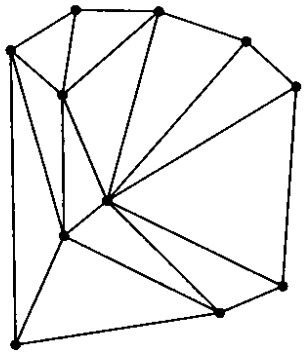


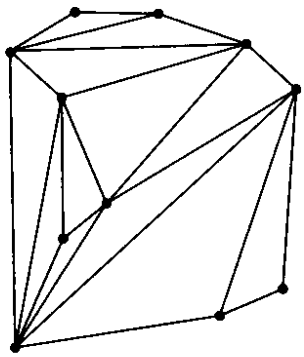
- Visualisierung eines Terrains durch perspektivische Aufsicht:



Triangulierungen von Punktmengen i. allg. nicht eindeutig

Bsp.:





linke Variante wirkt "ausgeglichener" als die rechte

verschiedene Optimalitätskriterien für Triangulierungen von *S*:

- minimale Kantenlängensumme
  Nachteil: verhindert nicht die Erzeugung langer, dünner Dreiecke (ungünstig bei FEM u. Visualisierung)
- *Max-Min-Winkelkriterium*: der kleinste vorkommende Dreieckswinkel wird maximiert
- *Min-Max-Winkelkriterium*: der größte vorkommende Dreieckswinkel wird minimiert
- *Max-Min-Radiuskriterium*: der kleinste Radius der in die Dreiecke einbeschriebenen Kreise wird maximiert
- *Min-Max-Radiuskriterium*: der größte Radius der in die Dreiecke einbeschriebenen Kreise wird minimiert
- *Max-Min-Flächenkriterium*: der kleinste Flächeninhalt der Dreiecke wird maximiert
- *Max-Min-Höhenkriterium*: die kleinste Höhe der Dreiecke wird maximiert

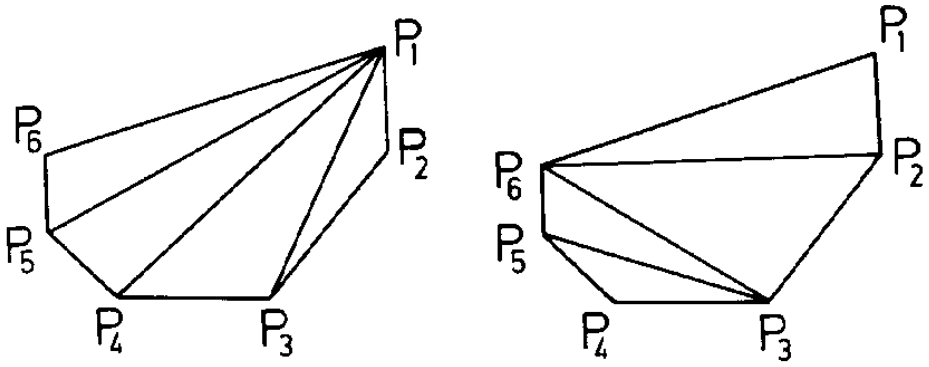
Alle Kriterien können (in speziellen Fällen) unterschiedliche Triangulierungen liefern!

Eine Triangulierung *T* heißt *lokal optimal* bzgl. eines Kriteriums *K*, wenn jedes Viereck, definiert durch je 2 entlang einer gemeinsamen Kante aneinandergrenzende Dreiecke von *T*, bzgl. *K* optimal trianguliert ist.

Eine Triangulierung *T* der Punktmenge *S* heißt *global optimal* bzgl. *K*, wenn jede andere Triangulierung von *S* ungünstiger als *T* bzgl. *K* ist.

Eine Punktmenge kann mehrere lokal optimale Triangulierungen haben:

Zwei bzgl. des Min-Max-Winkelkriteriums lokal optimale Triangulierungen von 6 Punkten:



Das *Max-Min-Winkelkriterium* ist das einzige bekannte Kriterium, für das lokale Optima stets auch globale Optima sind!

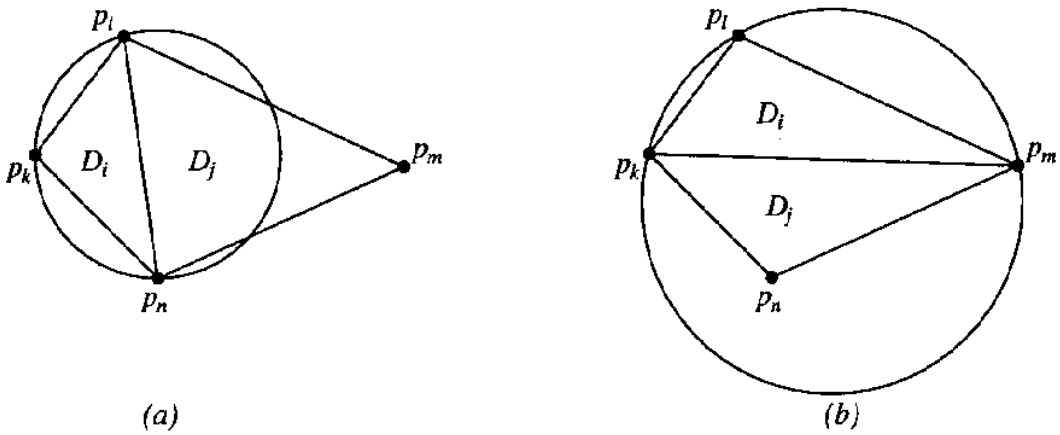
$\Rightarrow$ Auffinden des globalen Optimums dann durch lokale Operationen unabh. von der Vorgehensweise möglich.

(Aber: Häufig führen das Max-Min- und das Min-Max-Winkelkriterium zur gleichen Triangulierung – vergleichende Tests mit Zufalls-Punktmengen: Abweichungen nur in ca. 10 % der Fälle.)

Satz:
Die mit dem Max-Min-Winkelkriterium konstruierte Triangulierung ist die Delaunay-Triangulierung.

Lokales Umkreiskriterium für eine Triangulierung:

Für je 4 Punkte, die zu 2 benachbarten Dreiecken der Triangulierung gehören, enthält der Umkreis des einen Dreiecks *nicht* den vierten Punkt.



lokales Umkreiskriterium: links: erfüllt, rechts: nicht erfüllt

globales Umkreiskriterium: für sämtliche Dreiecke der Triangulierung enthält ihr Umkreis keine Punkte aus *S*.

lokales Umkreiskrit. überall erfüllt $\Rightarrow$ globales Umkreiskrit. gilt.

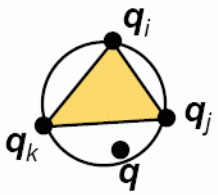
- Bei Nichterfüllung des lokalen Umkreiskrit. wird retrianguliert durch Diagonalentausch
- durch solche lokalen Änderungen ist das globale Optimum erreichbar
- *Optimum identisch zu Delaunay-Triangulation* !

