4.1. Voronoidiagramme

Theorie

Wir wollen in diesem Abschnitt einer Menge von Punkten in der Ebene eine planare Unterteilung zuweisen, so dass jede Facette alle Punkte der Ebene enthält, die einem Punkt in der Menge "näher" als allen anderen Punkten in der Menge sind. Diese Unterteilung heißt **Voronoidiagramm**. Als Abstandsmaß für die Nähe verwenden wir den Euklidischen Abstand.

$$\text{dist}(p.q):$$
$$= \sqrt{(p_x - q_x)^2 + \left(p_y - q_y\right)^2}$$

Definition 4.1: Sei $P := \{p_1 , \ldots p_n\}$ eine Menge n verschiedener Punkte in der Ebene. Eine planare Unterteilung der Ebene mit n Facetten $V(p_i)$ mit

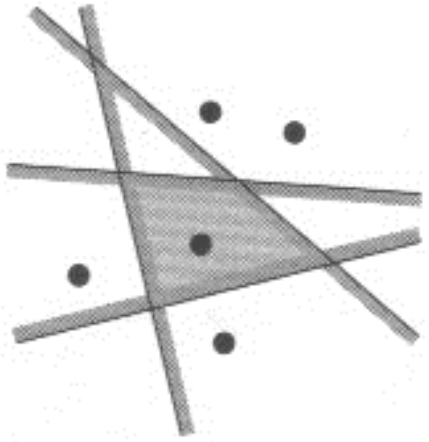
$$q \in V(p_i) \Leftrightarrow dist(q, p_i) < dist\left(q, p_j\right) \forall j \neq i$$

heißt **Voronoidiagramm *V*or(*P*) von *P*.**

4.1 Voronoidiagramme

Da die Menge der Punkte mit gleichem Abstand zu p_i und p_j eine Gerade ist, ergeben sich die Voronoizellen als Schnitte von offenen Halbräumen. Sei l_{ij} die auf $\overline{pq}$ in der Mitte senkrecht stehende Gerade. Ferner sei $h(p,q)$ die offene Halbebene, die p enthält und von l berandet ist.

Bem 4.2: $V(p_i) = \cap_{1 \le j \le n, j \ne i} h(p_i, p_j)$. $V(p_i)$ ist eine offene, konvexe, polygonale berandete Region.



4.1 Voronoidiagramme

Um ein wenig mehr über die Voronoizellen zu erfahren, brauchen wir ein paar geometrische Überlegungen.

Theorem 4.3.: Sei P eine Menge *n* verschiedener Punkte der Ebene. Wenn alle Punkte auf einer Geraden liegen, besteht Vor(*P*) aus *n*-1 parallelen Geraden. Ansonsten ist Vor(*P*) (bzgl. der Kanten und Ecken) ein verbundener Graph und die Kanten sind Liniensegmente oder Halbgeraden.

Beweis: Der erste Teil ist einfache Geometrie. Daher untersuchen wir den zweiten Teil. Wir wissen schon, dass die Kanten Teile von Geraden sind. Dazu sei eine Kante *e* in Vor(*P*) eine Gerade. Ferner seien V(p_i) und V(p_j) durch *e* berandet. Sei nun p_k nicht zu p_i und p_j kollinear. Dann schneidet die Grenze zwischen V(p_i) und V(p_k) sicher *e* und daher kann e nicht Kante in Vor(*P*) sein! Daraus folgt auch, dass Vor(*P*) (bzgl. der Kanten) zusammenhängt, denn alle Facetten sind konvex, so dass eine trennende Facette eine von zwei parallelen Geraden berandete Facette sein müsste. QED

4.1 Voronoidiagramme

Für die Konstruktion und vor allem Analyse eines Algorithmus ist die Komplexität von Vor(*P*) in Abhängigkeit der Zahl *n* der Punkte von großer Bedeutung.

Theorem 4.4.: Für $n > 3$ gibt es maximal $2n$-5 Ecken und $3n$-6 Kanten in Vor(*P*).

Beweis: Im Fall kollinearer Punkte reicht 4.3. Ansonsten nutzen wir die Eulerformel $v - e + f = 2$ mit *v* Ecken, *e* Kanten und *f* Facetten, wobei wir aber zu *v* noch die Ecke im Unendlichen hinzuzählen müssen, da wir Halbgeraden als Kante haben. Nun hat jede Kante genau 2 Ecken und jede Ecke mindestens drei Kanten nach der Konstruktion der Voronoizellen. Es folgt $(v + 1) - e + n = 2$, $2e >= 3(v+1)$,

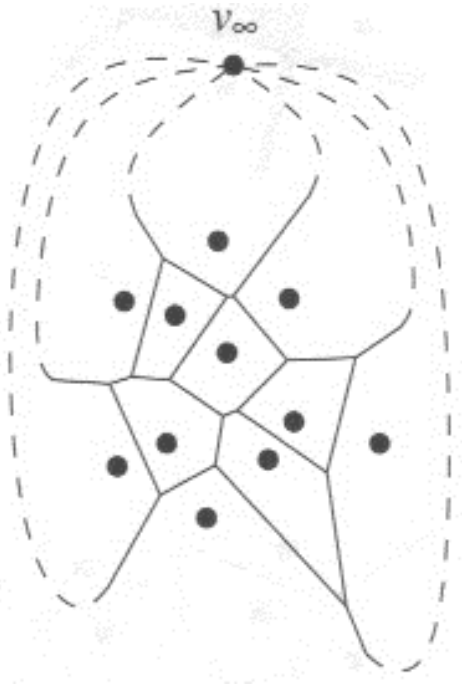
also

$$6 = 3(v + 1) - 3e + 3n <= 2e - 3e + 3n = -e + 3n \quad => \quad e <= 3n - 6.$$

$$2 = (v+1) - e + n >= (v+1) - 3n + 6 + n = v - 2n + 7 \quad => \quad v <= 2n - 5$$

QED

4.1 Voronoidiagramme



4.1 Voronoidiagramme

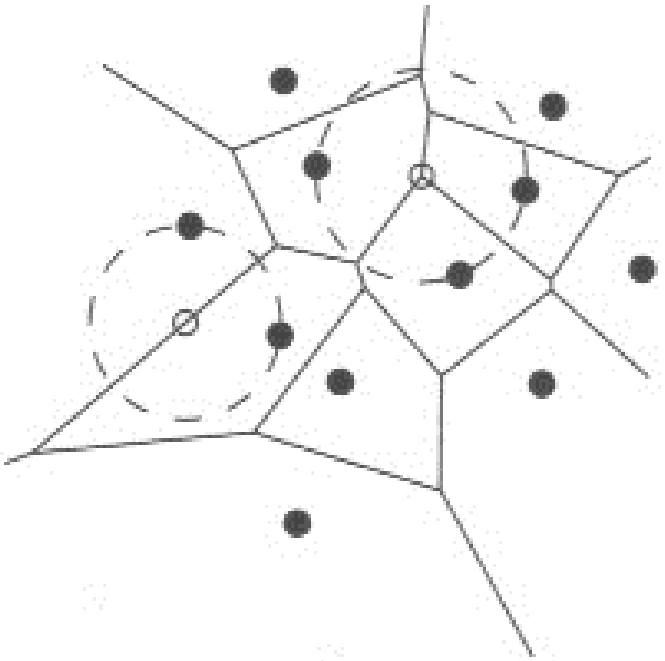
Das letzte Theorem ist besonders nützlich, da es ½ n(n – 1) trennende Geraden gibt, aber nur wenige, eben maximal 3*n* – 6, liefern auch Kanten im Voronoidiagramm. Für einen effizienten Algorithmus dürfte ein Kriterium, wann eine trennende Gerade zu einer Kante beiträgt und welche Schnitte dieser Kanten Ecken in Vor(*P*) sind, sehr nützlich sein.

Theorem 4.5.: Für ein Voronoidiagramm einer Menge *P* von Punkten gilt:

(i) Ein Punkt *q* der Ebene ist Ecke von Vor(*P*) *gdw* der größte leere Kreis $C_p(q)$ um *q* durch drei oder mehr Punkte aus *P* führt.

(ii) Eine trennende Gerade *l* zu zwei verschiedenen Punkten p_i, p_j in P definiert eine Kante in Vor(*P*) gdw es einen Punkt *q* auf *l* gibt, dessen größter leerer Kreis $C_p(q)$ nur p_i und p_j auf seinem Rand hat.

4.1 Voronoidiagramme



4.1 Voronoidiagramme

Beweis: (i) Sei q ein Punkt mit einem solchen Kreis durch p_i , p_j, p_k. Dann liegt q auf dem Rand der Voronoizellen zu p_i, p_j, p_k, da das Innere leer ist. Folglich ist q Ecke des Voronoidiagrammes. Andererseits ist jede Ecke von Vor(P) inzident zu mindestens drei Zellen $V(p_i)$, $V(p_j)$ und $V(p_k)$ aus P und somit gleich weit von diesen entfernt, wodurch sich ein solcher Kreis ergibt.

