Konstruktion des Voronoi-Diagramms

Untere Schranke für den Zeitaufwand:

für n-elementige Punktmenge jedenfalls $\Omega(n \log n)$, da mit VD die konvexe Hülle in linearer Zeit bestimmbar.

Wenn man die n Punkte nach einer Koordinate sortiert hat, lässt sich die konvexe Hülle anschließend in linearer Zeit konstruieren
Das Voronoi-Diagramm ist "schwieriger":

Satz:
Angenommen, die n Punkte in S sind bereits nach auf-steigenden x-Koordinaten sortiert. Dann erfordert die Konstruktion des Voronoi-Diagramms von S immer noch $\Omega(n \log n)$ Zeit.

Beweis: s. Klein 1997, S. 270 f.

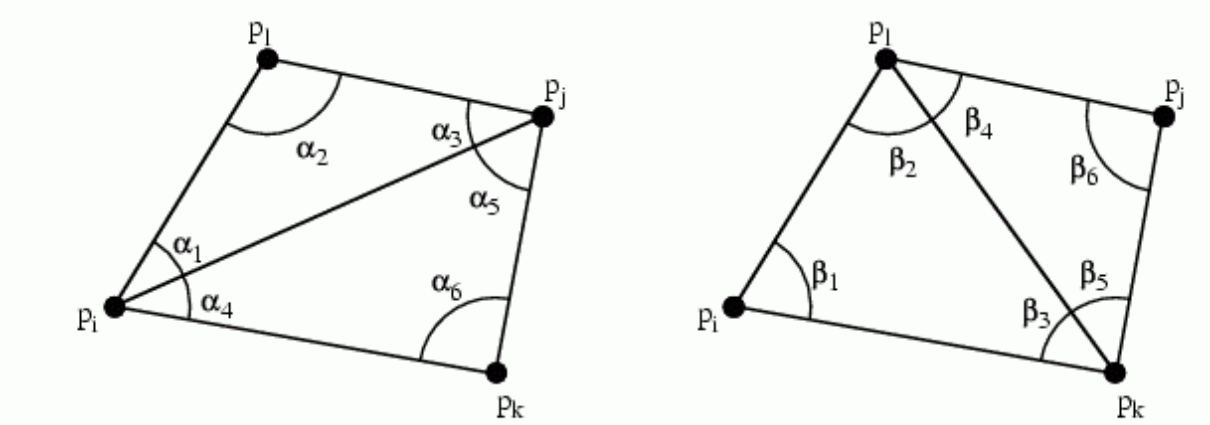
Naives Verfahren:

jede der n Voronoi-Regionen als Schnitt von n–1 Halbebenen berechnen
$\Rightarrow$ jede Region in Zeit $O(n^2)$, oder effizienter in Zeit $O(n \log n)$
$\Rightarrow$ gesamtes VD in Zeit $O(n^3)$ bzw. $O(n^2 \log n)$

Wenn Delaunay-Triangulierung vorhanden, VD in Zeit $O(n)$ konstruierbar (und umgekehrt): Konstruktion des dualen Graphen

erstes effizienteres Verfahren konstruiert die Delaunay-Triangulierung, und zwar inkrementell (Hinzufügen von Punkten zu S)

dazu zunächst Benennungen:



- 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)

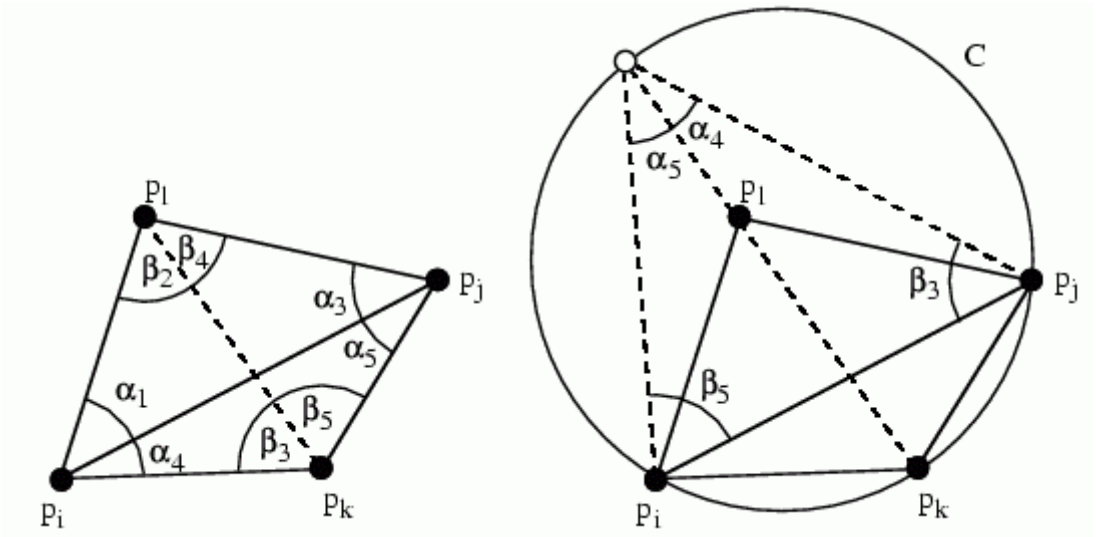
man nennt dies auch einen "Lawson-Flip".

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:

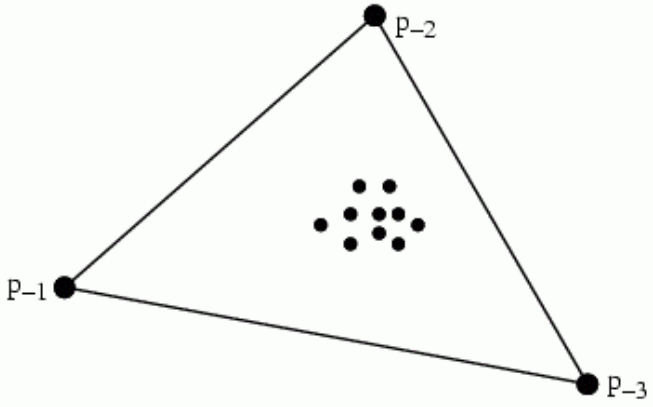
Die Kante $\overline{p_i p_j}$ sei inzident zu den Dreiecken $p_i p_j p_k$ und $p_i p_j p_l$, und C sei der Kreis durch p_i, p_j und p_k. Die Kante $\overline{p_i p_j}$ ist unzulässig genau dann, wenn der Punkt p_l im Innern von C liegt. Bilden die Punkte p_i, p_j, p_k und p_l ein konvexes Viereck und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

(Beweis s. Hinrichs 2002)



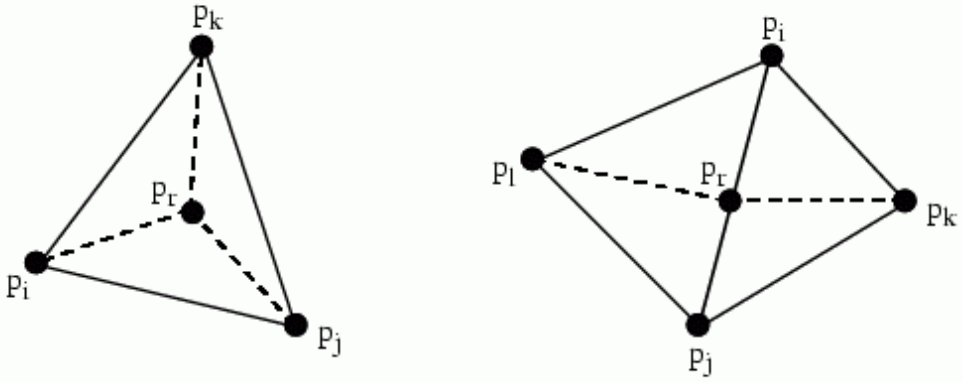
Randomisierter, inkrementeller Algorithmus zur Berechnung der DT (nach Hinrichs 2002):

- 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) :

DelaunayTriangulation(S)

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

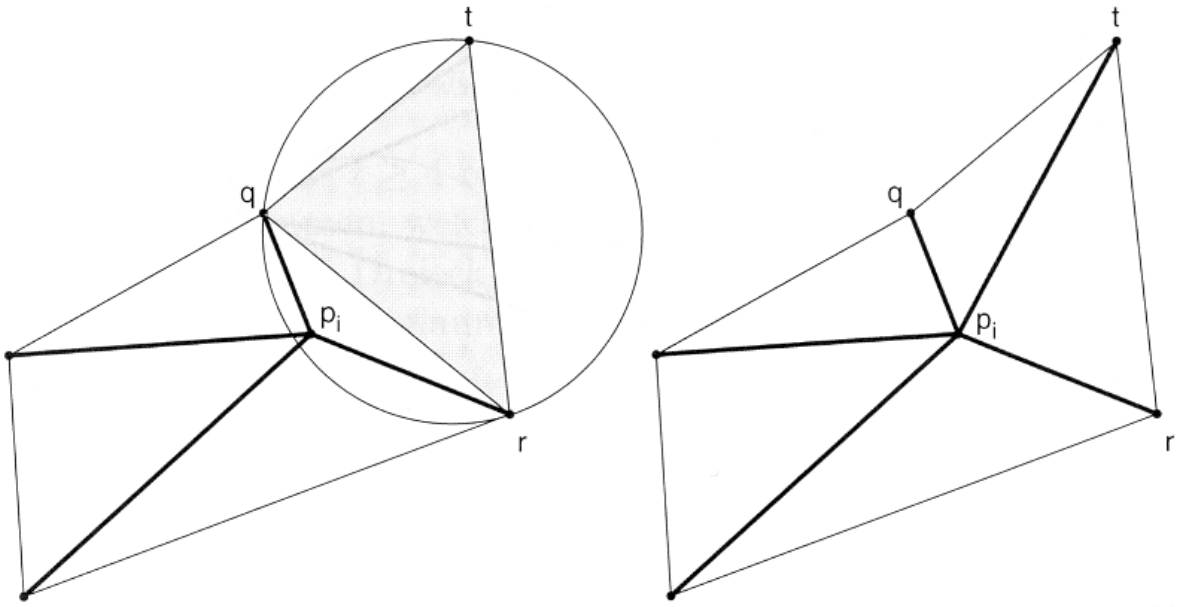
Ausgabe: Eine Delaunay-Triangulierung von S.