Umkreis-Test:

Im Umkreis eines Delaunay-Dreiecks liegt kein 4. Punkt

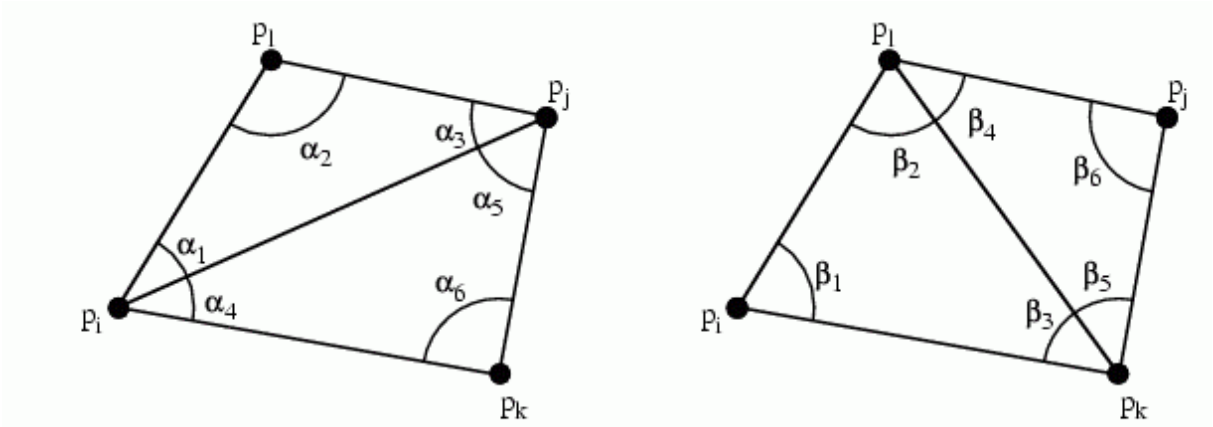
Geg.: Dreieck $(\mathbf{q}_i,\mathbf{q}_j,\mathbf{q}_k)$ und $\mathbf{q}$

Frage: Liegt $\mathbf{q}$ im Umkreis?



$$\Longleftrightarrow \det \begin{pmatrix} q_{i,x} & q_{i,y} & \|\boldsymbol{q}_i\|^2 & 1 \\ q_{j,x} & q_{j,y} & \|\boldsymbol{q}_j\|^2 & 1 \\ q_{k,x} & q_{k,y} & \|\boldsymbol{q}_k\|^2 & 1 \\ q_x & q_y & \|\boldsymbol{q}\|^2 & 1 \end{pmatrix} > 0$$

(Bartz 2005)



- Kante $e = \overline{p_i p_j}$ heißt eine *unzulässige Kante*, falls

  $$\min_{1 \le i \le 6} \alpha_i < \min_{1 \le i \le 6} \beta_i$$

- Kante unzulässig, falls lokal Vergrößerung des kleinsten Winkels durch Flippen dieser Kante

(Hinrichs 2002)

man nennt dies auch einen "Lawson-Flip".

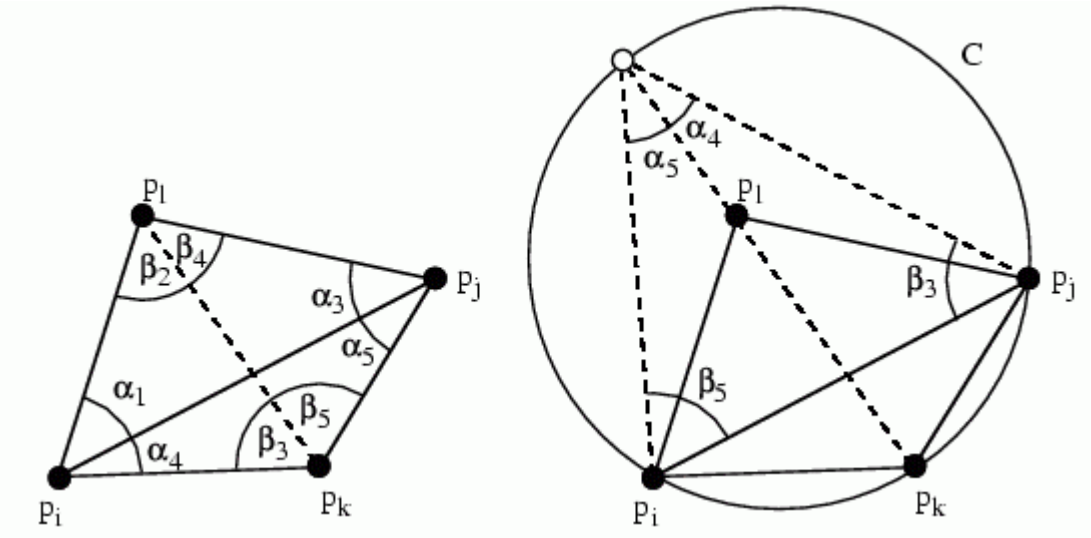
Die DT ist dadurch charakterisiert, dass alle Kanten zulässig sind.

- Es ist nicht notwendig, $\alpha_1, \ldots, \alpha_6, \beta_1, \ldots, \beta_6$ zu berechnen, um zu überprüfen, ob eine Kante zulässig ist $\Rightarrow$
  Kriterium zur Überprüfung:

lokales Umkreiskriterium.

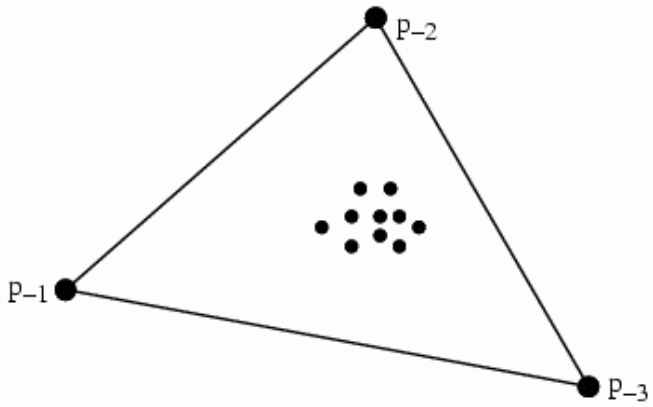
Die Kante $\overline{p_i p_j}$ sei inzident zu den Dreiecken $p_i p_j p_k$ und $p_i p_j p_l$, und C sei der Kreis durch p_i, p_j und p_k. Die Kante $\overline{p_i p_j}$ ist unzulässig genau dann, wenn der Punkt p_l im Innern von C liegt. Bilden die Punkte p_i, p_j, p_k und p_l ein konvexes Viereck und liegen sie nicht auf einem gemeinsamen Kreis, so ist entweder $\overline{p_i p_j}$ oder $\overline{p_k p_l}$ eine unzulässige Kante.