(ii) Sei q ein Punkt aus dem 2. Teil des Theorems. Dann gilt

$$\text{dist}(q, p_i) = \text{dist}(q, p_j) < \text{dist}(q, p_k) \quad \text{für alle } 1 < k < n,\ k <> i, j.$$

Dann muss q auf dem Rand von Voronoizellen liegen und kann nach (i) keine Ecke sein, muss also auf einer Kante liegen und diese wird durch l gebildet. Umgekehrt, wenn die trennende Gerade l eine Kante von Vor(P) enthält, ist jeder innere Punkt der Kante ein q mit den gesuchten Eigenschaften. QED

4.1 Voronoidiagramme

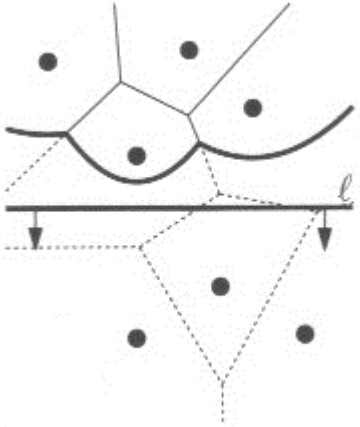
Algorithmus und Analyse

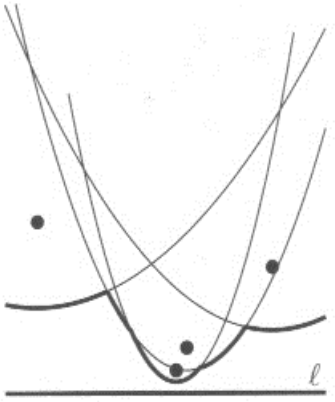
Ein naiver Ansatz zu den Voronoidiagrammen führt offensichtlich auf $O(n^2)$ oder noch mehr Aufwand. Wir wollen daher einmal wieder auf unsere erprobte **plane sweep** Technik zurückgreifen.

Die direkte Anwendung des Prinzips bedeutet, dass wir eine Gerade über die Ebene ziehen und hinter der Geraden das Voronoidiagramm aufbauen, wobei wir uns Information über die Situation auf der Geraden (Zustandsstruktur) merken. Voronoidiagramm und Zustand ändern sich nur an endlich vielen Punkten, die von der sweep line überschritten werden, den event points.

4.1 Voronoidiagramme

Leider ist es diesmal nicht ganz so einfach, da das Voronoidiagramm oberhalb der (waagrechten) sweep line auch von Punkten unterhalb der sweep line abhängt. Dies gilt aber nur für den Bereich, der außerhalb einer Vereinigung von Parabeln liegt! Diese Parabeln sind die Punkte, die von einem erreichten Punkt p und der sweep line l den gleichen Abstand haben!





Den Rand dieser Vereinigung von Parabeln nennen wir **Strand.**

4.1 Voronoidiagramme

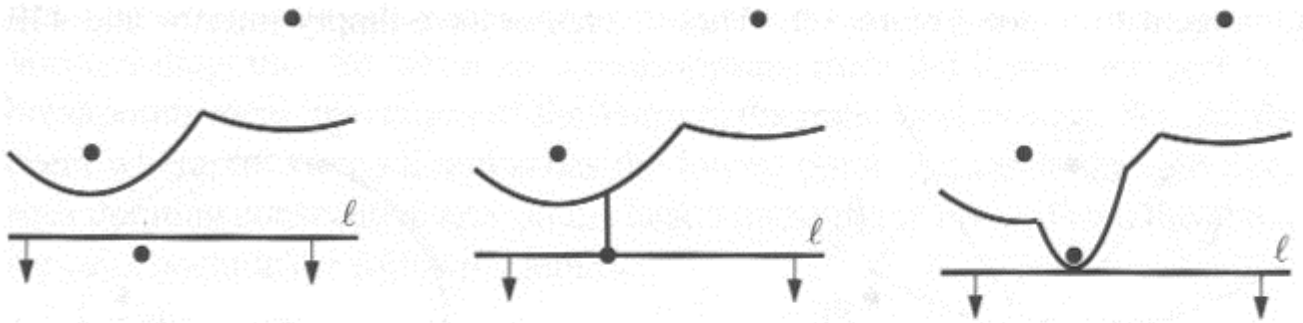
Für jede x-Koordinate läuft der Strand durch den niedrigsten Punkt aller Parabeln.

Bemerkung 4.6.: Der Strand ist *x*-monoton, d. h. jede vertikale Gerade schneidet den Strand in genau einem Punkt.

Es ist leicht zu sehen, dass eine Parabel zu mehreren Bögen am Strand beitragen kann. Ferner liegen die Schnitte der Parabeln genau auf den Kanten des Voronoidiagramms. Als Zustandsstruktur wählen wir daher den Strand! Bevor wir die Datenstruktur festlegen, betrachten wir die auftretenden Ereignisse (Events).

4.1 Voronoidiagramme

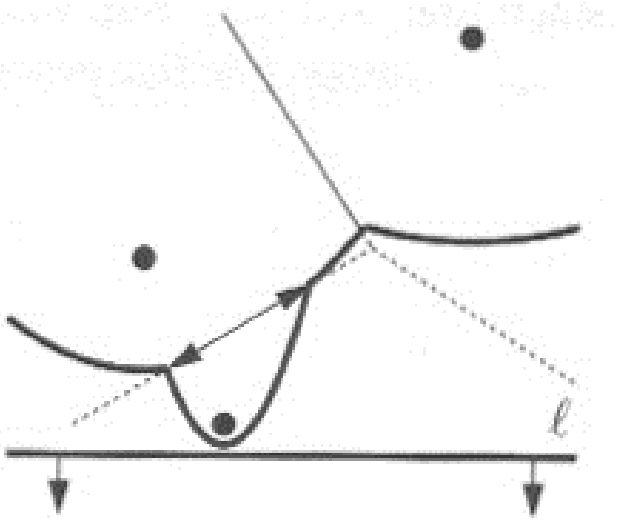
Ein neuer Bogen entsteht, wenn die sweep line einen neuen Punkt aus P erreicht.



A new arc appears on the beach line because a site is encountered

4.1 Voronoidiagramme

Der neue Bogen verursacht zwei Parabelschnitte, die jeweils eine Kante von Vor(P) erzeugen.



4.1 Voronoidiagramme

Dies ist der einzige Weg für das Entstehen neuer Bögen. Das Erreichen eines neuen Punktes durch die sweep line l nennen wir **Punktereignis** (**site event**).

Lemma 4.7. Der einzige Fall, der eine neue Parabel am Strand erzeugt, ist ein Punktereignis.

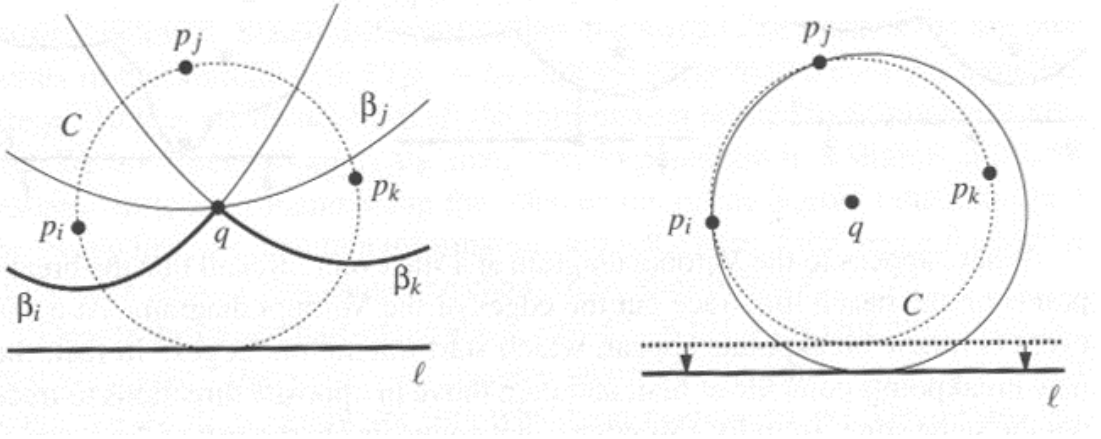
Beweis: Wir nehmen an, dass eine bereits existierende Parabel β_j zu p_j den Strand schneidet, während die sweep line weiterbewegt wird. Im ersten Fall schneidet β_j eine Parabel β_i im Inneren. Sei l_y die y-Koordinate, wo sich β_j und β_i gerade berühren. Dann sind die beiden Parabeln tangential, wo sie sich berühren und es gibt genau einen Schnittpunkt. Die Formel für β_j mit $p_j = (p_{j,x}, p_{j,y})$ lautet:

$$\beta_j : y = \frac{1}{2(p_{j,y} - l_y)}(x^2 - 2p_{j,x}x + p_{j,x}^2 + p_{j,y}^2 - l_y^2)$$

und die Formel für β_i ist analog. Wenn $p_{j,y}$ und $p_{i,y}$ größer als l_y sind (beide Punkte oberhalb der sweep line bei l_y), kann es nicht nur einen Schnittpunkt von β_i und β_j geben.

4.1 Voronoidiagramme

(2) Im zweiten Fall tritt β_j im Schnittpunkt zweier Bögen β_i , β_k auf. Sei q der Schnittpunkt von β_i und β_k , an dem β_j im Strand auftritt. Ferner sei β_i links von q und β_k rechts von q.