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

1. if $\overline{p_i p_j}$ ist unzulässig then
2. Sei $p_i p_j p_k$ das zu $p_r p_i p_j$ entlang $\overline{p_i p_j}$ benachbarte Dreieck.
3. Flippe $\overline{p_i p_j}$, d.h. ersetze $\overline{p_i p_j}$ durch $\overline{p_r p_k}$.
4. *LegalizeEdge*(p_r, $\overline{p_i p_k}$, $\mathcal{T}$)
5. *LegalizeEdge*(p_r, $\overline{p_k p_j}$, $\mathcal{T}$)

- 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.

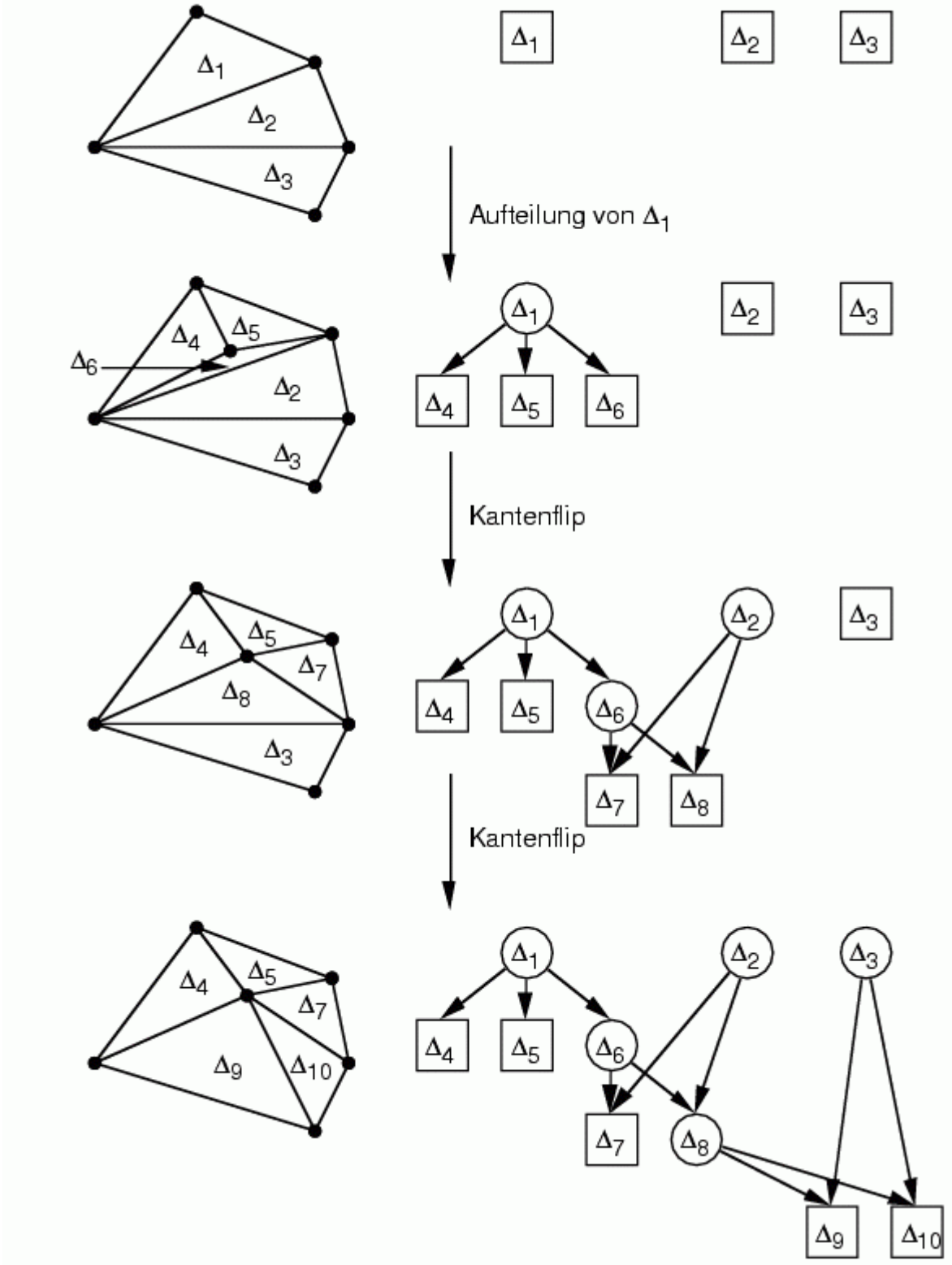
- Aufteilung eines Dreiecks $p_i p_j p_k$ in 3 (oder 2) neue Dreiecke nach Einfügen eines Punktes p_r:
  - Füge 3 (oder 2) neue Blätter der Suchstruktur $\mathcal{D}$ hinzu.
  - Das dem Dreieck $p_i p_j p_k$ entsprechende Blatt von $\mathcal{D}$ wird zu einem internen Knoten mit Zeigern auf die 3 (oder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

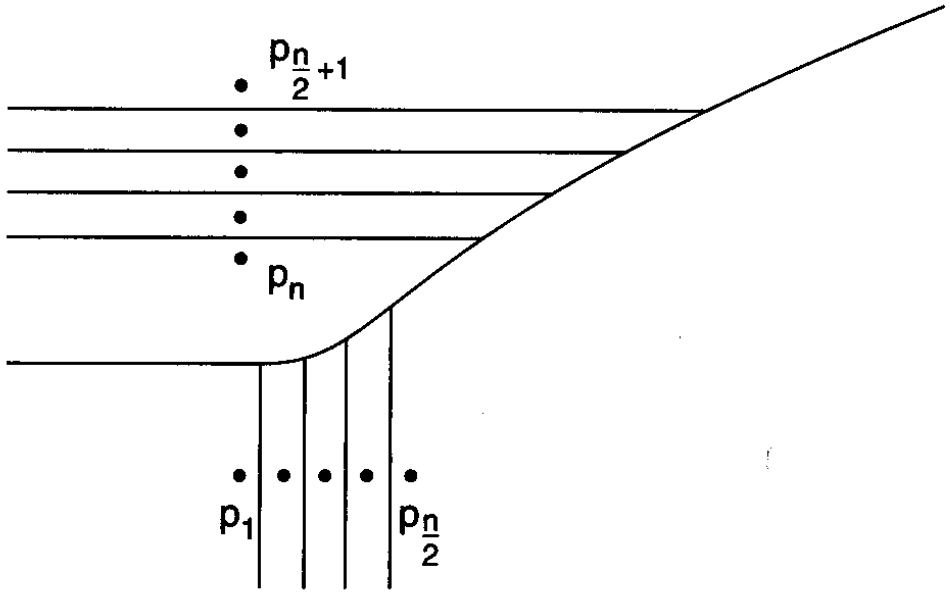
Man kann zeigen:
Die *erwartete* Anzahl der Dreiecke, die durch den Algorithmus erzeugt wird, beträgt höchstens $9n + 1$.
(Beweis: s. Hinrichs 2002)

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.

Jedoch: *worst case* - Verhalten ungünstiger

jedes inkrementelle Konstruktionsverfahren für das VD benötigt im ungünstigsten Fall $\Omega(n^2)$ viele Schritte:

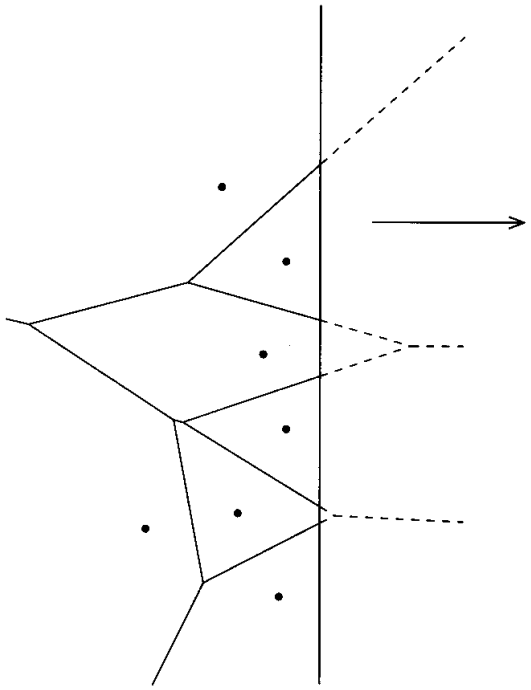


(Klein 1997)

Punkte auf *x*-Achse seien bereits eingefügt, Punkte auf *y*-Achse sollen von oben nach unten eingefügt werden $\Rightarrow$ VR des jeweils letzten Punktes hat mit denen der unteren Punkte gem. Kanten, diese müssen alle wieder entfernt werden beim Neueinfügen.

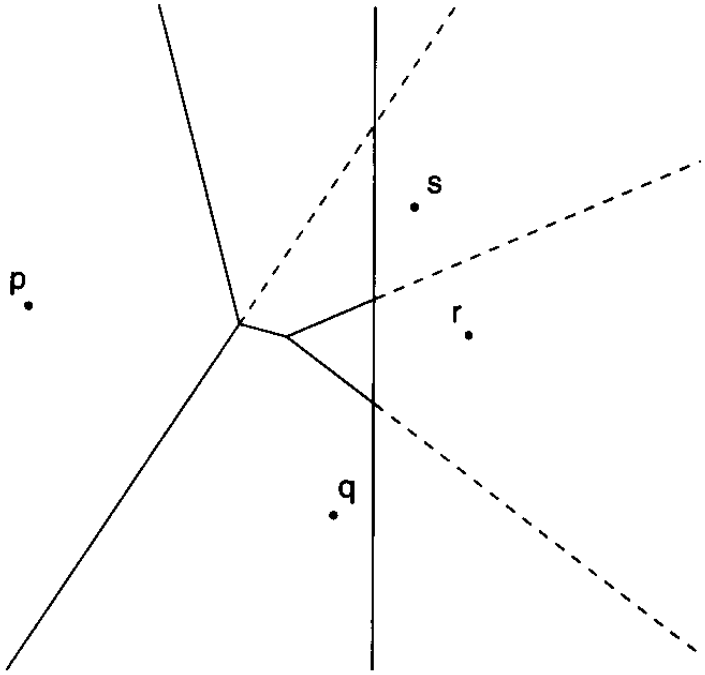
besseres Verfahren (auch im worst case):