(vgl. Hinrichs 2002)



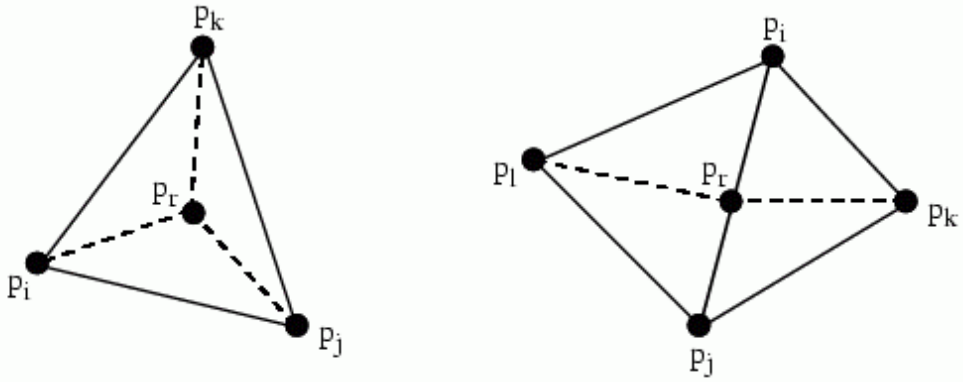
Randomisierter, inkrementeller Algorithmus zur Berechnung der DT (nach Hinrichs 2002):

- Beginne mit einem großen, die Punktmenge S enthaltenden Dreieck $p_{-1}p_{-2}p_{-3}$ und berechne die Delaunay-Triangulierung von $\Omega \cup S$, wobei $\Omega := \{p_{-1},p_{-2},p_{-3}\}$:



- Nach Berechnung der Delaunay-Triangulierung von $\Omega \cup S$ werden p_{-1}, p_{-2}, p_{-3} mit allen inzidenten Kanten entfernt → p_{-1}, p_{-2}, p_{-3} müssen weit genug entfernt von den Punkten aus S gewählt werden, damit sie keine Dreiecke in der Delaunay-Triangulierung von S zerstören. Insbesondere dürfen sie nicht innerhalb eines durch drei Punkte aus S definierten Kreises liegen.
- Algorithmus fügt Punkte in einer Zufallsreihenfolge ein und erhält eine Delaunay-Triangulierung der gegenwärtigen Punktmenge aufrecht.

- Hinzufügen des Punktes p_r:
  - Bestimme zunächst das Dreieck der gegenwärtigen Triangulierung, das den Punkt p_r enthält, und füge Kanten von p_r zu den Eckpunkten dieses Dreiecks ein.
  - Liegt p_r auf einer Kante e der Triangulierung, so füge Kanten von p_r zu den der Kante e gegenüberliegenden Eckpunkten der beiden in e aneinandergrenzenden Dreiecke ein:



→ Resultierende Triangulierung ist aber nicht notwendigerweise eine Delaunay-Triangulierung, da durch Hinzufügen von p_r einige der existierenden Kanten unzulässig werden können.

- Behebung des Problems durch Aufruf einer Prozedur *LegalizeEdge* für jede potentiell unzulässige Kante: *LegalizeEdge* ersetzt mit Hilfe von Kantenflips unzulässige Kanten durch zulässige.

Top-Down-Beschreibung des Algorithmus

(hier keine Behandlung von Spezialfällen wie den Rand-Kanten; s. dazu Hinrichs 2002, http://web.archive.org/web/20030620220921/http://wwwmath.uni-muenster.de/u/chr/Geometrie/) :

DelaunayTriangulation(S)

Eingabe: Eine Menge S von n Punkten in der Ebene.

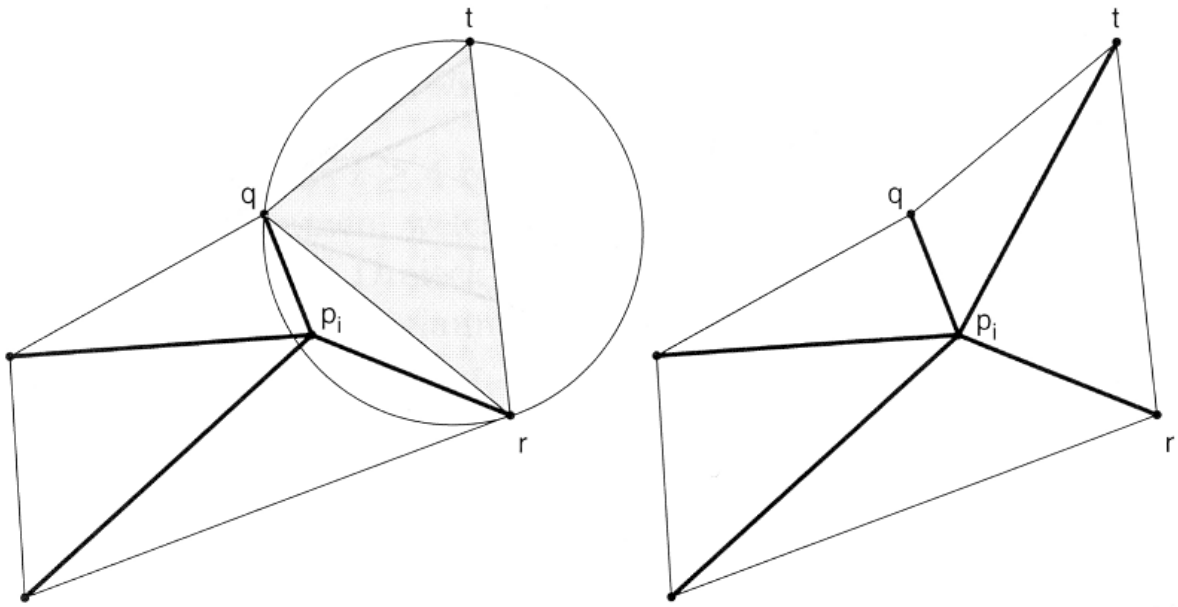
Ausgabe: Eine Delaunay-Triangulierung von S.