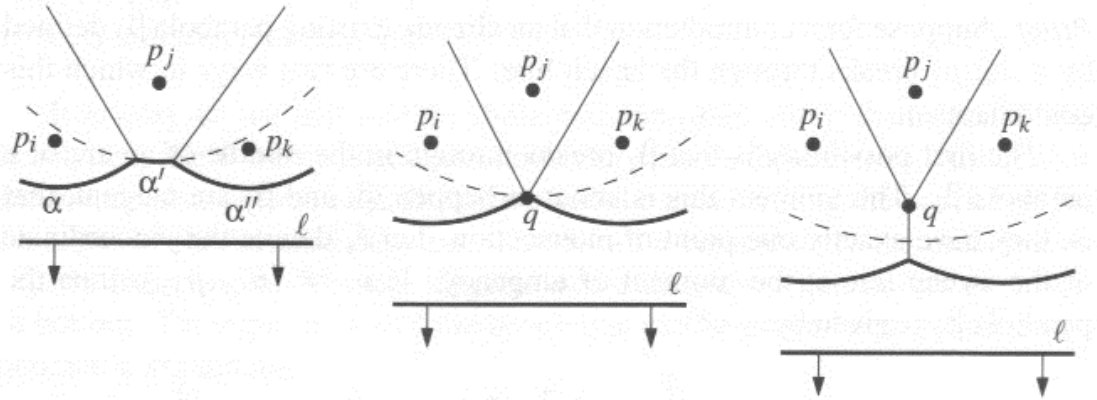
The situation when β_j would appear on the beach line, and the circle when the sweep line has proceeded

Dann gibt es einen Kreis durch p_i, p_j und p_k, der *l* berührt. Wegen der relativen Lage auf dem Kreis liegt p_k bei einer Bewegung von l nach unten in dem Kreis durch p_i und p_j, der l berührt. Ferner liegt p_i innerhalb des Kreises durch p_j und p_k, der l berührt. Daher erzeugt β_j keinen Bogen am Strand. QED

4.1 Voronoidiagramme

Bemerkung 4.8. Der Strand besteht aus max. $2n - 1$ parabolischen Bögen, denn jedes Punktereignis erzeugt einen neuen Bogen und kann einen Bogen in zwei Bögen teilen.

Neben den Punktereignissen kann es noch passieren, dass ein Bogen zu einem Punkt zusammengezogen wird und dann verschwindet.



An arc disappears from the beach line

4.1 Voronoidiagramme

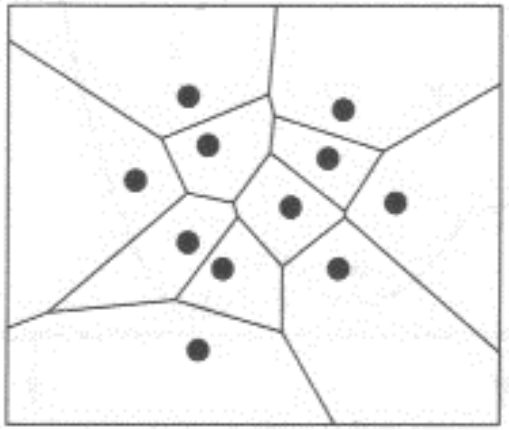
Wenn wir die drei beteiligten Bögen wie im Bild β, β', β'' nennen, so lässt sich analog zu 4.7 zeigen, dass β und β'' nicht zur gleichen Parabel gehören. Daher liegt der Punkt *q*, an dem β' verschwindet, im Zentrum des Kreises durch drei Punkte p_i, p_j, p_k in *P*. Ferner liegt der tiefste Punkt dieses Kreises auf *l* ! Wenn *l* durch den tiefsten Punkt eines Kreises durch die erzeugenden Punkte von drei im Strand aufeinander folgenden Bögen läuft, sprechen wir von einem Kreisereignis (circle event). Es gilt also:

Lemma 5.9.: Ein Bogen kann nur durch ein Kreisereignis vom Strand verschwinden.

4.1 Voronoidiagramme

Für unseren Algorithmus wählen wir folgende Datenstrukturen für das Voronoidiagramm, den Strand *T* und die Ereignissschlange *Q*:

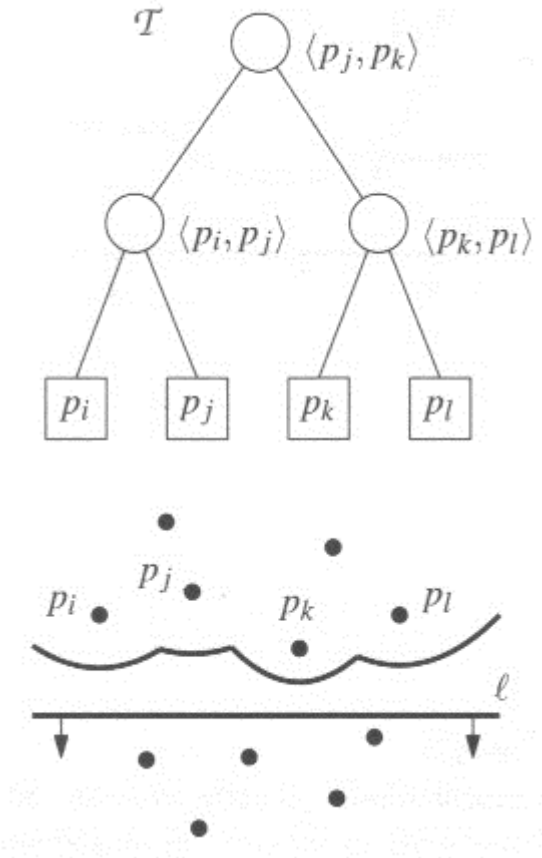
- Das im Bau befindliche Voronoidiagramm repräsentieren wir durch eine doppelt verknüpfte Kantenliste, wobei wir ein ausreichend großes berandetes Rechteck (bounding box) nutzen, um dem Problem von Halbgeraden unter den Kanten zu entgehen.



4.1 Voronoidiagramme

- Den Strand, also die Zustandsstruktur unserer sweep line, repräsentieren wir durch einen balancierten binären Suchbaum T. Die Blätter repräsentieren die Bögen am Strand von links nach rechts und enthalten einen Zeiger auf den erzeugenden Punkt in *P*. Die inneren Knoten bezeichnen die Parabelschnitte und können genau wie die Bögen von links nach rechts geordnet werden. Im Knoten legen wir Zeiger zu den beiden erzeugenden Punkten ab. Wenn wir in einem Punktereignis den Bogen suchen, in dem der neue Punkt liegt, bestimmen wir aus den beiden Punkten und der *y*-Koordinate der sweep line die Position des Schnittpunktes der Parabeln in konstanter Zeit und können so den richtigen Bogen - also das richtige Blatt im binären Baum in $O(\log n)$ ermitteln.

4.1 Voronoidiagramme



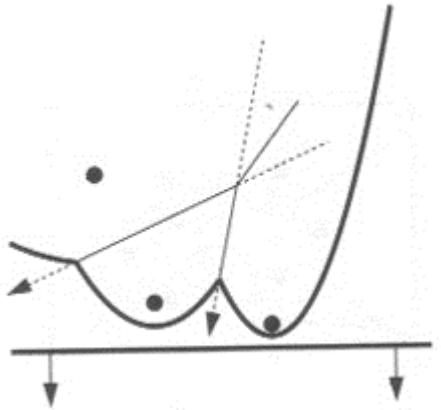
4.1 Voronoidiagramme

- *(Fortsetzung)* Wir speichern in jedem Blatt auch einen Zeiger auf das Kreisereignis in der Eventschlange Q, indem der Bogen zum Blatt verschwindet. Gibt es dieses Ereignis (noch) nicht in Q, so ist der Zeiger null. Ferner haben die inneren Knoten in T einen Zeiger auf die zugehörige Kante in Vor(P).

- Die Ereignissschlange (event queue) Q wird als Prioritätsschlange (priority queue) realisiert, wobei die Priorität die y-Koordinate ist. Die Punktereignisse erhalten die y-Koordinate des Punktes und die Kreisereignisse die y-Koordinate des untersten Punktes im Kreis.

4.1 Voronoidiagramme

Die Punktereignisse sind alle zu Beginn des Algorithmus bekannt. Die Kreisereignisse müssen dagegen erzeugt werden. Für je drei aufeinander folgende Bögen auf dem Strand, die ein Kreisereignis definieren können, erzeugt unser Algorithmus ein entsprechendes Ereignis. Dabei ist zu beachten, dass die Schnittpunkte der Bögen auseinander driften können, so dass sie sich nicht treffen. In diesem Fall ist kein Kreisereignis zu erzeugen.



4.1 Voronoidiagramme

Ferner kann es sein, dass ein Kreisereignis nicht eintritt, weil ein neuer Bogen durch ein Punktereignis entsteht und das Kreisereignis verhindert. Dann ist das Kreisereignis (mittels des Zeigers vom Blatt zum Kreisereignis) aus *Q* zu entfernen.