Sweep-Verfahren



Idee: Bewege Sweep–line von links nach rechts über S,
berechne sukzessive $VD(S)$

Problem: $VR(p)$ beginnt schon vor Ort p



Beobachtung:

Es gibt ein Gebiet G links der Sweepline L,

für das das VD bereits festliegt:

$G = \{v \in \mathbb{R}^2 : \quad |\overline{vp}| \leq \text{Abstand}(v, L) \text{ für ein } p \in S\}$

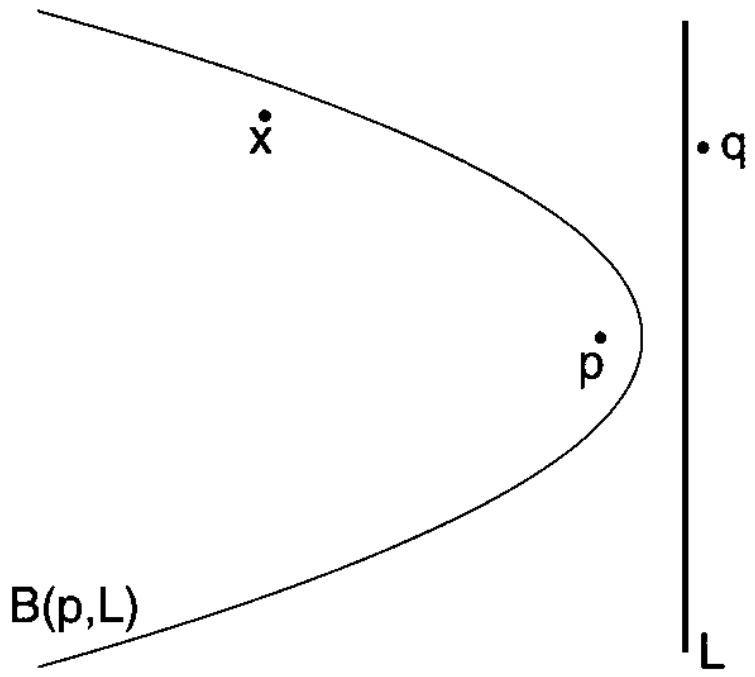
für $S = \{p\}$: Inneres der Parabel $B(p, L)$:

$(p.x - v.x)^2 + (p.y - v.y)^2 \leq v.x - L.x$

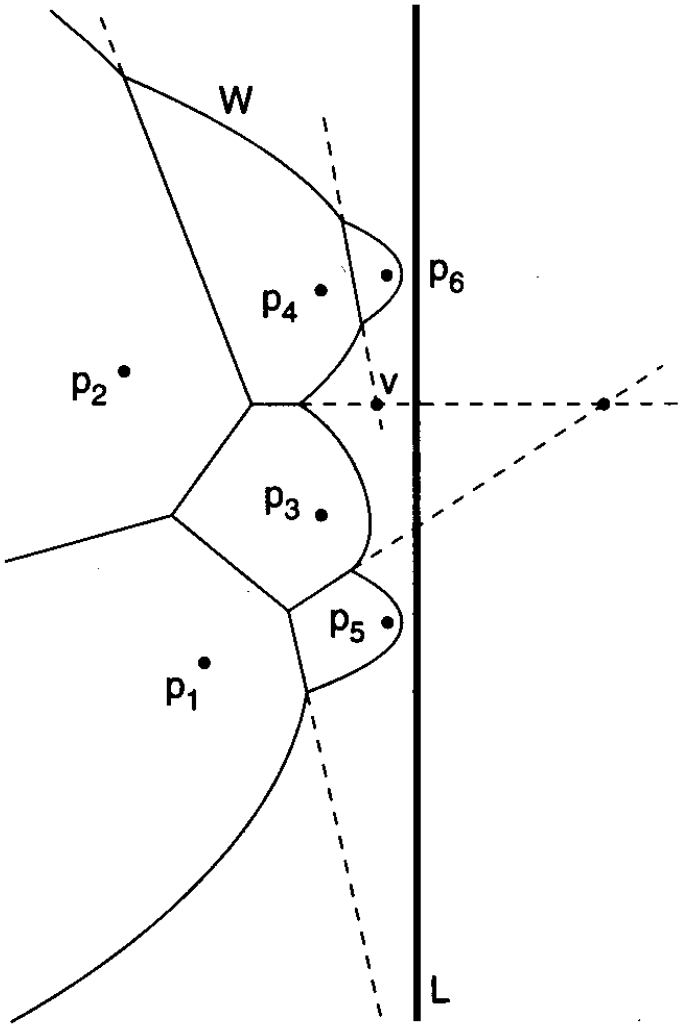
i.allg. wird G durch Folge W ("Wellenfront")

von Parabelbögen ("Wellenstücken") $P(p, q, r)$ begrenzt

Kein Punkt rechts der Sweep-Geraden kann das Gebiet zur Linken der Parabel beeinflussen:



"Wellenfront" *W*:
Rand der Voronoi-Region der Geraden *L*



Die Wellenfront wird manchmal auch "beach line" genannt (Gärtner 1996).

$aktive\ Punkte =$ die Punkte q aus S, die gerade ein Wellenstück $P(p,q,r)$ zu W beitragen

Beobachtung: Schnittpunkt zweier in W benachbarter Wellenstücke (z.B. Teile von $B(p,L)$ und $B(q,L)$) rückt längs des geraden Bisektors $B(p,q)$ vor.

$Spike\ s(p,q)$
$=$ Verlängerung des Bisektors $B(p,q)$ jenseits W; p, q aktiv

Beobachtung: Schnittpunkte auf W benachbarter Spikes sind potentielle Voronoi–Knoten

$\rightarrow$ als zukünftige Ereignisse vormerken

Beobachtung: W ist zusammenhängend und y–monoton

$\rightarrow$ verwalte Wellenfront in der Y–Struktur SSS,

Wellenstücke $P(p,q,r)$ nach y–Koordinaten geordnet

$+$ Verweise auf Spikes:

oberhalb $P(p,q,r)$: $s(p,q)$, unterhalb: $s(q,r)$

Beobachtung: $VR(L, S \cup \{L\})$ hat $O(n)$ viele Ecken

$\rightarrow |W| = O(n)$

Zwei Arten von Ereignissen:

1. site event: ($Punkt$–$Ereignis$)

Sweep–line L trifft neuen Punkt v aus S

$\rightarrow$ Bisektor $B(v,L)$ zunächst degenerierte Parabel $\overline{wv}$

mit $w =$ Schnitt von W mit Horizontalgerade durch v,

$\overline{wv}$ öffnet sich beim Fortschreiten von L

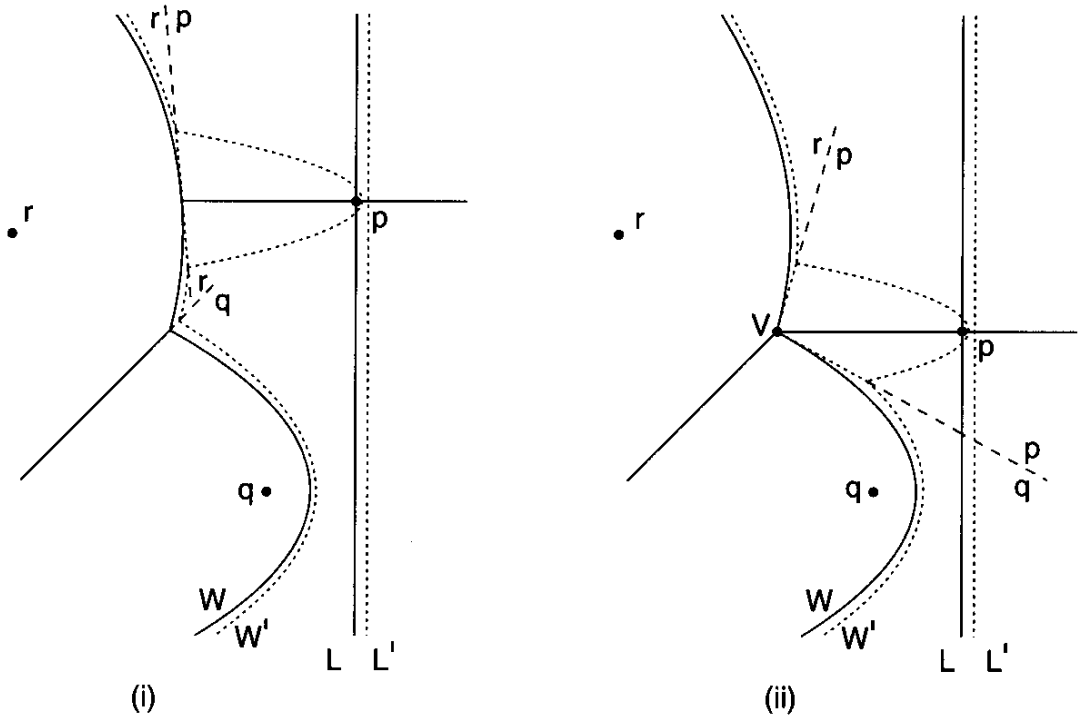
Sei $q \in S$ (aktiver) Punkt mit $w \in P(p,q,r) \subseteq W$.

$P(p,q,r)$ aufspalten in $P(p,q,v)$, $P(q,v,q)$, $P(v,q,r)$

Bei Fortschreiten von L über v hinweg

entstehen 2 neue Spikes $s(q,v) = \overline{w?}$, $s(v,q) = \overline{w?}$

aktualisiere Schnittpunkte benachbarter Spikes in ES



Spezialfall: w liegt auf einem Spike $s(p,q) = \overline{y?}$

$\rightarrow w$ ist Voronoiknoten. Ausgabe w, (y,w)

neues Wellenstück $P(p,v,q)$ einfügen,

es entstehen 2 neue Spikes $s(p,v)$, $s(v,q)$

Grenzen von $P(p,v,q)$ laufen entlang $s(p,v)$, $s(v,q)$

2. circle event (*spike–Ereignis*)

Wellenstück $P(p,q,r)$ aus W trifft auf den Schnittpunkt z seiner beiden Spikes $s(p,q) = \overline{x?}$ und $s(q,r) = \overline{y?}$ und verschwindet

d.h. L ist rechte Tangente an Kreis $\odot(p,q,r)$ mit Zentrum z

$\rightarrow z$ ist Voronoiknoten. Ausgabe z, (x,z), (y,z)

lösche $P(p,q,r)$ mit $s(p,q)$, $s(q,r)$ aus SSS

lösche weitere Schnittpunkte mit $s(p,q)$, $s(q,r)$ aus ES

berechne neuen Spike $s(p,r)$

aktualisiere SSS–Vorgänger $P(l,p,q)$ zu $P(l,p,r)$

aktualisiere SSS–Nachfolger $P(q,r,t)$ zu $P(p,r,t)$

berechne Schnittpunkte jetzt benachbarter Spikes $\rightarrow ES$

Bemerkung: Zeitpunkt des Spike–Ereignis: $z.x + |\overline{xz}|$

Bemerkung: Weitere Ereignisse gibt es nicht:

Solange die Spikes sich nicht schneiden und keine neuen Punkte getroffen werden, laufen die Wellenstücke ungehindert zwischen ihren Spikes weiter.