1. Seien p_{-1}, p_{-2} und p_{-3} drei passende Punkte, so daß S im Dreieck $p_{-1}p_{-2}p_{-3}$ enthalten ist.
2. Initialisiere $\mathcal{T}$ als die Triangulierung, die aus dem einzelnen Dreieck $p_{-1}p_{-2}p_{-3}$ besteht.
3. Bestimme eine Zufallspermutation $p_1, p_2, \ldots, p_n$ der Elemente von S.
4. for r := 1 to n do
5. Finde das Dreieck $p_ip_jp_k \in \mathcal{T}$, das p_r enthält.
6. if p_r liegt im Inneren des Dreiecks $p_ip_jp_k$ then
7. Füge Kanten von p_r zu den Eckpunkten des Dreiecks $p_ip_jp_k$ ein $\Rightarrow$ Aufteilung von $p_ip_jp_k$ in drei Dreiecke.
8. *LegalizeEdge*(p_r, $\overline{p_ip_j}$, $\mathcal{T}$)
9. *LegalizeEdge*(p_r, $\overline{p_jp_k}$, $\mathcal{T}$)
10. *LegalizeEdge*(p_r, $\overline{p_kp_i}$, $\mathcal{T}$)

else {p_r liegt auf einer Kante von $p_ip_jp_k$, z.B. auf $\overline{p_ip_j}$}

11. Füge Kanten von p_r zu p_k und dem dritten Eckpunkt p_l des anderen an $\overline{p_ip_j}$ angrenzenden Dreiecks ein, wodurch die beiden an $\overline{p_ip_j}$ angrenzenden Dreiecke in vier Dreiecke aufgeteilt werden.
12. *LegalizeEdge*(p_r, $\overline{p_ip_l}$, $\mathcal{T}$)
13. *LegalizeEdge*(p_r, $\overline{p_lp_j}$, $\mathcal{T}$)
14. *LegalizeEdge*(p_r, $\overline{p_jp_k}$, $\mathcal{T}$)
15. *LegalizeEdge*(p_r, $\overline{p_kp_i}$, $\mathcal{T}$)
16. Entferne p_{-1}, p_{-2}, p_{-3} mit allen ihren inzidenten Kanten aus $\mathcal{T}$.
17. return $\mathcal{T}$

- Welche Kanten können durch das Einfügen von Punkt p_r unzulässig werden?
- Beobachtung: Zuvor zulässige Kante $\overline{p_i p_j}$ kann durch Einfügen von p_r unzulässig werden , falls eines der beiden zu $\overline{p_i p_j}$ inzidenten Dreiecke sich ändert.
  $\Rightarrow$ Nur die Kanten neuer Dreiecke müssen überprüft werden.
- Überprüfung durch Prozedur *LegalizeEdge*, die Kanten testet und möglicherweise Kantenflips durchführt. Nach einem Kantenflip können andere Kanten unzulässig werden $\rightarrow$ *LegalizeEdge* ruft sich rekursiv für solch potentiell unzulässigen Kanten auf.



Solange der Umkreis eines angrenzenden Dreiecks den neuen Punkt enthält, werden *edge flips* ausgeführt

LegalizeEdge(p_r, $\overline{p_ip_j}$, $\mathcal{T}$)

p_r ist der eingefügte Punkt und $\overline{p_ip_j}$ ist die Kante von $\mathcal{T}$, die eventuell geflippt werden muß.

1. if $\overline{p_ip_j}$ ist unzulässig then
2. Sei $p_ip_jp_k$ das zu $p_rp_ip_j$ entlang $\overline{p_ip_j}$ benachbarte Dreieck.
3. Flippe $\overline{p_ip_j}$, d.h. ersetze $\overline{p_ip_j}$ durch $\overline{p_rp_k}$.
4. *LegalizeEdge*(p_r, $\overline{p_ip_k}$, $\mathcal{T}$)
5. *LegalizeEdge*(p_r, $\overline{p_kp_j}$, $\mathcal{T}$)

- Algorithmus ist korrekt, da jede unzulässig gewordene Kante getestet wird:
  - Jede neue Kante e ist zulässig und muß daher nicht mehr getestet werden.
  - Eine Kante kann nur durch Änderung eines der inzidenten Dreiecke unzulässig werden.
- Algorithmus terminiert, da in jeder Iteration der Winkelvektor größer wird und es nur endlich viele Triangulierungen von S gibt.

jede während des Algorithmus neu erzeugte Kante ist eine Kante der DT

Effizienz wird gesteigert durch geeignete *Datenstruktur*:

- Auffinden des den Punkt p_r enthaltenden Dreiecks durch eine Punktlokalisierungsstruktur $\mathcal{D}$, die parallel zur Delaunay-Triangulierung aufgebaut wird:
  - $\mathcal{D}$ ist ein gerichteter azyklischer Graph.
  - Die Blätter von $\mathcal{D}$ entsprechen den Dreiecken der gegenwärtigen Triangulierung $\mathcal{T}$.
  - Jedes Blatt von $\mathcal{D}$ wird durch einen Doppelzeiger mit dem entsprechenden Dreieck in $\mathcal{T}$ verkettet.
  - Interne Knoten von $\mathcal{D}$ entsprechen Dreiecken, die zu einem früheren Zeitpunkt in der Triangulierung enthalten waren, aber später zerstört wurden.
- Initialisierung von $\mathcal{D}$ in Schritt 2 von *DelaunayTriangulation* mit einem einzelnen Blattknoten, der dem Dreieck $p_{-1}p_{-2}p_{-3}$ entspricht.