4.1 Voronoidiagramme

Der Algorithmus geht nun die Ereignisse durch. Bei einem Punktereignis können bis zu drei neue Tripel von Bögen entstehen. Im ersten Tripel ist der neue Bogen links, im zweiten in der Mitte und im dritten rechts. Da im zweiten Tripel die beiden äußeren Bögen von der gleichen Parabel stammen, gibt es hier kein Kreisereignis. Für die beiden anderen Tripel ist zu testen, ob die Bögenschnitte aufeinander zu oder auseinander streben. Ferner ist für die verschwindenden Tripel das zugehörige Kreisereignis zu löschen, falls eines existiert.

4.1 Voronoidiagramme

Lemma 4.10. Jede Ecke des Voronoidiagramms wird durch ein Kreisereignis entdeckt.

Beweis: Zu jeder Ecke q gibt es nach 4.5 drei Punkte p_i, p_j, p_k in P auf einem Kreis $C_p(q)$ ohne Punkte aus P im Inneren. Ferner sei O.E. p_i, p_j, p_k die Reihenfolge der Punkte im Uhrzeigersinn auf dem Kreis: Wir betrachten die sweep line kurz (infinitesimal) vor dem Erreichen des untersten Punktes von C. Da es keine Punkte aus P im Inneren von $C_p(q)$ gibt, existiert ein Kreis durch p_i und p_j tangential zu l und benachbarte Bögen β_i, β_j am Strand. Analog gibt es einen solchen Kreis durch p_j und p_k und benachbarte Bögen β_j', β_k. Ferner gilt $\beta_j = \beta_j'$, da kein weiterer Bogen dazwischen liegen kann. Also wird die Ecke durch das Kreisereignis zu p_i, p_j, p_k entdeckt. QED

4.1 Voronoidiagramme

Wir können den Algorithmus nun notieren, wobei wir beachten, dass T am Ende nicht leer ist, sondern die verbleibenden Bogenschnitte (innere Knoten) zu den Halbgeraden gehören, die für $y = -\infty$ erst enden.

Algorithm VORONOIDIAGRAM(P)
Input. A set $P := \{p_1, \ldots, p_n\}$ of point sites in the plane.
Output. The Voronoi diagram Vor(P) given inside a bounding box in a doubly-connected edge list $\mathcal{D}$.

1. Initialize the event queue Q with all site events, initialize an empty status structure $\mathcal{T}$ and an empty doubly-connected edge list $\mathcal{D}$.
2. **while** Q is not empty
3. **do** Remove the event with largest y-coordinate from Q.
4. **if** the event is a site event, occurring at site p_i
5. **then** HANDLESITEEVENT(p_i)
6. **else** HANDLECIRCLEEVENT(γ), where γ is the leaf of $\mathcal{T}$ representing the arc that will disappear
7. The internal nodes still present in $\mathcal{T}$ correspond to the half-infinite edges of the Voronoi diagram. Compute a bounding box that contains all vertices of the Voronoi diagram in its interior, and attach the half-infinite edges to the bounding box by updating the doubly-connected edge list appropriately.
8. Traverse the half-edges of the doubly-connected edge list to add the cell records and the pointers to and from them.

4.1 Voronoidiagramme

Die Punktereignisse (site events) behandelt HANDLESITEEVENT.

HANDLESITEEVENT(p_i)

1. If $\mathcal{T}$ is empty, insert p_i into it (so that $\mathcal{T}$ consists of a single leaf storing p_i) and return. Otherwise, continue with steps 2– 5.
2. Search in $\mathcal{T}$ for the arc α vertically above p_i. If the leaf representing α has a pointer to a circle event in Q, then this circle event is a false alarm and it must be deleted from Q.
3. Replace the leaf of $\mathcal{T}$ that represents α with a subtree having three leaves. The middle leaf stores the new site p_i and the other two leaves store the site p_j that was originally stored with α. Store the tuples $\langle p_j, p_i \rangle$ and $\langle p_i, p_j \rangle$ representing the new breakpoints at the two new internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary.
4. Create new half-edge records in the Voronoi diagram structure for the edge separating $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$, which will be traced out by the two new breakpoints.
5. Check the triple of consecutive arcs where the new arc for p_i is the left arc to see if the breakpoints converge. If so, insert the circle event into Q and add pointers between the node in $\mathcal{T}$ and the node in Q. Do the same for the triple where the new arc is the right arc.

4.1 Voronoidiagramme

Die Kreisereignisse (circle events) behandelt HANDLECIRCLEEVENT.

HANDLECIRCLEEVENT(γ)

1. Delete the leaf γ that represents the disappearing arc α from $\mathcal{T}$. Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary. Delete all circle events involving α from Q; these can be found using the pointers from the predecessor and the successor of γ in $\mathcal{T}$. (The circle event where α is the middle arc is currently being handled, and has already been deleted from Q.)
2. Add the center of the circle causing the event as a vertex record to the doubly-connected edge list $\mathcal{D}$ storing the Voronoi diagram under construction. Create two half-edge records corresponding to the new breakpoint of the beach line. Set the pointers between them appropriately. Attach the three new records to the half-edge records that end at the vertex.
3. Check the new triple of consecutive arcs that has the former left neighbor of α as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into Q, and set pointers between the new circle event in Q and the corresponding leaf of $\mathcal{T}$. Do the same for the triple where the former right neighbor is the middle arc.

4.1 Voronoidiagramme

Lemma 4.11.: Der Algorithmus benötigt $O(n \log n)$ Zeit und $O(n)$ Speicher.

Beweis: Die Operationen auf dem Baum T und der Ereignisschlange Q, also Suchen, Einfügen und Löschen, erfordern $O(\log n)$ Zeit. Die nötigen Operationen auf der doppelt verknüpften Kantenliste erfordern $O(1)$ Zeit. Für ein Ereignis haben wir eine konstante Anzahl dieser Operationen, also $O(\log n)$. Offensichtlich gibt es n Punktereignisse. Ferner erzeugt jedes Kreisereignis eine Ecke von Vor(p), da wir überflüssige Kreisereignisse rechtzeitig entfernen. Daher gibt es $2n$-5 Kreisereignisse. Daraus folgt die $O(n \log n)$ Zeitschranke und mit der max. Anzahl von $2n - 1$ Bögen am Strand (also max. $2n - 1$ Kreisereignisse gleichzeitig in Q) auch $O(n)$ für den Speicher. QED

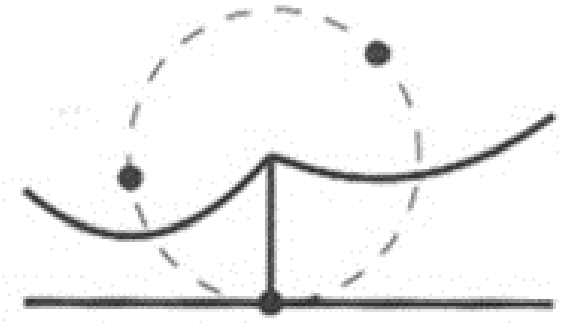
4.1 Voronoidiagramme

Wir werfen noch einen kurzen Blick auf Spezialfälle:

(1) Haben zwei Punkte gleiche y-Koordinate, so ist jede Reihenfolge zulässig. Nur wenn dies beim 2. Punkt passiert, muss man aufpassen, da es keinen Bogen oberhalb von p_2 gibt.

(2) Wenn zwei Kreisereignisse zusammentreffen, liegen vier Punkte auf einem Kreis. Die einfachste Lösung ist diesen Fall nicht besonders zu behandeln und die entstehenden Kanten der Länge *0* später aus dem Voronoidiagramm zu entfernen.

(3) Ferner kann ein Punktereignis p_i genau unterhalb eines Schnittes zweier Bögen auftreten.



4.1 Voronoidiagramme

In diesem Fall wird einer der beiden Bögen durch den neuen geteilt. Dabei entsteht ein Bogen der Länge *0*, der ein Kreisereignis an der Position p_i verursacht. Wenn dieses Kreisereignis abgearbeitet wird, verschwindet der Bogen der Länge *0* und es entsteht die benötigte Ecke des Voronoidiagramms.

(4) Wenn drei aufeinander folgende Bögen am Strand durch drei kollineare Punkte in *P* erzeugt werden, gibt es keinen Kreis durch die Punkte und wir erzeugen kein Kreisereignis.

4.1 Voronoidiagramme

Theorem 4.12.: Das Voronoidiagramm einer Menge von n Punkten in der Ebene kann durch ein sweep line Verfahren in $O(n \log n)$ Zeit mit $O(n)$ Speicher berechnet werden.

Es sei angemerkt, dass das Sortieren von n Zahlen als Berechnen eines quasi eindimensionalen Voronoidiagramms aufgefasst werden kann und daher $\Omega(n \log n)$ untere Schranke ist !