Ereignisstruktur ES

enthält (x–) Koordinaten der Punkte aus S

bzw. der Schnittpunkte in SSS benachbarter Spikes

und für Schnittpunkte $z = s(p,q) \cap s(q,r)$:

Verweis auf Wellenstück $P(p,q,r)$ in SSS

Operationen: *deletemin*, *insert*, *delete*, *empty*

$\rightarrow$ priority queue, z.B. als bal. binärer Suchbaum

Sweep–Status–Struktur SSS

Operationen: *insert*, *delete*, *lookup*

für Wellenstücke P nach y–Koordinaten

$\rightarrow$ in Zeit $O(\log |SSS|)$ z.B. bei bal. bin. Suchbaum

Der Algorithmus:

initialisiere SSS, ES

x–sortiere die $p \in S$, füge in ES als site events ein

while ! $ES.empty()$ **do**

$E \leftarrow ES.deletemin()$; // get next event

Fall 1: E ist Punkt–Ereignis: ... (s.o.)

Fall 2: E ist Spike–Ereignis: ... (s.o.)

od;

for alle in SSS verbliebenen Wellenstücke $P(p,q,r)$:

Ausgabe $s(p,q)$, $s(q,r)$ als einseitig unbeschränkte Voronoikanten

Komplexität des Algorithmus:

$|SSS| = O(n)$ s.o.

$|ES| = O(n)$ falls nur Schnittpunkte auf W benachbarter Spikes in ES gehalten werden

$\rightarrow$ je Ereignisbearbeitung Zeit $O(\log n)$

$\exists\ O(n)$ viele Ereignisse, da jedes $\geq$1 Voronoiknoten/–kante ergibt

$\rightarrow$ gesamt Zeit $O(n \log n)$

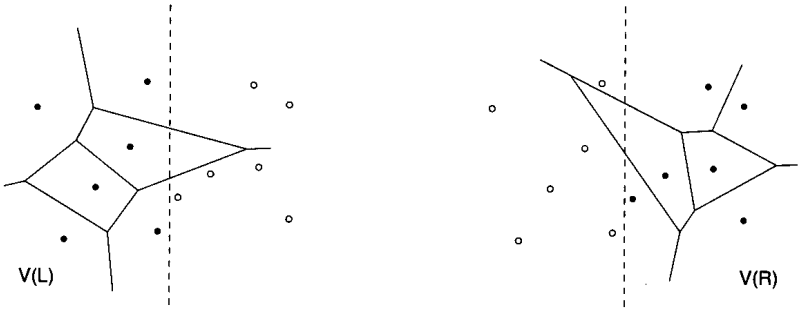
(Keßler 1998)

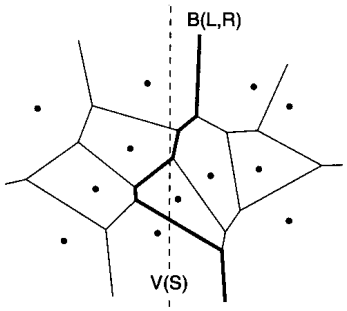
$\Rightarrow$

Satz:
Das Voronoi-Diagramm von *n* Punkten in der Ebene lässt sich mit dem Sweep-Verfahren im *worst case* in der Zeit *O*(*n* log *n*) und mit linearem Speicherplatz berechnen, und das ist optimal.

Dieses Sweep-Verfahren hat den Namen "*Fortune's Sweep*".

Bemerkung:
Mit größenordnungsmäßig ebenso günstigem Aufwand lässt sich das VD auch mit dem *divide-and-conquer*-Ansatz bestimmen (Shamos 1975).
Prinzip: Teilung der Punktmenge in 2 Teilmengen längs einer Splitgeraden, Mischen der beiden VD der Teilmengen (s. Klein 1997).





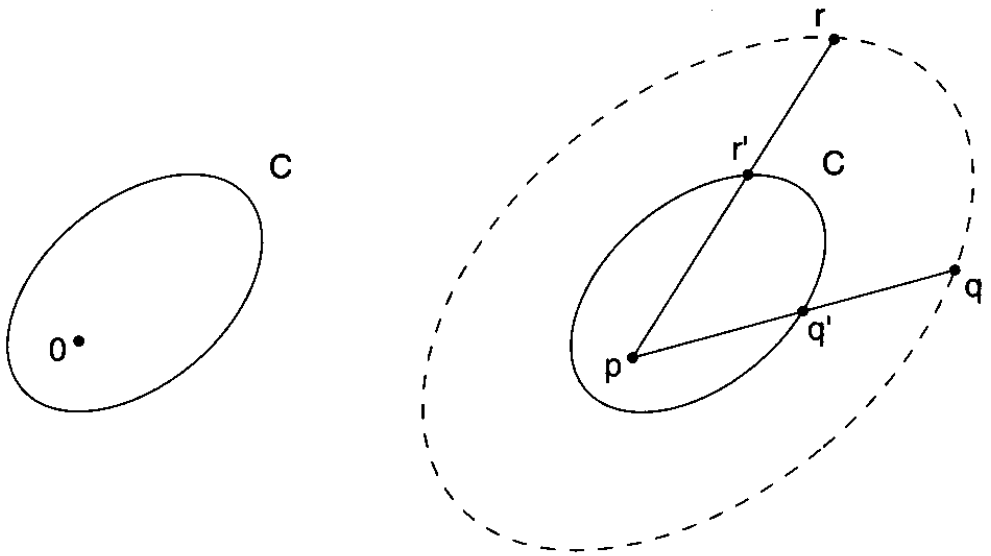
Verallgemeinerungen des Voronoi-Diagramms

Zugrundelegung anderer Distanzfunktionen

- Metriken d_k (vgl. Kap. 2.7), insbes. Manhattan-Distanz d_1
- nochmalige Verallgemeinerung: *konvexe Distanzfunktionen*

Sei *C* eine kompakte, konvexe Menge in der Ebene,
die den Nullpunkt im Inneren enthält (0 = "Zentrum" von *C*).
Abstand von *p* nach *q*:

- verschiebe *C* um den Vektor *p* (dann wird *p* zum Zentrum)
- bilde Strahl von *p* durch *q*
- dieser schneidet den Rand von *C* in genau einem Punkt *q*'
- setze d_C (*p*, *q*) = |*pq*| / |*pq*'| = Faktor, um den die nach *p* verschobene Kopie von *C* skaliert werden müsste, damit ihr Rand den Punkt *q* enthält
- zusätzlich sei d_C (*p*, *p*) = 0.



d_C heißt konvexe Distanzfunktion,
erfüllt pos. Definitheit und Dreiecksungleichung.
Symmetrie erfüllt ⇔ *C* symmetrisch zum Nullpunkt.

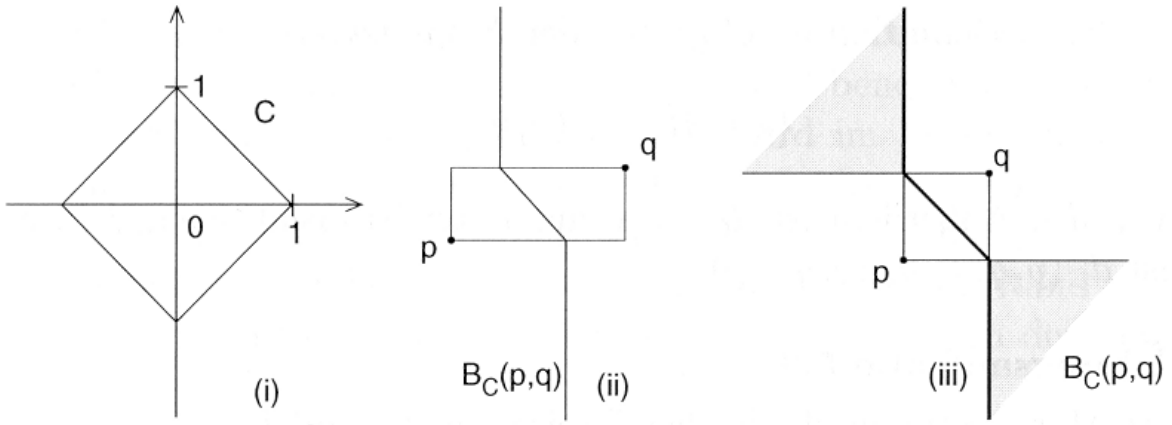
Der Einheitskreis von d_C , d.h. { x | d_C (x, 0) ≤ 1}, ist *C*.

d_C ist translationsinvariant.

d_k-Metriken sind Spezialfälle von d_C .

Definition des *Bisektors* zweier Punkte bzgl. d_C :
analog zum euklidischen Fall (Menge aller Punkte mit gleichem Abstand zu beiden Punkten)

Einheitskreis (i) und Bisektoren (ii, iii) der Manhattan-Metrik:



Der "pathologische Fall" des flächigen Bisektors (iii) kann nur auftreten, wenn der Einheitskreis *C* stellenweise abgeplattet ist (Geradenstücke in seinem Rand enthält)
Def.: *C* streng konvex $:\Leftrightarrow$ ∂C enthält kein Geradenstück

für kompakte, streng konvexe Mengen *C* in der Ebene mit dem Nullpunkt im Inneren ist jeder Bisektor bzgl. d_C homöomorph zu einer Geraden.

(Beweis: s. Klein 1997, S. 243 f.)