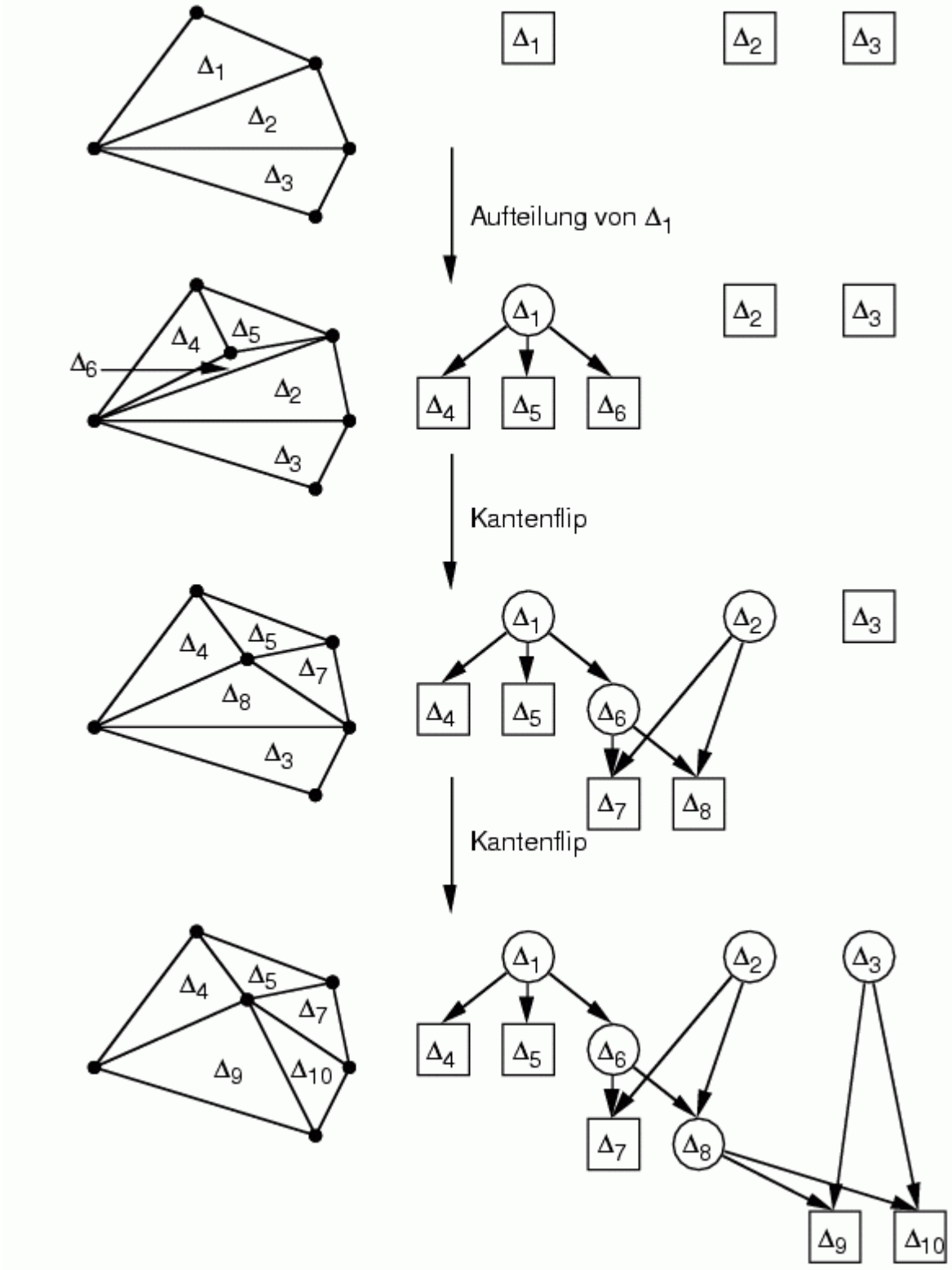
- Aufteilung eines Dreiecks $p_ip_jp_k$ in 3 (oder 2) neue Dreiecke nach Einfügen eines Punktes p_r:
  - Füge 3 (oder 2) neue Blätter der Suchstruktur $\mathcal{D}$ hinzu.
  - Das dem Dreieck $p_ip_jp_k$ entsprechende Blatt von $\mathcal{D}$ wird zu einem internen Knoten mit Zeigern auf die 3 (oder 2) neuen Blätter.
- Bei einem Kantenflip werden zwei Dreiecke $p_ip_jp_k$ und $p_ip_jp_l$ durch die Dreiecke $p_ip_kp_l$ und $p_jp_kp_l$ ersetzt:
  - Erzeuge in $\mathcal{D}$ Blätter für die neuen Dreiecke $p_ip_kp_l$ und $p_jp_kp_l$.
  - Die den Dreiecken $p_ip_jp_k$ und $p_ip_jp_l$ entsprechenden Blätter werden zu internen Knoten mit jeweils 2 Zeigern auf die 2 neuen Blätter.

Die Datenstruktur heißt auch *History* oder *Delaunay-DAG* (directed acyclic graph).

Wenn diese Struktur vorliegt, ist der Suchaufwand beim Einfügen eines neuen Knotens linear zur Anzahl Knoten auf dem Suchpfad, also "niedrig":

- Lokalisierung des nächsten der Triangulierung hinzuzufügenden Punktes p_r:
  - Beginnend am Wurzelknoten von $\mathcal{D}$, der dem Dreieck $p_{-1}p_{-2}p_{-3}$ entspricht, überprüfe die 3 Kinder der Wurzel, um festzustellen, in welchem Dreieck p_r liegt, und steige zu dem entsprechenden Kindknoten hinab.
  - Wiederhole den Vorgang mit den Kindern dieses Knotens usw., bis ein Blatt von $\mathcal{D}$ erreicht wird. Dieses Blatt entspricht einem Dreieck in der gegenwärtigen Triangulierung.

Beispiel:


Δ1
Δ2
Δ3
Aufteilung von Δ1
Δ4
Δ5
Δ6
Kantenflip
Δ7
Δ8
Kantenflip
Δ9
Δ10

Man kann zeigen:
Die *erwartete* Anzahl der Dreiecke, die durch den Algorithmus erzeugt wird, beträgt höchstens 9*n* + 1.
(Beweis: s. Hinrichs 2002)

Satz:

Algorithmus *DelaunayTriangulation* benötigt zur Berechnung einer Delaunay-Triangulierung einer Menge S von n Punkten in der Ebene O(n·log n) erwartete Zeit und hat einen erwarteten Speicherplatzbedarf von O(n).

Beweis:

M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf: Computational Geometry: Algorithms and Applications, Abschnitt 9.4.

Die konvexe Hülle einer Punktmenge

Definition:

Eine Menge $S \subseteq \mathbb{R}^d$ heißt *konvex*, wenn mit $p \in S$ und $q \in S$ auch das Segment, das p und q verbindet, in S liegt, d.h. $\overline{pq} \subseteq S$. $\overline{pq}$ ist dabei die Menge aller Punkte der Form $\alpha p + \beta q$ mit $\alpha \geq 0$, $\beta \geq 0$ und $\alpha + \beta = 1$.
Eine *Konvexkombination* von Punkten $p_1, p_2, \ldots, p_k$ ist eine Summe der Form $\sum_{i=1}^{k} \alpha_i p_i$ mit $\alpha_i \geq 0$, $\sum_{i=1}^{k} \alpha_i = 1$.

- Ein Liniensegment besteht somit aus allen Konvexkombinationen seiner Endpunkte, und ein Dreieck besteht aus allen Konvexkombinationen seiner 3 Eckpunkte. Ein Tetraeder besteht aus allen Konvexkombinationen seiner 4 Ecken.

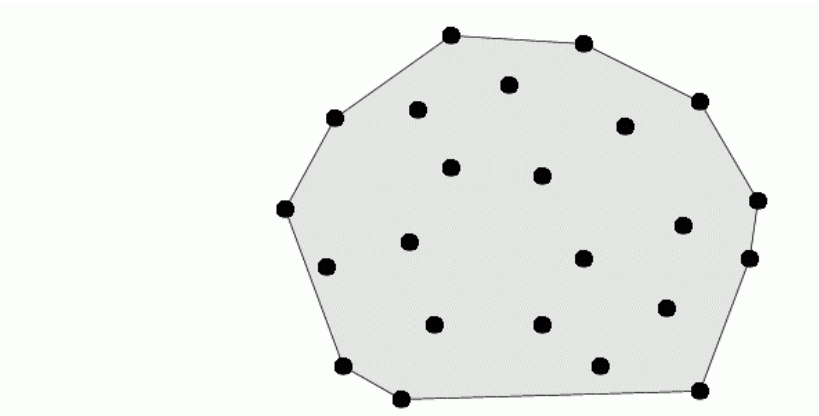
Definition 3.1.2:

Die konvexe Hülle einer Menge $S \subseteq \mathbb{R}^d$ ist

$$ch(S) = conv(S) = \bigcap_{\substack{T \supset S \\ T \text{ konvex}}} T \quad .$$

- Durchschnitt konvexer Mengen konvex $\Rightarrow$ ch(S) konvex $\Rightarrow$ konvexe Hülle einer Menge S ist die kleinste konvexe Menge, die S umfaßt.