4.1 Voronoidiagramme

Literatur

Voronoizellen sind nach den Arbeiten von Voronoi [G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques, premier Mémoire: Sur quelques propriétés des formes quadratiques positives parfaites. J. Reine Angew. Math., 133:97-178, 1907, G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième Mémoire: Recherches sur les parallèlloèdres primitifs. J. Reine Angew. Math., 134:198-287, 1908] benannt. Da sie vorher schon bei Dirichlet [L. Dirichlet. Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. J. Reine Angew. Math., 40:209-227, 1850] auftraten, spricht man auch von Dirichlet Tesselierungen. Man kann das Prinzip bis Descartes [R. Descartes, Principia Philosophiae, Teil 3, 1644] zurückverfolgen. Okabe et al. gibt einen Überblick über Geschichte und Anwendungen [A. Okabe, B. Boots, K. Sugihara, Spatial Tesselations: Concepts and Applications of Voronoidiagrams. John Wiley & Sons, Chichester, UK, 1992].

4.1 Voronoidiagramme

Der erste optimale $O(n \log n)$ Zeit Algorithmus basierte auf divide and conquer [M. I. Shamos and D. Hoey. Closest-point problems. In Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci., pages 151-162, 1975]. Unsere Methode stammt von Fortune [S. J. Fortune. A sweepline algorithm for Voronoi diagrams. Algorithmica, 2:153-174. 1987] in der Fassung von Guibas und Stolfi [L. J. Guibas and J. Stolfi. Ruler, compass and computer: the design and analysis of geometric algorithms. In R. A. Earnshaw, editor, Theoretical Foundations of Computer Graphics and CAD. NATO ASI Sereis F. Vol. 40 pages 111-165, Springer-Verlag, 1988].

4.1 Voronoidiagramme

Verallgemeinerungen der Voronoidiagramme gibt es viele [F. Aurenhammer. Voronoi diagrams. A survey of a fundamental geometric data structure. ACM Comput. Surv., 23:345-405, 1991, A. Okabe, B. Boots, and L. Sugihara. Spatial Tesselations: Concepts and Applications of Voronoi Diagrams. John Wiley & Sons, Chichester, U. K., 1992].

(1) In $\mathbb{R}^d$ ergibt sich als maximale Anzahl der Elemente (Ecken, Kanten, Seiten, ...) der Raumunterteilung $\Theta\left(n^{\lceil d/2 \rceil}\right)$ [V. Klee. On the complexity of *d*-dimensional Voronoi diagrams. Archiv der Mathematik, 34:75-80, 1980].

Die Berechnung kann in $O\left(n\log n + n^{\lceil d/2 \rceil}\right)$ optimaler Zeit erfolgen. [B. Chazelle. An optimal convex hull algorithm and new results on cuttings. In Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci., pages 29-38, 1991, K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. Discrete Comput. Geom., 4:387-421, 1989, R. Seidel. Small-dimensional linear programming and convex hulls made easy. Discrete Comput. Geom., 6:423-434, 1991]

4.1 Voronoidiagramme

(2) Für die L_1-Metrik (Blockmetrik, Manhattanmetrik)

$$dist_1(p,q) := |p_x - q_x| + |p_y - q_y|$$

sind alle Kanten des Voronoidiagramms horizontal, vertikal oder diagonal. Zu den L_P-Metriken

$$dist_p(p,q) := \sqrt[p]{|p_x - q_x|^p + |p_y - q_y|^p}$$

gibt es diverse Artikel [L. P. Chew and R. L. Drysdale, III. Voronoi diagrams based on convex distance functions. In Proc. 1st Annu. ACM Sympos. Comput. Geom., pages 235-244, 1985, D. T. Lee. Two-dimensional Voronoi diagrams in the L_p-metric. J. ACM, 27:604-618, 1980, D. T. Lee and C. K. Wong. Voronoi diagrams in L_1 (L_∞) metric with 2-dimensional storage applications. SIAM J. Comput., 9.200-211, 1980].

Auch gewichtete Metriken (additiv, multiplikativ) wurden behandelt [F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighed Voronoi diagram in the plane. Pattern Recogn., 17:251-257, 1984, S. J. Fortune. A sweepline algorithm for Voronoi diagrams. Algorithmica, 2:153-174, 1987].

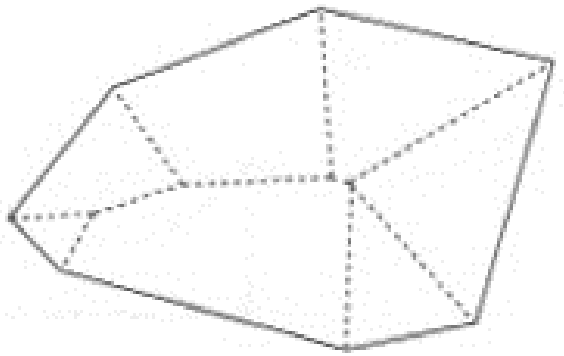
4.1 Voronoidiagramme

Weitere Abstandsfunktionen führen zu Powerdiagrammen.[F. Aurenhammer. A criterion for the affine equality of cell complexes in R^d and convex polyhedra in R^{d+1}. Discrete Comput. Geom, 2:49-64, 1987, F. Aurenhammer. Power diagrams: properties, algorithms and applications. SIAM J. Comput., 16:78-96, 1987, F. Aurenhammer. Linear combinations from power domains. Geom. Declicata, 28:45-52, 1988, F. Aurenahmmer, F. Hoffmann, and B. Aronov. Minkowski-type theorems and least squares clustering, Algorithmica, 20:61-76, 1998].

Auch abstrakte Diagramme ohne Abstandsfunktionen sind möglich [R. Klein. Abstract Voronoi diagrams and their applications. In Computational Geometry and its Applications. Lecture Notes in Computer Science, Vol. 333, pages 148-157, Springer-Verlag, 1988, R. Klein. Concrete and Abstract Voronoi Diagrams. Lecture Notes in Computer Science, Vol. 400, Springer-Verlag, 1989, R. Klein. K. Mehlhorn, and S. Meiser. Randomized incremental contruction of abstract Voronoi diagrams. Comput. Geom. Theory Appl., 3:157-184, 1993, K. Mehlhorn, S. Meiser, and C. O'Dunlaing. On the construction of abtract Voronoi diagrams. Discrete Comput. Geom., 6:211-224, 1991].

4.1 Voronoidiagramme

(3) Man kann statt der Punkte auch weitere Objekte verwenden, etwa Kanten und gelangt dann zur Medial Axis Transform. [F. Chin, J. Snoeyink, and C.-A. Wang. Finding the medial axis of a simple polygon in linear time. In Proc. 6th Annu. Internat. Sympos. Algorithms Comput. (ISAAC 95). Lecture Notes in Computer Science, Vol. 1004, pages 382-391, Springer-Verlag, 1995].



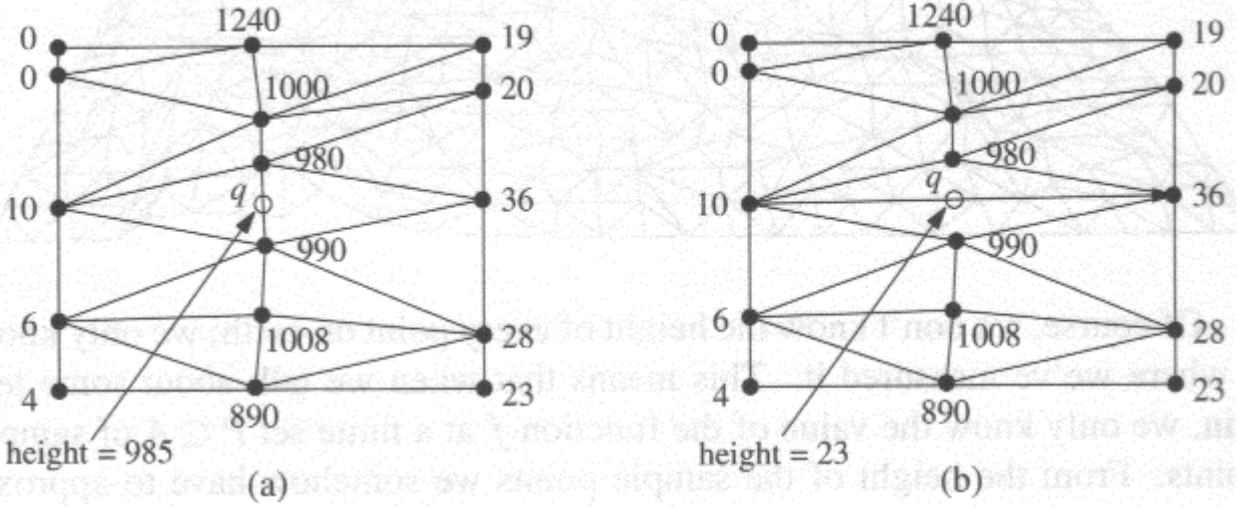
4.1 Voronoidiagramme

(4) Schließlich kann man statt des minimalen Abstandes zu einem Punkt auch noch dem Abstand zu den nächsten Punkten fragen. Dies liefert Voronoidiagramme der Ordnung k [F. Aurenhammer and O. Schwarzkpf. A simple on-line randomized incremental algorithm for compupting higher order Voronoi diagrams. Internat. J. Comput. Geom. Appl., 2:363-381, 1992, J.-D. Boissonnat, O. Bevillers, R. Schott, M. Teillaud. A semidynamic construction of higher-order Voronoi diagrams and its randomized analysis. Algorithmica 9:329-356, 1993, B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing kth order Voronoi diagrams. IEEE Trans. Comput., C-36:1349-1354, 1987].