Voronoi-Region von *p* in einer Punktmenge *S* bzgl. d_C :
Schnitt aller offenen Gebiete, die vom Bisektor zwischen *p* und *q* begrenzt werden und *p* enthalten, wobei *q* alle Punkte aus $S \setminus \{p\}$ durchläuft. (= analog zum euklidischen Fall)

Achtung: verallgemeinerte Voronoi-Regionen sind i.allg. nicht konvex (da die Bisektoren keine Geraden sind).

Aber es gilt:
Jede Voronoi-Region bzgl. d_C , *C* streng konvex, ist sternförmig und enthält *p* in ihrem Kern.
($\Rightarrow$ sie ist insbesondere zusammenhängend.)

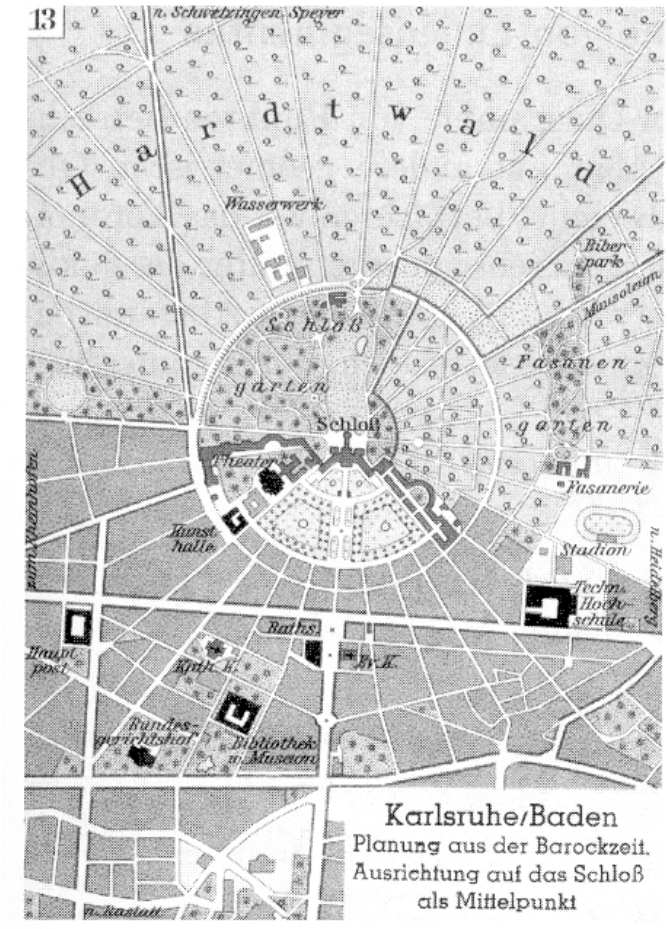
(Beweis: s. Klein 1997, S. 245.)

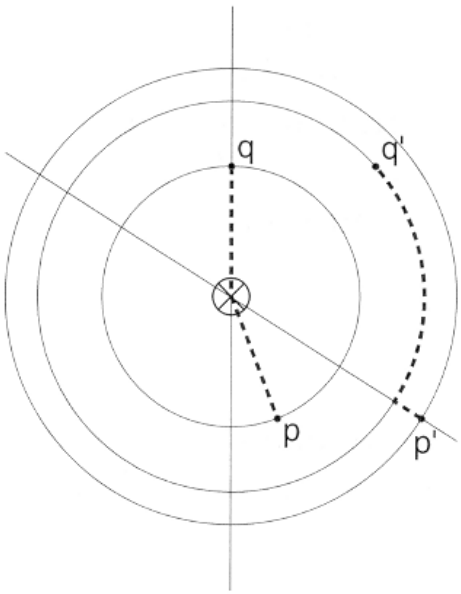
Ferner gilt wie im euklidischen Fall:
Zwei Voronoi-Regionen bzgl. derselben Punktmenge *S* und derselben streng konvexen Distanzfunktion haben höchstens eine gemeinsame Voronoi-Kante.

(Beweis: s. Klein 1997, S. 247 f.)

Es gibt andere Metriken, die nicht als konvexe Distanzfunktionen dargestellt werden können.
Beispiel:

die Karlsruhe-Metrik
(Entfernung im Straßensystem von Karlsruhe):

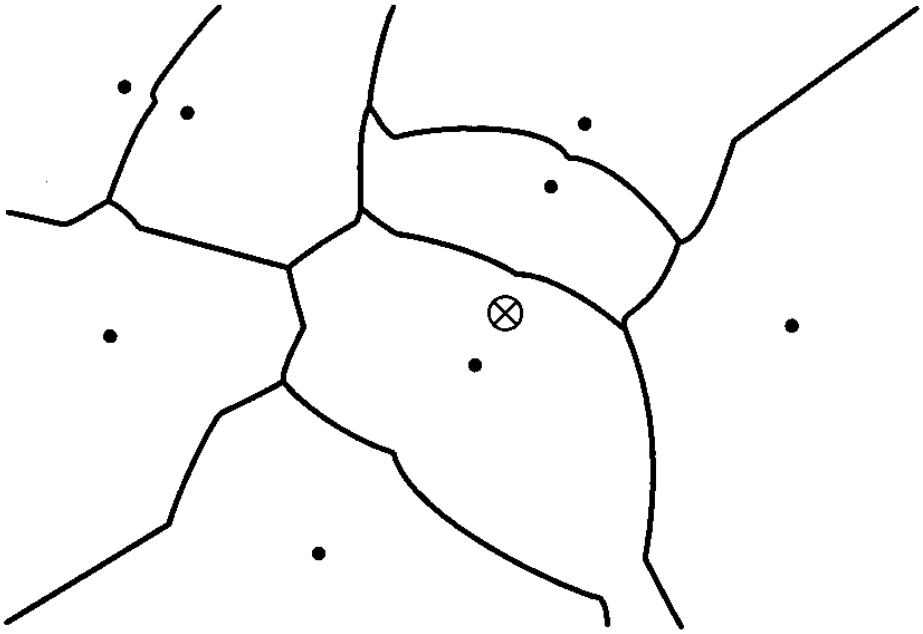




(aus Klein 1997)

die Karlsruhe-Metrik ist nicht translationsinvariant
(siehe Abb. rechts: $d(p', q') \neq d(p, q)$)

Ein Voronoi-Diagramm von 8 Punkten in der Karlsruhe-Metrik:



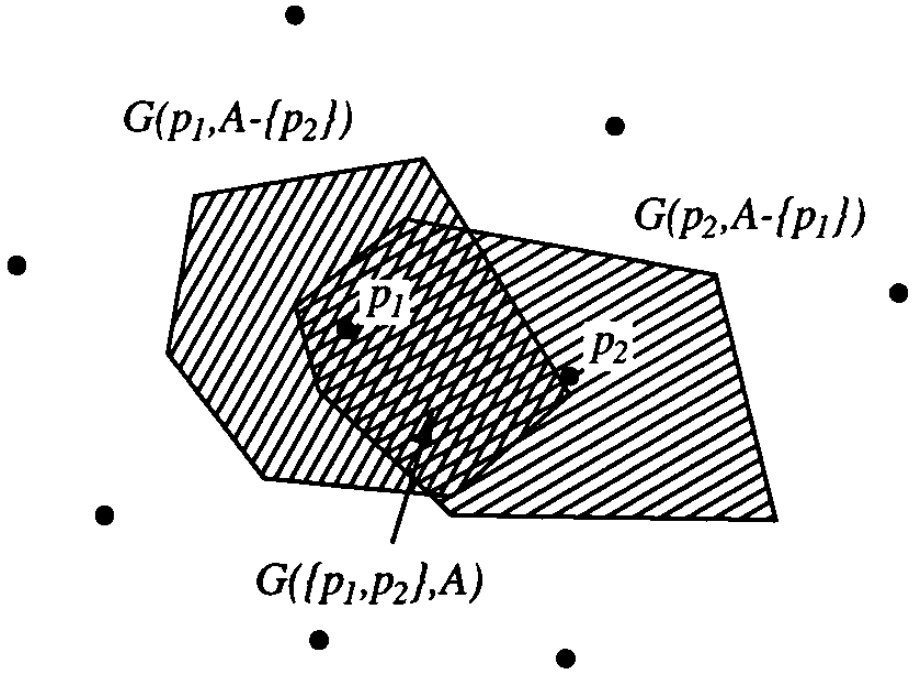
Voronoi-Diagramme k-ter Ordnung

Sei S Punktmenge und $B \subseteq S$.
Voronoi-Region $VR(B)$ = Menge aller Punkte, die *jedem* Punkt in B näher liegen als einem Punkt in $S \setminus B$.
Sei $|B| = k$, dann spricht man von VR der *Ordnung k*.

Für einelementiges B: herkömmliche VR.

Eine VR 2. Ordnung kann aus VR erster Ordnung konstruiert werden:

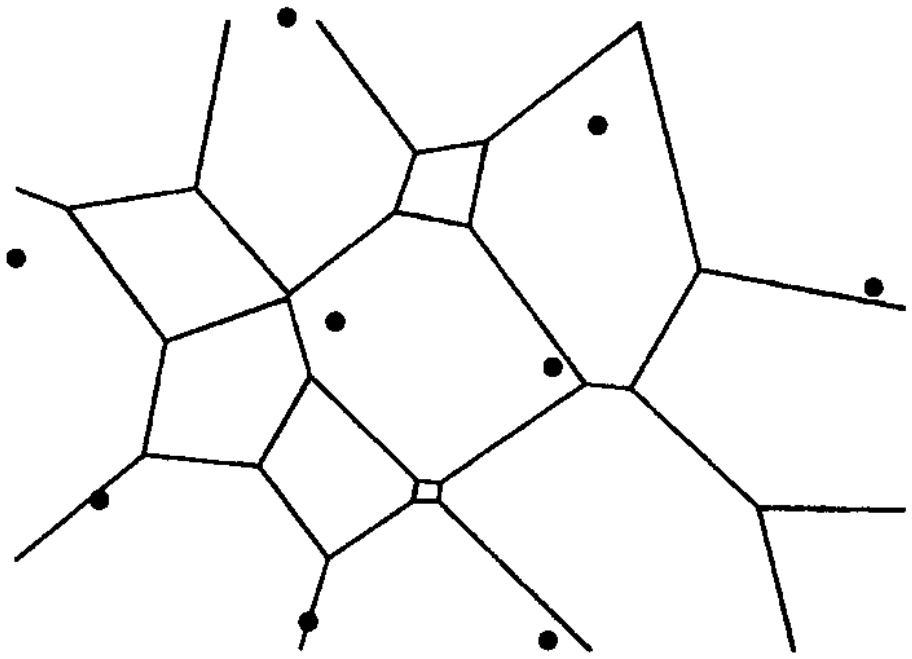
$$VR(\{p_1, p_2\}, S) = VR(p_1, S\setminus\{p_2\}) \cap VR(p_2, S\setminus\{p_1\}).$$



(*VR* heißt hier *G*, *S* heißt *A*) (aus Schmitt, Deussen & Kreeb 1996)

Alle Punkte in der doppelt schraffierten Fläche liegen näher zu p_1 und p_2 als zu irgendeinem anderen Punkt aus *S*.