(Hinrichs 2002)



Analogie (?): Nägel und Gummiband

Charakterisierung der konvexen Hülle:

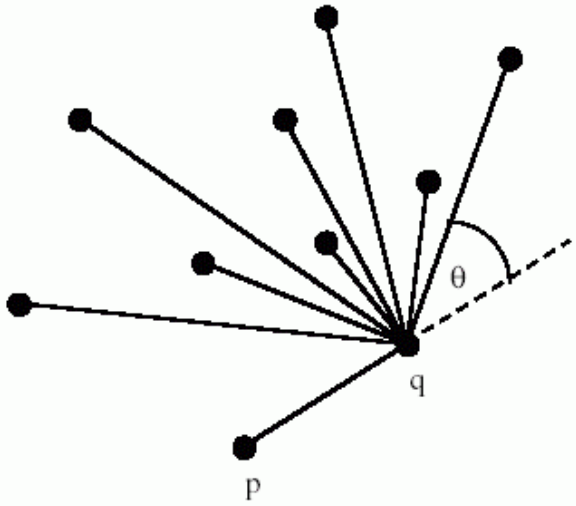
1. Die konvexe Hülle von S ist die Menge aller Konvexkombinationen von Punkten aus S.
2. Die konvexe Hülle von $S \subseteq \mathbb{R}^d$ ist die Menge aller Konvexkombinationen von d+1 (oder weniger) Punkten aus S.
3. Die konvexe Hülle von $S \subseteq \mathbb{R}^d$ ist der Schnitt aller Halbräume, die S umfassen. Ein Halbraum im 2-d ist eine Halbebene, d.h. die Menge aller Punkte, die auf oder auf einer bestimmten Seite einer Geraden liegen. Ein Halbraum im 3-d ist die Menge aller Punkte, die auf oder auf einer bestimmten Seite einer Ebene liegen. Die Verallgemeinerung des Begriffs Halbraum auf den $\mathbb{R}^d$ ist offensichtlich.
4. Die konvexe Hülle einer endlichen Menge von Punkten $S \subseteq \mathbb{R}^2$ ist das kleinste konvexe Polygon P, das S umfaßt, d.h. es gibt kein anderes konvexes Polygon P' mit $P \supset P' \supseteq S$, $P \neq P'$.
5. Die konvexe Hülle einer endlichen Menge von Punkten $S \subseteq \mathbb{R}^2$ ist das S umfassende konvexe Polygon P mit der kleinsten Fläche.
6. Die konvexe Hülle einer endlichen Menge von Punkten $S \subseteq \mathbb{R}^2$ ist das S umfassende konvexe Polygon P mit dem kleinsten Umfang.
7. Die konvexe Hülle einer Menge von Punkten $S \subseteq \mathbb{R}^2$ ist die Vereinigung aller Dreiecke, die durch Punkte aus S aufgespannt werden. Dies ist eine Reformulierung von 2) für d = 2.

(Hinrichs 2002)

1. Algorithmus zur Konstruktion der konvexen Hülle:

Jarvis's march

- Algorithmus von Jarvis (1973), auch bekannt als gift wrapping Methode
- Nutze nach Bestimmung einer Kante $\overline{pq}$ der konvexen Hülle aus, daß eine weitere Kante existieren muß, die q als einen ihrer Endpunkte besitzt. Diese Kante ist diejenige, die mit $\overline{pq}$ den kleinsten Winkel θ bildet:



vom aktuellen Endpunkt aus wird zu jedem der übrigen Punkte der Winkel bestimmt; der Punkt mit dem kleinsten Winkel bestimmt die nächste (Extrem-) Kante, die zur konvexen Hülle gehört.

Startpunkt: Punkt mit kleinster *y*-Koordinate

Anschauliche Vorstellung: ein Band wird um die Menge herumgelegt und, ausgehend vom Startpunkt, jeweils zum Punkt mit dem kleinsten Winkel "straffgezogen".

verallgemeinerungsfähig auf höhere Dim. $\Rightarrow$ dann nicht Band, sondern "Fläche" zum Einwickeln $\Rightarrow$ daher der Name "gift wrapping".

Pseudocode:

Algorithm: GIFT WRAPPING
Find the lowest point (smallest y coordinate).
Let i_0 be its index, and set $i \leftarrow i_0$.
repeat
    for each $j \neq i$ do
        Compute counterclockwise angle θ from previous hull edge.
    Let k be the index of the point with the smallest θ.
    Output (p_i, p_k) as a hull edge.
    $i \leftarrow k$
until $i = i_0$