Als Komplexität ergibt sich $\Omega\ (k(n-k)$ [D. T. Lee. On k-nearest neighbor. Voronoi diagrams in the plane. IEEE Trans. Comput., C-31:478-487, 1982]. Der schnellste bekannte Algorithmus benötigt $O(n \log^3 n + k(n-k))$ Zeit [P. K. Agarwal, M. de Berg, J. Matousek, and O. S. Zkopf. Constructing levels in arrangements and higher order Voronoi diagrams. SIAM J. Comput. 27:654-667, 1998]

4.2 Delaunaytriangulierung

Häufig stellt sich in der geometrischen Datenverarbeitung die Aufgabe, eine gegebene Punktmenge zu triangulieren. Schon bei den vier Punkten eines konvexen Vierecks lässt sich dies nicht eindeutig lösen. Man braucht also ein zusätzliches Kriterium, um zu einer eindeutigen Lösung zu kommen. In der Visualisierung dienen Triangulierungen häufig zur Interpolation von Messwerten an irregulär verteilten Positionen. Dabei möchte man gutmütige Dreiecke, die nicht zu "dünn" sind.



Flipping one edge can make a big difference

4.2 Delaunaytriangulierung

Ein häufiger Lösungsansatz liegt in der möglichst guten Vermeidung kleiner Winkel. Man sucht also eine Triangulierung, bei der der kleinste Winkel maximal wird. Diese Triangulierung heißt auch **Delaunay-Triangulierung**. Ihre Eigenschaften, ein effizienter Algorithmus und die Analyse der Komplexität werden in diesem Abschnitt besprochen.

4.2 Delaunaytriangulierung

Theorie

Definition 4.13: Sei **P=**{p_1, ... , p_q} eine Menge von Punkten in der Ebene. Eine **maximale planare Unterteilung *S* von *P*** ist eine planare Unterteilung, so dass keine zwei Punkte mehr durch eine Kante verbunden werden können, ohne dass *S* seine Eigenschaft als planare Unterteilung verliert. Eine **Triangulierung** von *P* ist eine maximale planare Unterteilung.

4.2 Delaunaytriangulierung

Aus der Definition folgt die Existenz von Triangulierungen und wegen der Triangulierbarkeit aller polygonalen Facetten auch, dass eine Triangulierung aus Dreiecken besteht. Ferner ist leicht einzusehen, dass die konvexe Hülle stets den Rand einer Triangulierung bilden muss. Dann lässt sich die Anzahl der Kanten und Dreiecke in einer Triangulierung mittels der Eulerformel ableiten.

4.2 Delaunaytriangulierung

Theorem 4.14: Sei *P* eine Menge von *n* Punkten in der Ebene, die nicht alle kollinear sind. Ferner seien *k* Punkte Ecken der konvexen Hülle. Dann hat jede Triangulierung von *P* $2n - 2 - k$ Dreiecke und $3n - 3 - k$ Kanten.

Beweis: Sei *m* die Anzahl der Dreiecke, *f* die Anzahl der Facetten und *e* die Anzahl der Kanten. Wegen der unbeschränkten Facette gilt $f = m + 1$. Jede Kante berandet 2 Facetten. Die Dreiecke haben 3 Kanten und die unbeschränkte Facette *k*. Also $2e = 3m + k$. Euler sagt uns:

(a) $$2 = p - e + f = n - \frac{3}{2}m - \frac{k}{2} + m + 1 \Rightarrow \frac{1}{2}m = n - 1 - \frac{k}{2} \Rightarrow m = 2n - 2 - k$$

(b) $$2 = p - e + f = n - e + m + 1 = n - e + 2n - 2 - k + 1 \Rightarrow e = 3n - 3 - k$$

QED

4.2 Delaunaytriangulierung

Wir brauchen nun einen Formalismus, um die Triangulierung bzgl. der kleinsten auftretenen Winkel zu vergleichen.

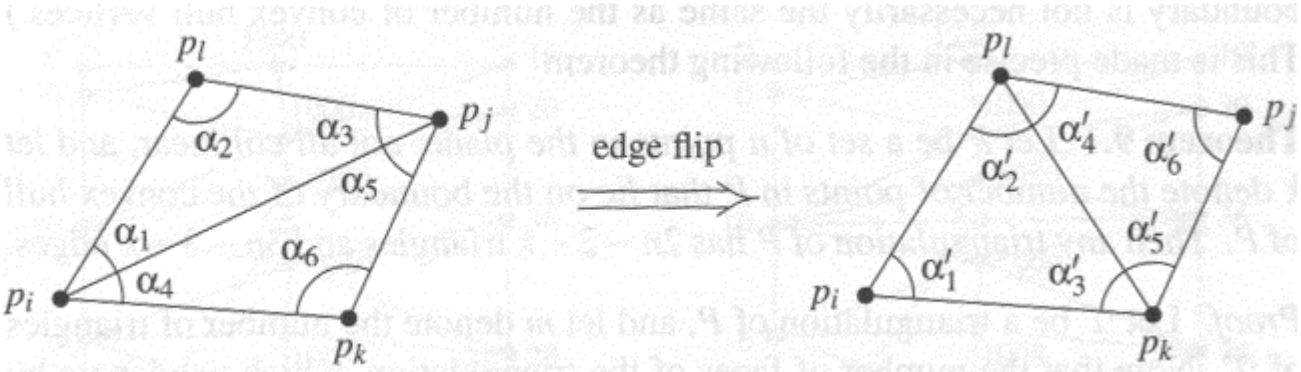
Definition 4.15: Sei *T* eine Triangulierung von *P* mit m Dreiecken. Betrachte die aufsteigend sortierte Sequenz $\alpha_{1,}\alpha_2, \cdots, \alpha_{3\mathrm{m}}$ der 3m Winkel in **T**, also $\alpha_i \leq \alpha_k \forall k > i.$ Wir nennen $A(T) = (\alpha_1, \cdots, \alpha_{3\mathrm{m}})$ den **Winkelvektor von** *T*. Ferner definieren wir die lexikographische Ordnung auf den Winkelvektoren:

$$A(T) < A(T') gdw\ \exists 1 \leq i \leq 3\mathrm{m}: \alpha_j = \alpha'_j für\ j < i, \alpha_i < \alpha'_i$$

Eine Triangulierung *T* heiße **winkeloptimal**, falls *A*(*T*) < *A*(*T'*) für alle Triangulierungen *T'* von *P*.

4.2 Delaunaytriangulierung

Eine offensichtliche, elementare Operation für das Verändern von Triangulierungen ist das Vertauschen von Diagonalen innerhalb eines konvexen Vierecks (flipping edges).



Flipping an edge

4.2 Delaunaytriangulierung

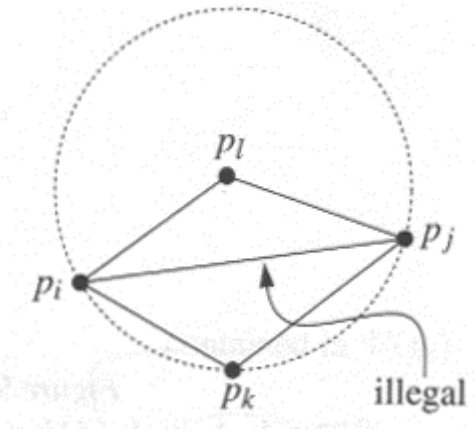
Dabei ändern sich die Winkel $(\alpha_1, \cdots, \alpha_6)\ in\ A(T)\ in\ (\alpha'_1, \cdots, \alpha'_6) um.$ Die Kante $e = \overline{p_i p_j}$ nennen wir **illegal** falls

$$\min_{1 \le i \le 6} \alpha_i < \min_{1 \le i \le 6} \alpha'_i$$

Beobachtung 4.16: Sei *T* eine Triangulierung mit illegaler Kante *e*. Sei *T* ' die Triangulierung mit vertauschter Diagonale *e*. Es gilt *A*(*T*) < *A*(*T* ').

4.2 Delaunaytriangulierung

Lemma 4.17: Sei $\overline{p_i p_j}$ eine Kante inzident zu den Dreiecken $p_i\, p_j\, p_k$ und $p_i\, p_j\, p_l$. Wir betrachten den Kreis C durch p_i p_j und p_k: $\overline{p_i p_j}$ ist illegal gdw p_l im Inneren von C liegt. Bilden p_i p_j p_k p_l ein konvexes Viereck und liegen nicht auf einem Kreis, so ist entweder $\overline{p_i p_j}$ *oder* $\overline{p_k p_l}$ illegal.