Beispiel für ein Voronoi-Diagramm 2. Ordnung zu einer Punktmenge:



(aus Schmitt, Deussen & Kreeb 1996)

Das Power-Diagramm

"Power-Distanz" eines Punktes x zu einem Kreis (bzw. einer Kugel) $B_{c,p}$ (mit Mittelpunkt c und Radius p):

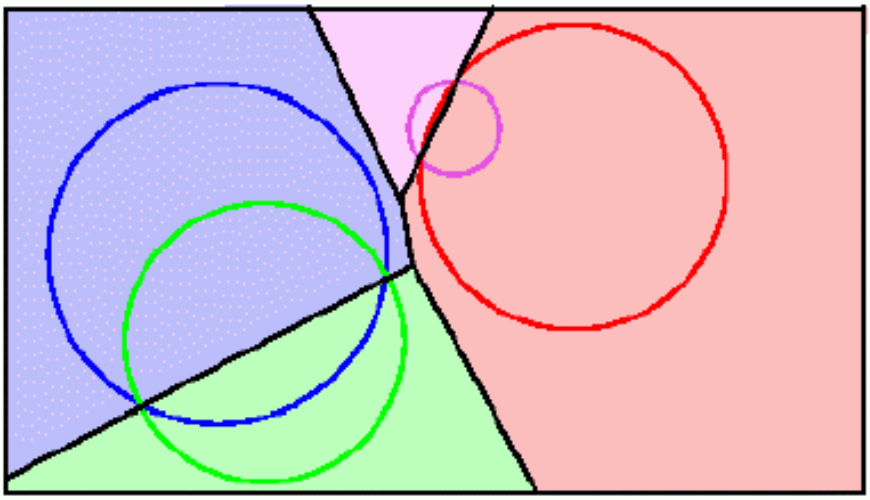
$$d_{\text{pow}}(x, B_{c,p}) = d^2(x, c) - p^2$$

- x im Inneren von $B_{c,p}$: Power-Distanz negativ
- x im Äußeren von $B_{c,p}$: Power-Distanz positiv

keine Metrik im math. strengen Sinne

Power-Diagramm:
gegeben: Menge S von *gewichteten* Punkten (c, p^2)
- ein Punkt c mit Gewicht p^2 wird durch einen Kreis $B_{c,p}$ mit Mittelpunkt c und Radius p dargestellt

jede Zelle eines Punktes (c, p^2) des Power-Diagramms besteht aus allen Punkten der Ebene (bzw. des Raumes), für die $B_{c,p}$ nächster Nachbar bzgl. der Power-Distanz ist.



Power-Diagramm von 4 gewichteten Punkten

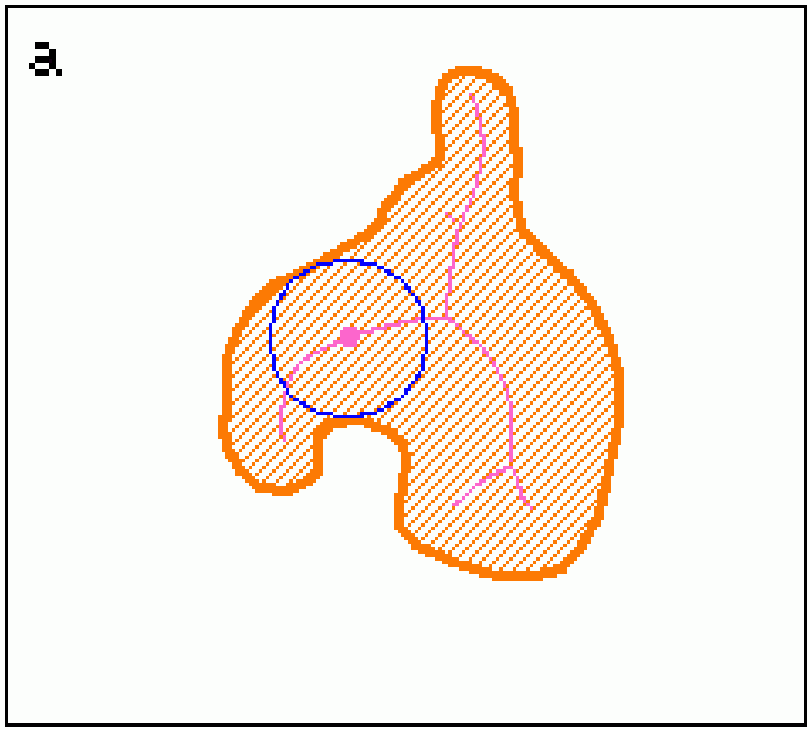
$\Rightarrow$ "gewichtete Form" des VD

Anwendung:
Rekonstruktion von Oberflächen aus 3D-Punktwolken (z.B. Laserscannerdaten)

"Power Crust"-Verfahren von Amenta, Choi & Kolluri (2001)

siehe **`http://www.cs.utexas.edu/users/amenta/powercrust`**
s.a. Referat von A. Christan,
`http://www-gs.informatik.tu-cottbus.de/~wwwgs/a3d_christan.pdf`

zusätzlich zur Oberfläche wird die Mittelachse des gescannten Objekts approximiert (Ort der Mittelpunkte aller maximalen Kugeln, die in das Objekt passen) (a)



prinzipieller Ablauf des Verfahrens:

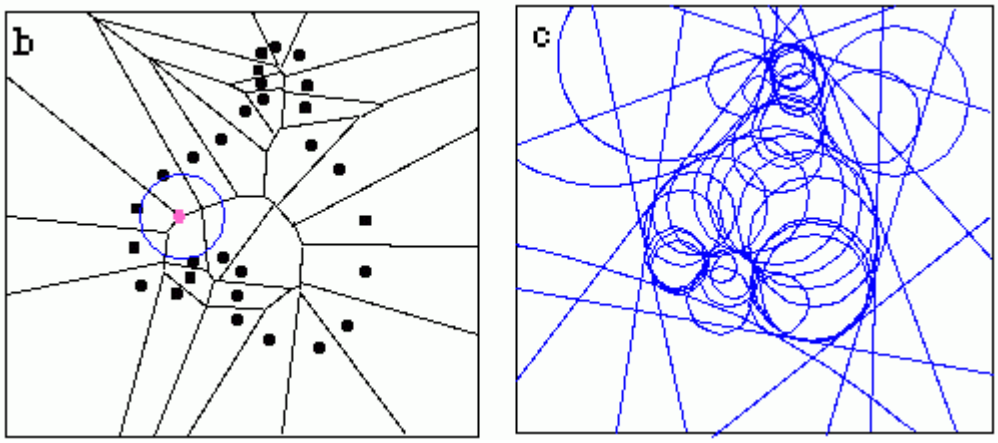
- Von der Menge S der gescannten Punkte wird das 3-D Voronoi-Diagramm konstruiert (b) (ein extremaler Knoten des VD mit maximaler Kugel ist eingezeichnet)

- bei dichter Punktwolke sind die Zellen des VD lang und dünn und senkrecht zur Oberfläche orientiert

- Teilmengen der Knoten des VD: innere und äußere *Pole*

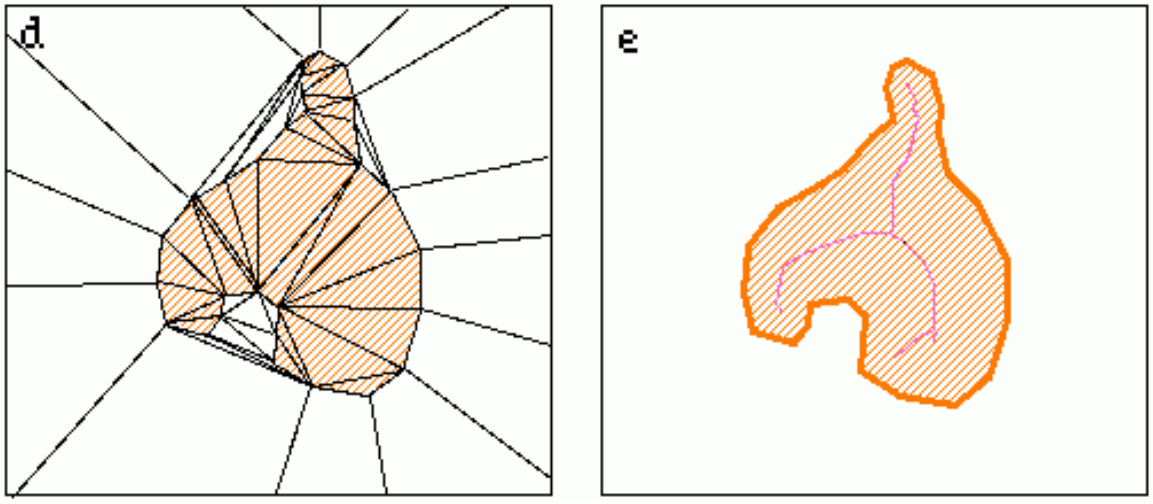
- Pole eines Punktes *s*: am weitesten entfernte Voronoi-Knoten im Inneren und im Äußeren der approximierten Menge
(Labelling-Algorithmus legt fest, welche Knoten des VD als "innen" und welche als "außen" angesehen werden)

Innere und äußere *Pol-Kugeln*: Kugeln um die Pole, welche die Punkte der Menge *S* gerade berühren (maximale Kugeln mit leerem Inneren) (c)



- die Radien der Pol-Kugeln definieren Gewichte der Pole
- die Menge der Mittelpunkte der Pol-Kugeln approximiert die Mittelachse der Menge *S*