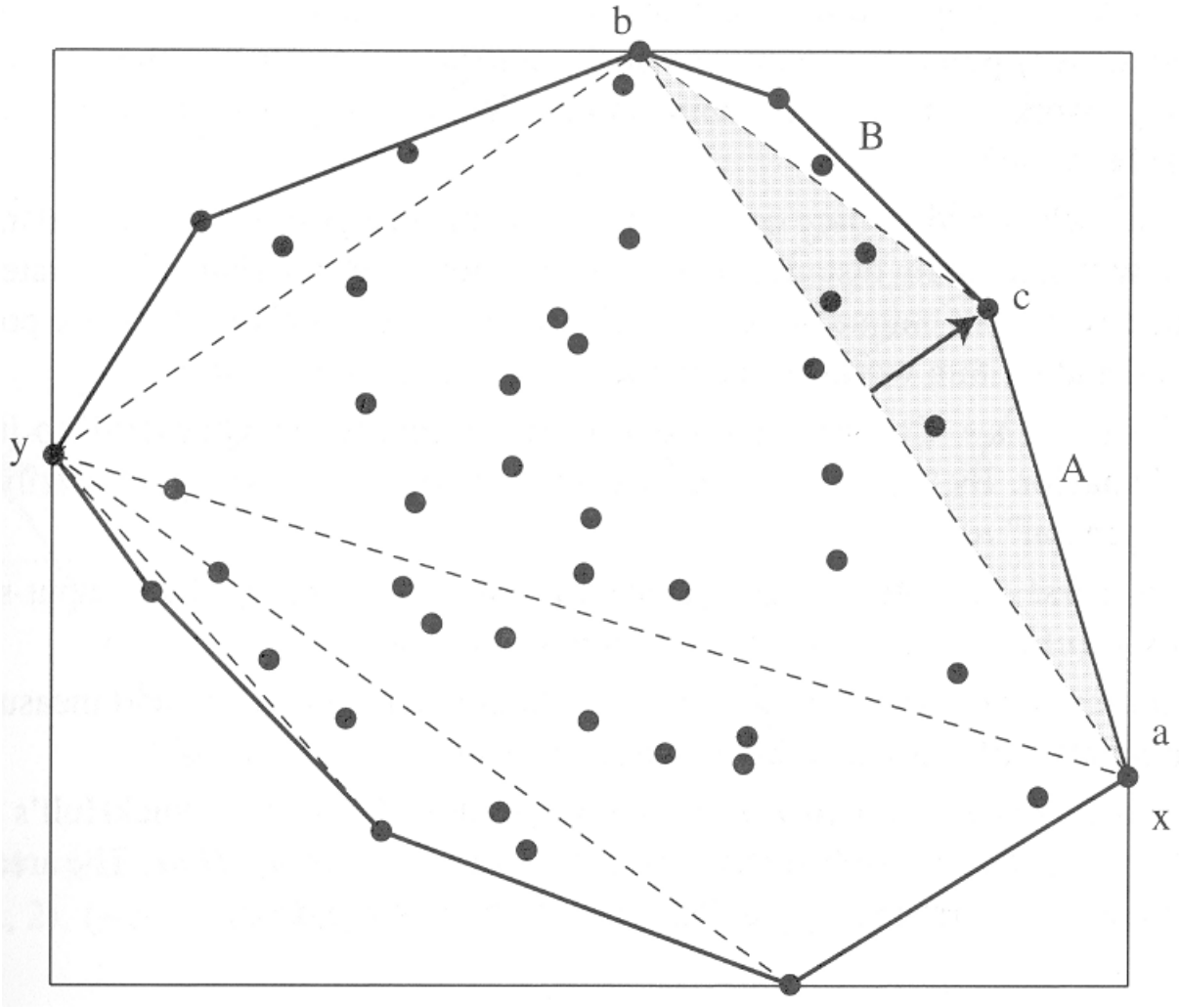
Zahl der Schleifendurchläufe:

- äußere Schleife: so oft, wie neue Kanten gezogen werden
- innere Schleife: n–1 Punkte werden zum Winkelvergleich herangezogen

$\Rightarrow$
Jarvis's March bestimmt die konvexe Hülle einer gegebenen Menge von n Punkten im R^2 in der Zeit $O(nh)$, wenn h die Zahl der Ecken der konvexen Hülle ist (outputsensitiver Algorithmus) und mit Speicherplatzbedarf $O(n)$.

2. Algorithmus: QuickHull

- Analogie zu Quicksort
- divide-and-conquer-Prinzip
- Grundidee: zunächst die "vielen" Punkte hinauswerfen, die klar im Inneren liegen, dann auf Bereiche näher am Rand konzentrieren und diese rekursiv genauso behandeln
- zunächst wird Strecke zwischen 2 Extrempunkten aus S gezogen (z.B. mit min. und max. x-Koordinate), dann wird oberhalb dieser Kante ein weiterer Extrempunkt (mit max. Abstand von der Kante) bestimmt und der Inhalt des Dreiecks eliminiert
- setze dies rekursiv für die verbleibenden Rest-Mengen fort



(aus O'Rourke 1998)

Pseudocode:

Algorithm: QUICKHULL
function *QuickHull*(a, b, S)
    if $S = \emptyset$ then return ()
    else
        $c \leftarrow$ index of point with max distance from ab.
        $A \leftarrow$ points strictly right of (a, c).
        $B \leftarrow$ points strictly right of (c, b).
        return *QuickHull*$(a, c, A) + (c) +$ *QuickHull*(c, b, B)

Komplexität:
QuickHull bestimmt die konvexe Hülle einer gegebenen Menge von n Punkten im "besten" Fall in der Zeit $O(n \log n)$, im schlimmsten Fall jedoch in der Zeit $O(n^2)$.
Speicherplatzbedarf $O(n)$. (Hinrichs 2001)

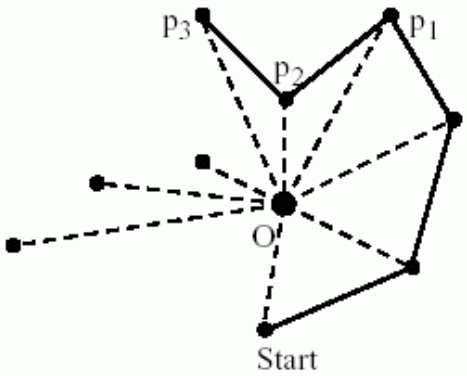
3. Algorithmus:

Graham's scan

- Ist es wirklich notwendig, wie in Algorithmus EXT-PUNKTE alle $O(n^3)$ Dreiecke zu betrachten, um die Extremalpunkte zu finden?
- Graham zeigte 1972, daß man die Extremalpunkte in linearer Zeit finden kann, wenn man im Gegensatz zum Algorithmus EXT-PUNKTE den Sortierschritt zuerst durchführt.
- Annahme: Ein Punkt im Innern der konvexen Hülle sei bekannt. OBdA sei dies der Ursprung des Koordinatensystems.