4.2 Delaunaytriangulierung

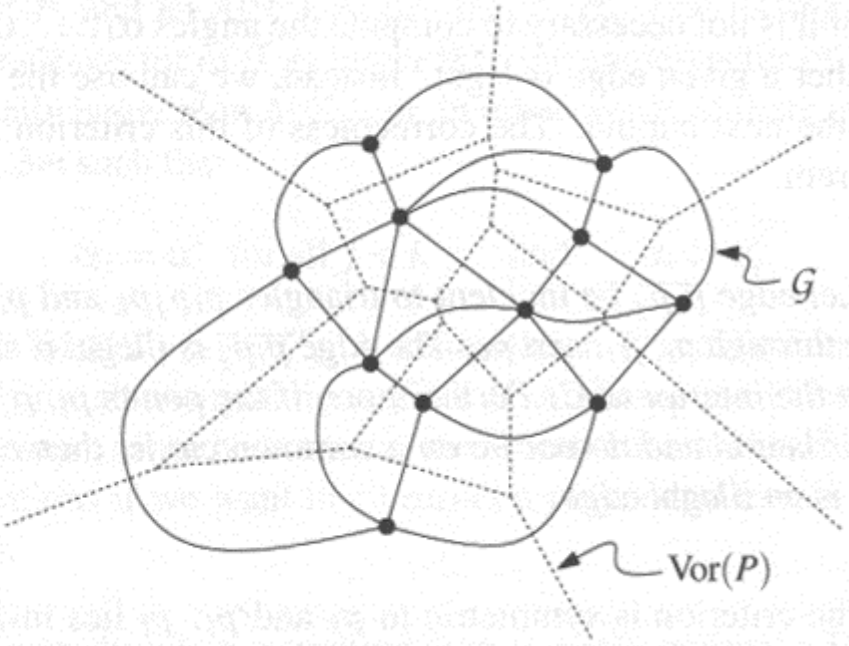
Da eine illegale Kante eine Verbesserung des Winkelvektors gestattet, ist eine winkeloptimale Triangulierung legal. Dies führt zu einem einfachen Algorithmus.

Algorithm LEGALTRIANGULATION($\mathcal{T}$)
Input. Some triangulation $\mathcal{T}$ of a point set P.
Output. A legal triangulation of P.
1. **while** $\mathcal{T}$ contains an illegal edge $\overline{p_i p_j}$
2. **do** (∗ Flip $\overline{p_i p_j}$ ∗)
3. Let $p_i p_j p_k$ and $p_i p_j p_l$ be the two triangles adjacent to $\overline{p_i p_j}$.
4. Remove $\overline{p_i p_j}$ from $\mathcal{T}$, and add $\overline{p_k p_l}$ instead.
5. **return** $\mathcal{T}$

4.2 Delaunaytriangulierung

Da jedes Vertauschen von Diagonalen den Winkelvektor vergrößert und somit zu einer neuen Triangulierung führt, ergibt sich aus der endlichen Anzahl der Triangulierungen, dass der Algorithmus terminiert. Leider ist der Algorithmus aufwändig.

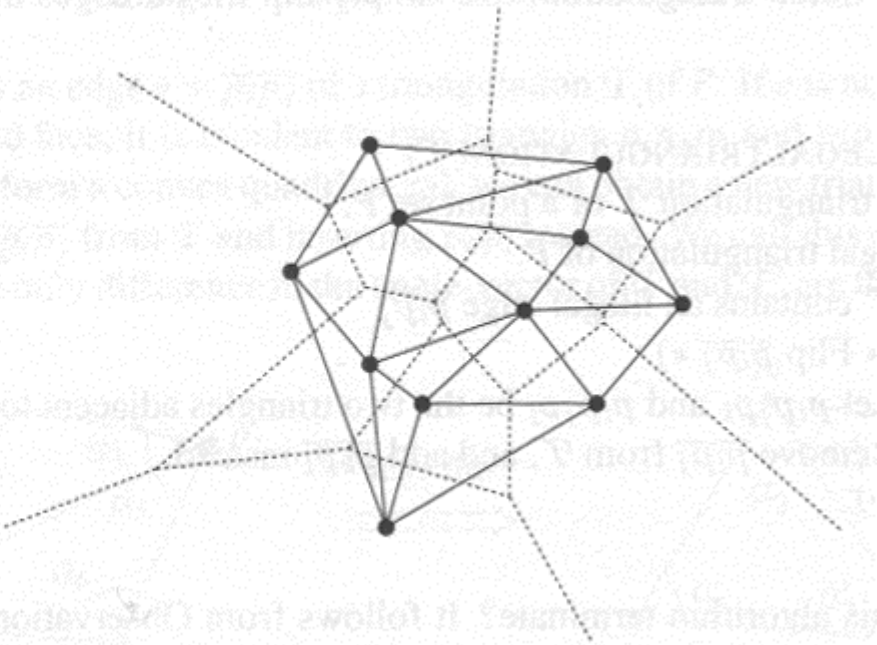
Bevor wir erneut an einen Algorithmus gehen, betrachten wir einige interessante Eigenschaften der Voronoizellen des Abschnittes 4.1. Sei *G* der zum Voronoidiagramm duale Graph.



The dual graph of Vor(P)

4.2 Delaunaytriangulierung

Diesen Graph betten wir mit geraden Kanten in die Ebene ein und nennen diese Einbettung Delaunaygraph zur Punktmenge *P*.



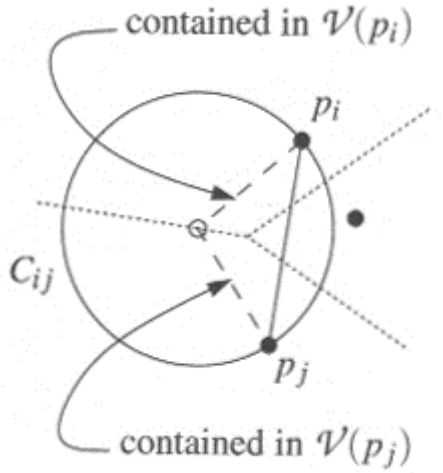
The Delaunay graph $\mathcal{DG}(P)$

4.2 Delaunaytriangulierung

Zunächst überzeugen wir uns davon, dass sich keine Kanten schneiden können.

Theorem 4.18: Der Delaunaygraph einer ebenen Punktmenge ist ein planarer Graph.

Beweis: Wir nehmen an, dass sich die Kanten $\overline{p_i p_j}$ *und* $\overline{p_k p_l}$ unseres Graphen schneiden. Dazu müssen p_i, p_j, p_k, p_l alle verschieden sein, da sich die Kanten sonst in einem Eckpunkt treffen würden. Nach Theorem 4.5.(ii) gibt es zur Kante e_{ij} des Voronoidiagrammes, die $V(p_i)$ und $V(p_j)$ trennt, um jeden inneren Punkt q_{ij} einen Kreis C_{ij} durch p_i und p_j, der keine anderen Punkte aus P trifft. Sei nun t_{ij} das Dreieck aus q_{ij}, p_i und p_j.



4.2 Delaunaytriangulierung

Wegen des Schnittes der Kanten unseres Graphen und der Kreisbedingung muss $\overline{p_k p_l}$ durch das ganze Dreieck laufen und auch die Strecke $\overline{q_{ij} p_i}$ oder $\overline{q_{ij} p_j}$ schneiden. Analog finden wir zu $\overline{p_k p_l}$ einen Punkt q_{kl}, einen Kreis C_{kl}, ein Dreieck t_{kl} und einen Schnitt von $\overline{p_i p_j}$ mit $\overline{q_{kl} p_k}$ $oder$ $\overline{q_{kl} p_l}$. Damit schneiden sich die beiden Dreiecke t_{kl} und $t_{i\ j}$, insbesondere etwa $\overline{q_{ij} p_i} und$ $\overline{q_{kl} p_k}$ (oder eine der andere vier Varianten). Da aber $\overline{q_{ij} p_i}$ und $\overline{q_{kl} p_k}$ in disjunkten Voronoizellen liegen, ist dies unmöglich! QED

4.2 Delaunaytriangulierung

Wie üblich, schließen wir nun einmal aus, dass die Punkte von *P* sich in spezieller Lage zueinander befinden. Hier heißt das, dass vier Punkte nicht auf einem Kreis liegen.

Definition 4.19: Die Punkte aus *P* sind in **allgemeiner** Lage, falls nicht vier Punkte auf einem Kreis liegen.

In diesem (und nur in diesem) Fall ist der Delaunaygraph eine Triangulierung, sonst muss man noch ein paar Facetten (Vierecke, Fünfecke, ...) auf irgendeine Weise triangulieren. Das Ergebnis nennen wir **Delaunaytriangulierung.**

4.2 Delaunaytriangulierung

Wie schon gesehen, sind die Eigenschaften der Voronoizellen für die Delaunaygraphen nützlich. Daher notieren wir über Theorem 4.5. (ii):

Theorem 4.20: Sei P eine Menge von Punkten in der Ebene.

(i) Drei Punkte $p_i, p_j, p_k \in P$ gehören zur gleichen Facette des Delaunaygraphs gdw der Kreis durch p_i, p_j, p_k keinen Punkt aus P im Inneren hat.

(ii) Zwei Punkte $p_i, p_j \in P$ bilden eine Kante im Delaunaygraph gdw es eine abgeschlossene Kreisscheibe durch p_i und p_j gibt, die keinen anderen Punkt aus P enthält.

Theorem 4.21: Sei P eine planare Punktmenge und T eine Triangulierung. Dann ist T eine Delaunaytriangulierung gdw der Umkreis jedes Dreieckes keinen Punkt aus P im Inneren hat.