- Approximation der Rücktransformation von der Mittelachse zur gesuchten Menge (Objektoberfläche): durch das Power-Diagramm der gewichteten Pole (d)



zu inneren Polen gehören innere Zellen
⇒ Grenze zwischen inneren und äußeren Zellen
= "*power crust*" = Approximation der Oberfläche (Output)

zusätzlicher Output: Verknüpfung der inneren Pole entspr. der Nachbarschaft ihrer Power-Diagramm-Zellen = Approximation der Mittelachse des Objektes (siehe (e))

Beispiel:

links: lasergenerierte Punktwolke
Mitte: mit dem Verfahren erzeugtes polygonales Modell ("wasserdicht")
rechts: approximierte Mittelachsenstruktur (Skelettierung)

(aus Amenta, Choi & Kolluri 2001, http s. oben)

Voronoi-Diagramme von Liniensegmenten

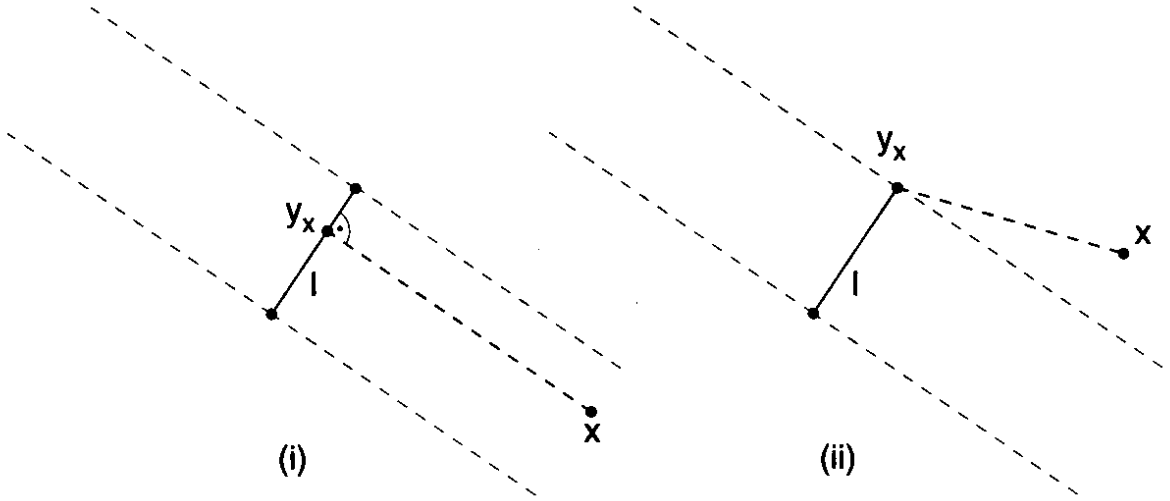
bisher nur punktförmige Objekte zugrundegelegt
→ Einflusszonen auch für andere Objekte denkbar

hier: *Liniensegmente* (Strecken)
Distanz sei gegeben durch die euklidische Metrik

Abstand Punkt x – Strecke s:

$$d(x, s) = \min \{ d(x, y) \mid y \in s \}$$

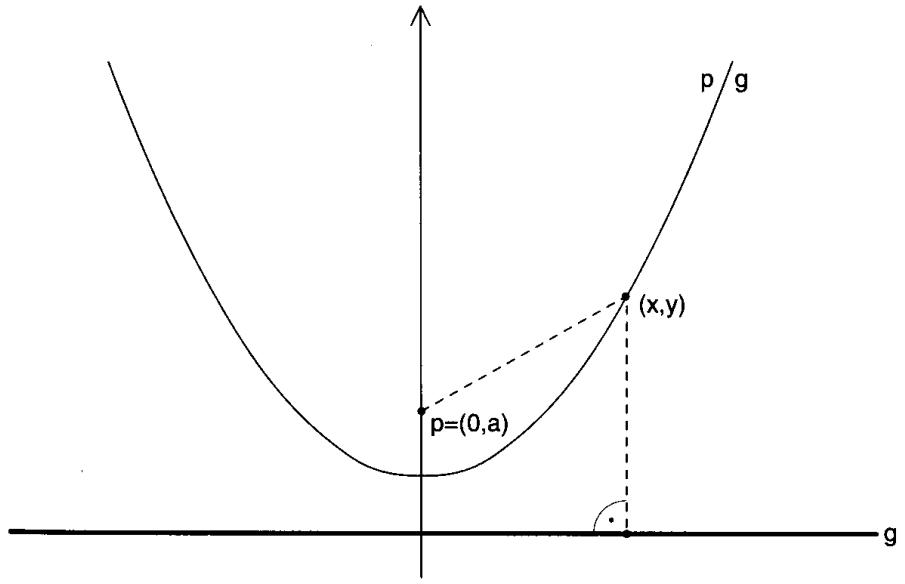
Minimum wird am x nächstgelegenen Punkt der Strecke angenommen – 2 Möglichkeiten:



Abstand Punkt – Gerade: nur Fall (i) tritt ein.

Bisektor von Punkt und Gerade:

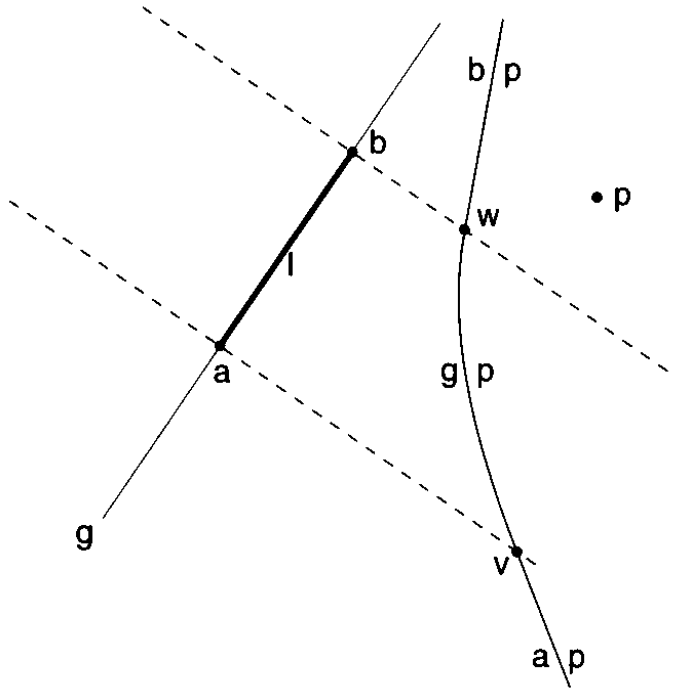
Parabel



hier spezielle Lage:
p: (0; *a*), *g*: *y*=0 $\Rightarrow$ Bisektor { (*x*, *y*) | $y = x^2/2a + a/2$ }.

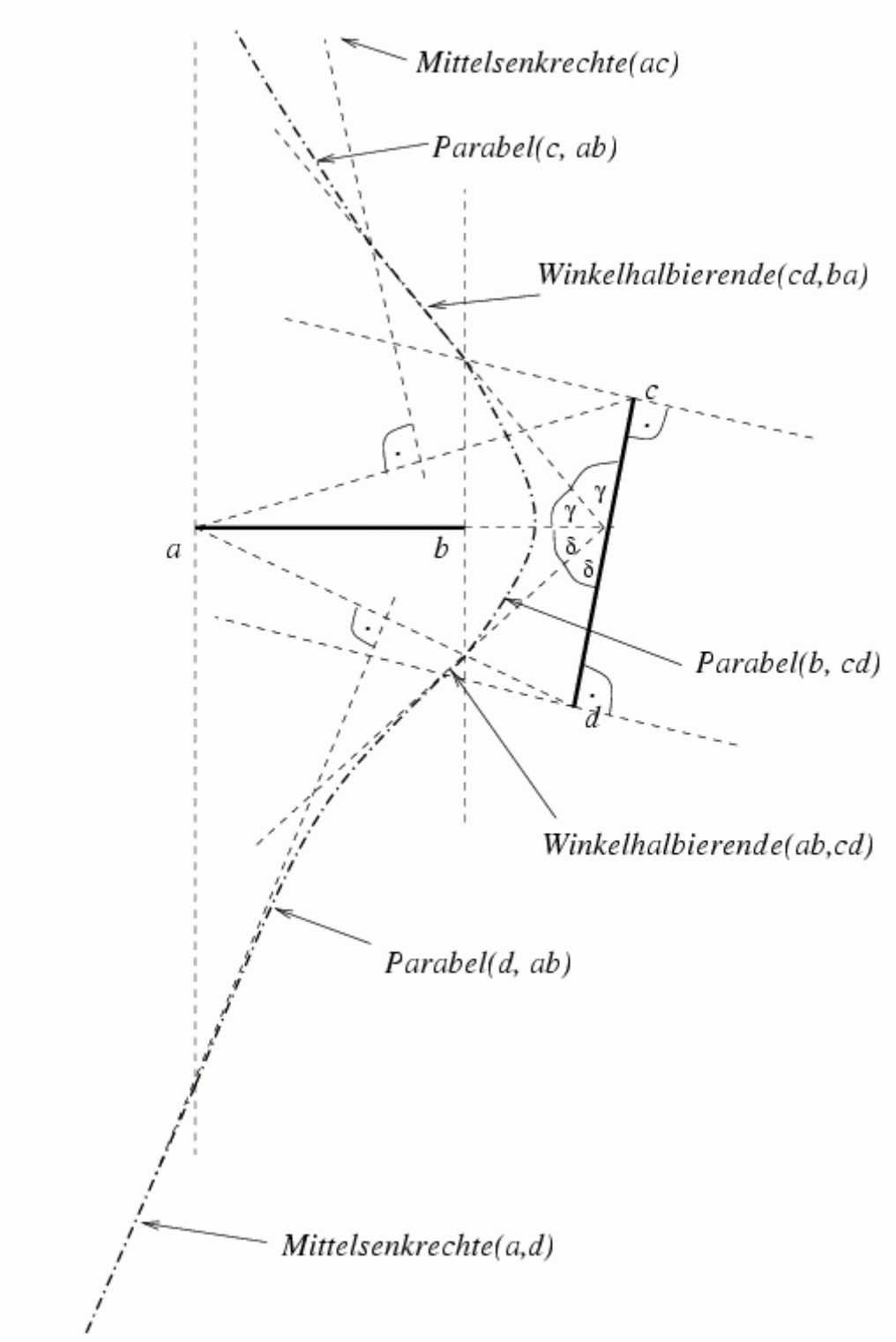
Bisektor Gerade – Gerade: Gerade (Winkelhalbierende)