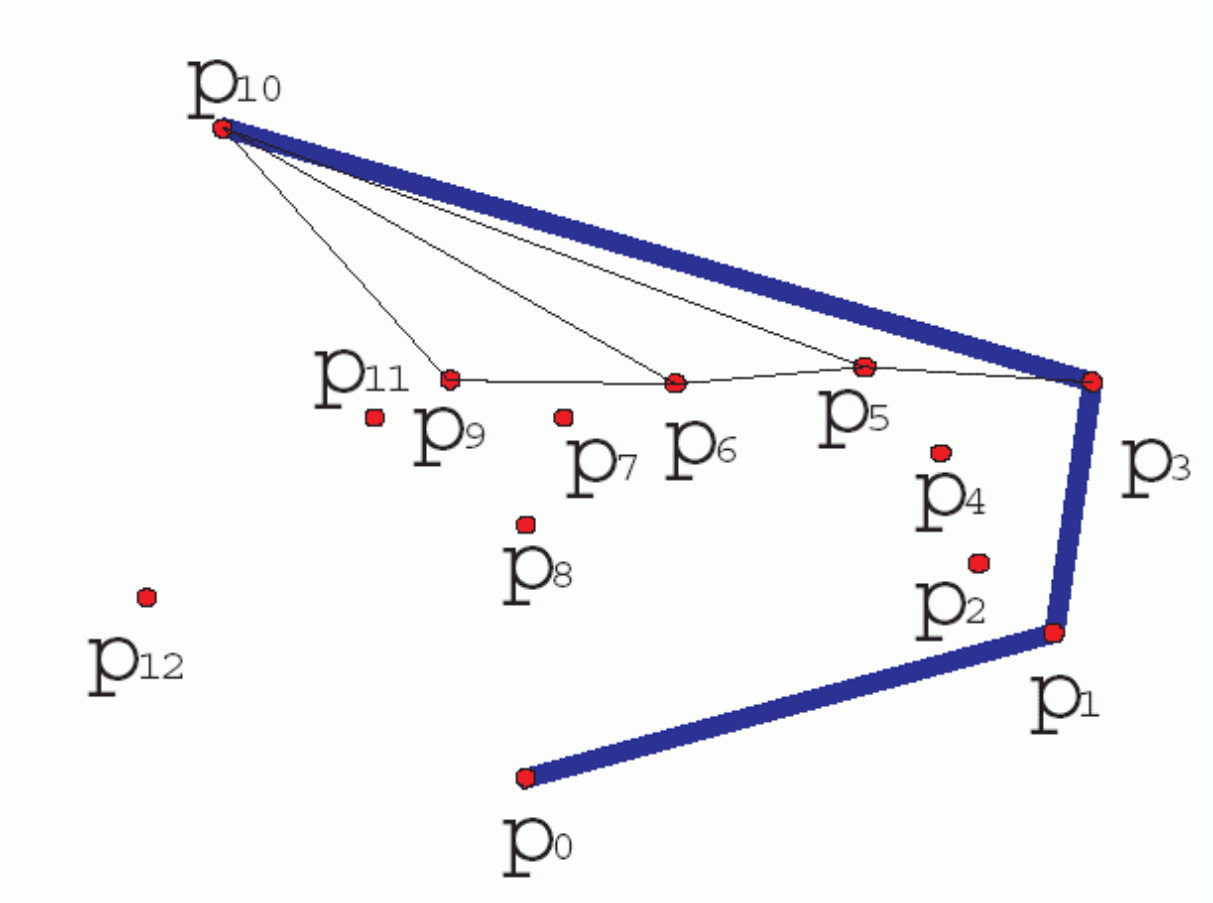
- Man sortiere die Punkte lexikografisch bzgl. des Polarwinkels (1. Kriterium) und der Entfernung zum Ursprung (Vergleich bzgl. Entfernung erfolgt nur, wenn die Punkte beide den gleichen Polarwinkel besitzen). Die sortierten Punkte werden in einer doppelt verketteten Liste abgelegt.

- Ist ein Punkt nicht Eckpunkt der konvexen Hülle, so liegt er in einem Dreieck (O, p, q), wobei p und q zwei aufeinanderfolgende Eckpunkte der konvexen Hülle sind.
- Graham's Algorithmus durchläuft die geordnete Liste und eliminiert dabei die Punkte, die nicht Eckpunkte der konvexen Hülle sind. Nach dem Durchlauf bleiben die Eckpunkte der konvexen Hülle in der gewünschten Anordnung übrig.



(hier würde p_2 eliminiert)

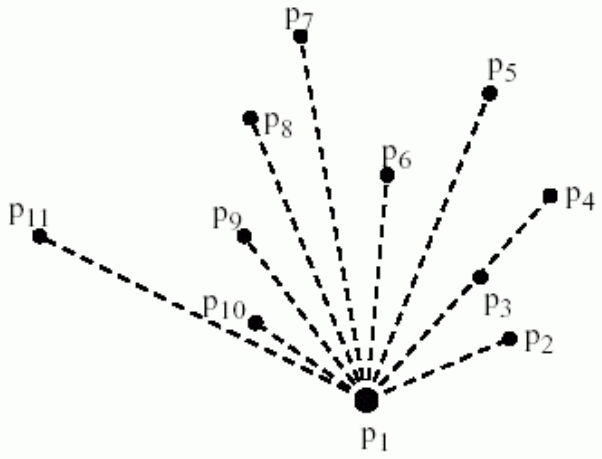
weiteres Beispiel:



genauere Beschreibung:

- Beginne Durchlauf an einem Punkt, der sicherlich Eckpunkt der konvexen Hülle ist, z.B. am kleinsten Punkt bzgl. $<_y$.
- Untersuche wir wiederholt Tripel von aufeinanderfolgenden Punkten, die um den Ursprung entgegen dem Uhrzeigersinn angeordnet sind, daraufhin, ob sie einen reflexiven Winkel, d.h. einen Winkel $\geq \pi$, bilden. Ist der innere Winkel $\angle(p_1, p_2, p_3)$ reflexiv, so bilden (p_1, p_2, p_3) eine Rechtsdrehung, ansonsten eine Linksdrehung.
- Konvexität $\Rightarrow$ beim Durchlauf der Eckpunkte der konvexen Hülle entgegen dem Uhrzeigersinn nur Linksdrehungen
- (p_1, p_2, p_3) Rechtsdrehung $\Rightarrow$ p_2 nicht Eckpunkt der konvexen Hülle
- Durchlauf der geordneten Punkteliste abhängig vom Winkeltest:
  1. Bilden (p_1, p_2, p_3) eine Rechtsdrehung, so wird p_2 aus der Liste eliminiert und als nächstes Tripel (p_0, p_1, p_3) betrachtet.
  2. Bilden (p_1, p_2, p_3) eine Linksdrehung, so schreitet man in der Liste weiter voran und betrachtet als nächstes das Tripel (p_2, p_3, p_4).
- Durchlauf terminiert, wenn der Ausgangspunkt wieder erreicht wird.
- Winkeltest benötigt O(1) Zeit.
- Nach jedem Test schreiten wir entweder voran (Schritt 2), oder wir eliminieren einen Punkt. Da es nur n Punkte gibt, kann man höchstens n Vorwärtsschritte durchführen, aber auch höchstens n Punkte eliminieren und daher höchstens n-mal zurückgehen $\Rightarrow$ Durchlauf benötigt O(n) Zeit.

- Modifikation des Algorithmus von Graham: Sortiere die gegebenen Punkte nicht bzgl. eines Punktes im Innern der konvexen Hülle, sondern bzgl. eines Eckpunktes der konvexen Hülle, z.B. des kleinsten Punktes bzgl. $<_y$.



Komplexität:

Graham's scan bestimmt die konvexe Hülle einer gegebenen Menge von n Punkten in optimaler Zeit O(n·log n) und Speicherplatz O(n).

Nachteil jedoch:
Erweiterung auf höhere Dimensionen nicht möglich, da Graham's scan grundlegend auf der Sortierung der Punkte nach ihrem Polarwinkel beruht, was sich nicht auf höhere Dim. übertragen lässt.