Bisektor Punkt – Strecke:



1 Parabelstück und 2 Halbgeraden

Bisektor zweier Strecken:



(Keßler 1998)

Bisektor $B(s, s')$ zweier disjunkter Liniensegmente s, s'
setzt sich stetig zusammen aus max. 7 Teilstücken:
Parabeln, Winkelhalbierenden, und Mittelsenkrechten;

Analog: Halbebene $H(s, s') = \{p \in \mathbb{R}^2 : \ dist(p, s) < dist(p, s')\}$
Voronoi–Region $VR(s)$ eines Liniensegmentes $s \in S$:

$$VR(s) = \bigcap_{s' \in S \setminus \{s\}} H(s, s')$$

Die VR von Strecken sind in einem verallgemeinerten Sinne sternförmig:

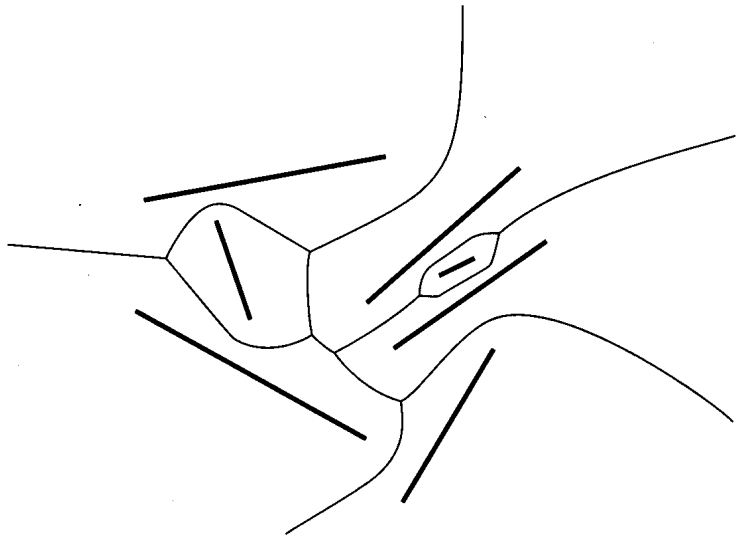
Sei *S* eine Menge von Strecken in der Ebene und $s \in S$.
Dann enthält die Voronoi-Region *VR*(*s*) mit jedem Punkt *x* auch die Strecke von *x* zum nächsten Punkt auf *s*.

Beweis: s. Klein 1997.

Insbesondere folgt: *VR*(*s*) ist zusammenhängend.

- Achtung: "Halbebene" *H*(*s*, *s*') und *VR*(*s*) i.allg. nicht mehr konvex!
- Zwei VR können mehr als eine gemeinsame Kante haben!

Beispiel eines Voronoi-Diagramms von 7 Strecken:



(Klein 1997)

Es gilt:

- Das Voronoi-Diagramm von *n* disjunkten Strecken in der Ebene hat *O*(*n*) viele Knoten und Kanten.
- Jede Kante besteht aus *O*(1) vielen Stücken.
- Der Rand einer Voronoi-Region enthält im Mittel höchstens 6 Kanten.

(s. Klein 1997)

Berechnung: mit Plane–Sweep analog zu VD von Punkten

wobei die Sweep–Status–Struktur ($\rightarrow$ Wellenfront) statt Parabelbögen $P(p, q, r)$ etwas kompliziertere Strukturen aus Parabeln, Winkelhalbierenden und Mittelsenkrechten verwalten muß

$\rightarrow$ VD für n Liniensegmente kann in Zeit $O(n \log n)$ berechnet werden.

Anwendung des VD von Strecken bei der Bewegungsplanung für Roboter

- kreisförmiger Roboter (Position = Koordinaten des Mittelpunktes)
- soll von gegebenem Startpunkt zu gegebenem Zielpunkt bewegt werden
- dabei sind Hindernisse zu berücksichtigen (Liniensegmente): Zu keinem Zeitpunkt darf das Innere des Kreises eine der Strecken der Szene schneiden

scheinbar sind unendlich viele mögliche Wege zu testen

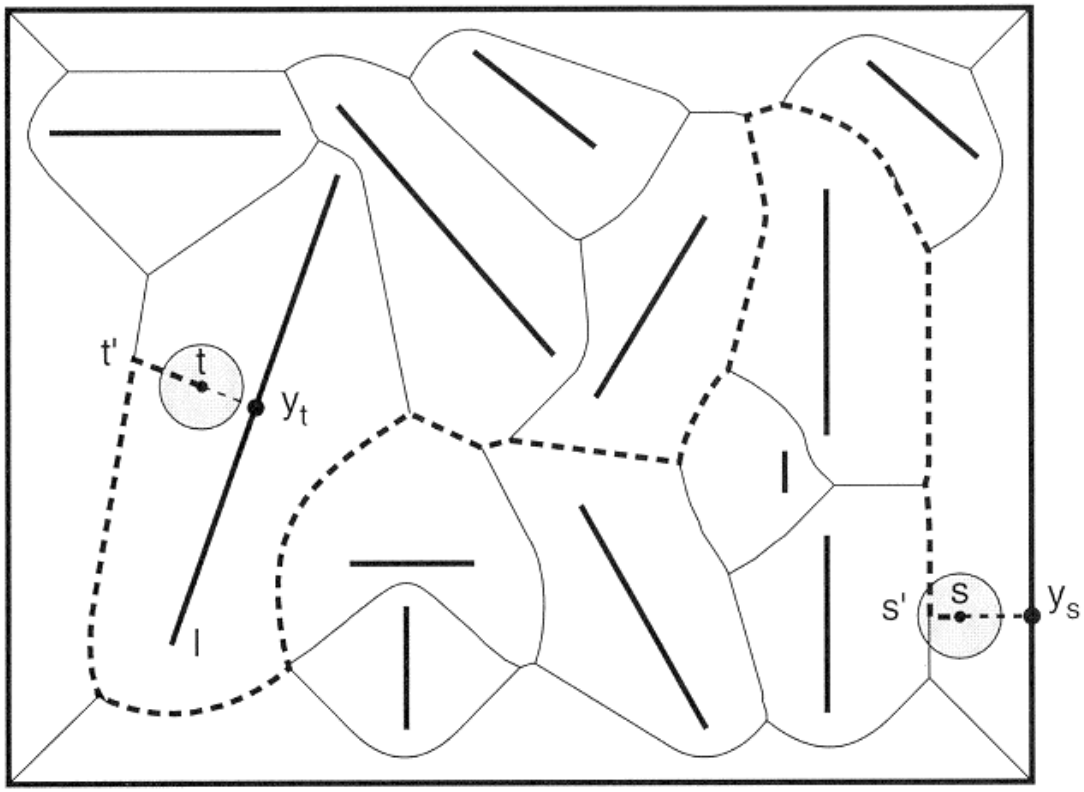
- Idee: in jeder Position die Sicherheitsabstände zu den Strecken der Szene maximieren

$\Rightarrow$ Bewegung längs der Kanten des von den Strecken der Szene definierten VD
(damit "endliches Problem")

Bewegungsplanung für Kreisscheibe R (Roboter) mit Radius r
in einer Szene $S \subset \mathbb{R}^2$ aus n Liniensegmenten (Hindernissen)
vom Startpunkt $A \in \mathbb{R}^2$ zum Ziel $B \in \mathbb{R}^2$.

Punkt $x \in \mathbb{R}^2$ heißt $kollisionsfrei :\Leftrightarrow dist(x, y) \geq r \; \forall y \in s \; \forall s \in S$
Bewegung = kollisionsfreier Weg W (alle Punkte in W kollisionsfrei)
Beobachtung: Falls ein kollisionsfreier Weg von A nach B existiert,
dann existiert ein kollisionsfreier Weg über Voronoikanten
= der Weg, der maximalen Abstand zu den Segmenten einhält

präziser:
Genau dann kann sich der Roboter kollisionsfrei von *A* nach *B* bewegen, wenn sein Radius *r* die Abstände von *A* zum nächsten Punkt y_A der Szene und von *B* zum nächsten Punkt y_B der Szene nicht überschreitet und wenn es eine kollisionsfreie Bewegung von *A*' nach *B*' längs der Kanten des VD *V*(*S*) gibt, wobei *A*' der Schnittpunkt des Strahls von y_A durch *A* mit dem VD ist (*B*' analog).



(aus Klein 1997; *A* heißt hier *s*, *B* heißt *t*.)

Algorithmus:

(0) Berechne $VD(S)$ $O(n \log n)$

(1) teste, ob A und B sichere Positionen sind $O(n)$

(2) Eliminiere alle (Teil–)Voronoikanten (ggf. aufspalten) e
mit $\exists x \in e$: x nicht kollisionsfrei $\rightarrow VD'(S)$ $O(n)$

(3) bestimme $VR(s)$, in dem A liegt, analog $VR(s')$ für B $O(\log n)$

(4) bestimme Voronoikante $e \subset \partial VR(s)$, die das Lot von A auf s schneidet in Punkt A' (analog für $B \rightarrow B'$) $O(n)$

Beobachtung: A kollisionsfrei $\rightarrow A'$ und $\overline{AA'}$ kollisionsfrei

(5) konstruiere kollisionsfreien Weg W' in VD' von A' nach B'
z.B. mit DFS $O(n)$

(6) falls ex.: Ausgabe $W = \overline{AA'} \cup W' \cup \overline{B'B}$

Laufzeit: $O(n \log n)$, Platz $O(n)$

Folgerung (vgl. Klein 1997):

Ist das Voronoi-Diagramm der *n* Strecken der Szene vorhanden, lässt sich für einen beliebigen Roboter-Radius *r* und beliebige Punkte *A* und *B* in der Zeit *O*(*n*) ein kollisionsfreier Weg von *A* nach *B* bestimmen oder aber feststellen, dass es keinen gibt.

Verallgemeinerung auf nicht kreisförmige Roboter:

- der Roboter sei nicht rotierfähig (nur Translationen erlaubt)
- die Form werde durch eine konvexe Menge *C* beschrieben

$\Rightarrow$ lege für das VD der Strecken die konvexe Distanzfunktion $d_{C'}$ zugrunde, wobei *C*' die Spiegelung von *C* am Referenzpunkt ist; Algorithmus lässt sich dann übertragen.