4.2 Delaunaytriangulierung

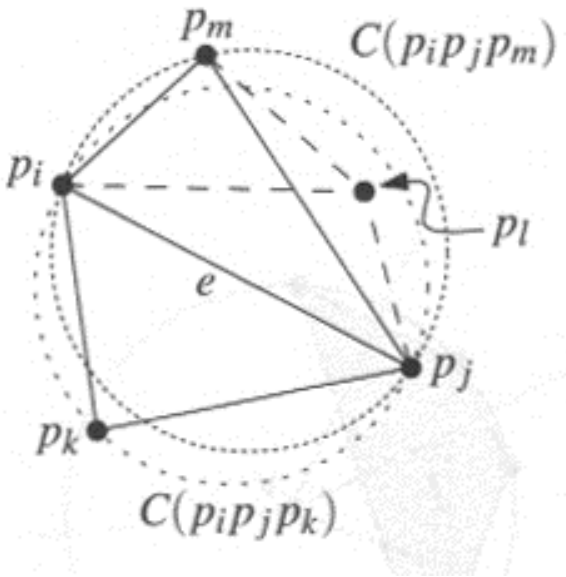
Nun zeigen wir noch, dass wir eigentlich Delaunaytriangulierungen suchen.

Theorem 4.22: Sei *P* eine planare Punktmenge. Eine Triangulierung von *P* ist legal gdw *T* eine Delaunaytriangulierung von *P* ist.

Beweis: Die Rückrichtung der Aussage folgt direkt aus dem letzten Theorem und Lemma 4.17.

Die Hinrichtung kann man durch Widerspruch schnell zeigen. Sei *T* also legale Triangulierung aber nicht Delaunay. Dann gibt es ein Dreieck $p_i\ p_j\ p_k$ und einen Punkt p_l innerhalb des Umkreises. Wir betrachten das Viereck $p_i\ p_j\ p_k\ p_l$. (*OE* seien $p_ip_jp_l$ und $p_i\ p_j\ p_k$ sich nicht überschneidende Dreiecke.) Unter allen diesen Paaren (Dreieck $p_ip_jp_k$, Punkt p_l) suche das Paar, das den Winkel $p_ip_lp_j$ maximiert. Betrachte nun das Nachbardreieck $p_ip_jp_m$ entlang p_ip_j. Dann sind p_m und p_l verschieden, da sonst p_ip_j nicht legal sein kann (vgl. Lemma 4.17).

4.2 Delaunaytriangulierung



Der Kreis C_{ijm} durch p_i,p_j,p_m enthält den Teil des Kreises C_{ijk} durch p_i,p_j,p_k außerhalb des Dreiecks $p_ip_jp_k$, weil sonst p_ip_j nicht legal wäre. Also liegt p_l in C_{ijm}. Es sei nun p_jp_m die Kante des Dreiecks $p_ip_jp_m$, bei der das Dreieck $p_jp_mp_l$ nicht das Dreieck $p_ip_jp_m$ schneidet. (Der Fall p_ip_m ist vollkommen analog.) Dann ist der Winkel $p_jp_lp_m$ größer als der Winkel $p_ip_lp_j$ nach Thales' Satz im Widerspruch zur Definition des Paares (Dreieck $p_ip_jp_k$, Punkt p_l). QED

4.2 Delaunaytriangulierung

Es folgt nun:

Theorem 4.23: Sei *P* eine Menge von Punkten in der Ebene. Jede winkeloptimale Triangulierung von *P* ist eine Delaunaytriangulierung. Ferner maximiert jede Delaunaytriangulierung den minimalen Winkel.

Wenn die Punkte in *P* in allgemeiner Lage sind, gibt es nur eine Delaunaytriangulierung und somit nur eine legale Triangulierung. Wenn *P* in "spezieller" Lage ist, wird im Allgemeinen nur der kleinste Winkel maximiert. Durch Vertauschen von Diagonalen innerhalb der nicht dreiseitigen Facetten des Delaunaygraphs kann man auch eine winkeloptimale Triangulierung finden.

4.2 Delaunaytriangulierung

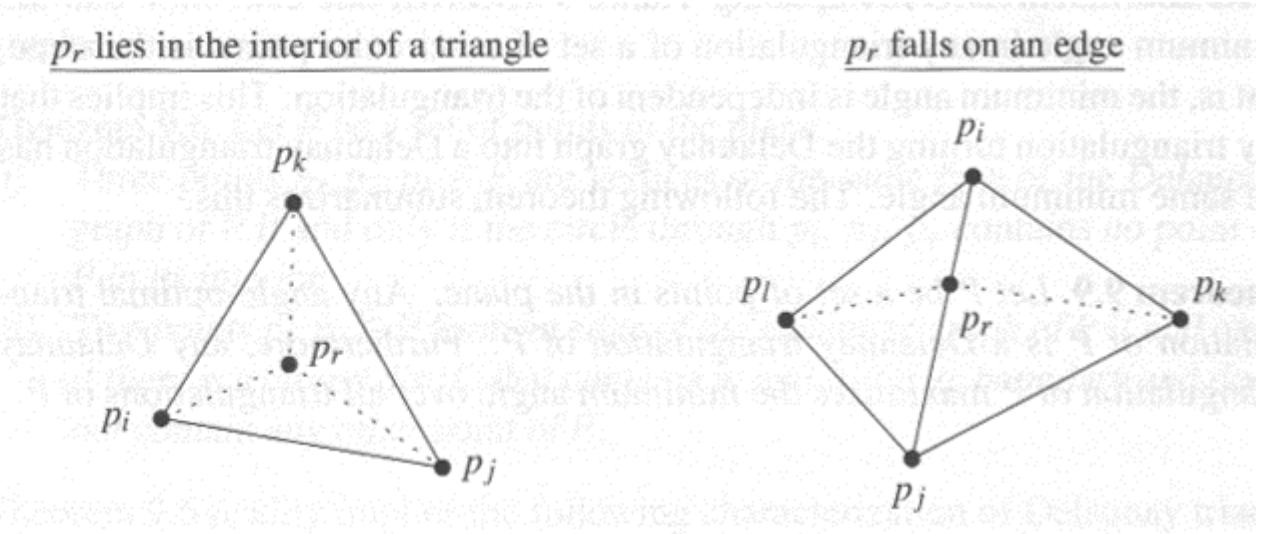
Algorithmus und Datenstrukturen

Eine Variante zur Lösung des Problems wäre es natürlich, ein Voronoidiagram zu berechnen und daraus die Delaunaytriangulierung zu ermitteln. Wir werden hier einen anderen Zugang nehmen, indem wir wieder einen nicht deterministischen, inkrementellen Algorithmus entwerfen.

Um Spezialfällen aus dem Weg zu gehen, beginnen wir mit einem ausreichend großen Dreieck p_{-1}, p_{-2}, p_{-3}, in dem P komplett enthalten ist. Damit die drei Punkte keinen Einfluss auf die Triangulierung von P haben, müssen sie außerhalb aller Kreise durch Tripel von P sein.

4.2 Delaunaytriangulierung

Der Algorithmus selbst fügt nun die Punkte aus P zufällig in die Triangulierung der bisherigen Punkte und p_{-1}, p_{-2}, p_{-3} ein. Dazu ermittelt er das Dreieck, in dem der neue Punkt liegt und fügt drei Kanten ein. Sollte der Punkt auf einer Kante liegen, sind nur zwei Kanten einzufügen. Um sicher zu stellen, dass es sich um eine Delaunaytriangulierung handelt, sind noch Diagonalen zu vertauschen. Dies erledigt LEGALIZEEDGE und wird anschließend betrachtet.



The two cases when adding a point p_r

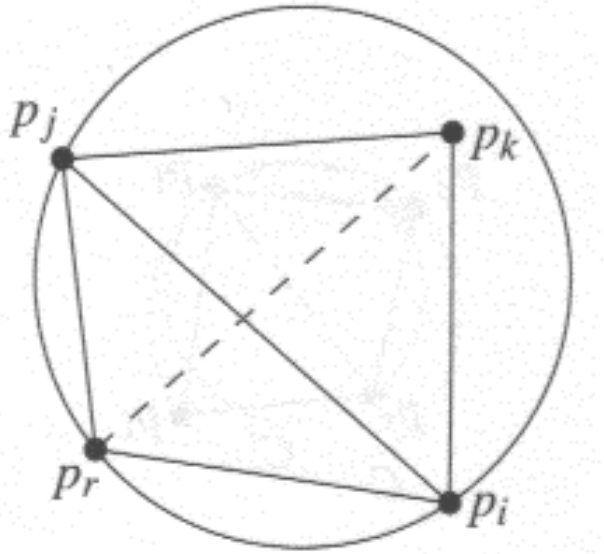
4.2 Delaunaytriangulierung

Algorithm DELAUNAYTRIANGULATION(P)
Input. A set P of n points in the plane.
Output. A Delaunay triangulation of P.

1. Let p_{-1}, p_{-2}, and p_{-3} be a suitable set of three points such that P is contained in the triangle $p_{-1}p_{-2}p_{-3}$.
2. Initialize $\mathcal{T}$ as the triangulation consisting of the single triangle $p_{-1}p_{-2}p_{-3}$.
3. Compute a random permutation $p_1, p_2, \ldots, p_n$ of P.
4. **for** $r \leftarrow 1$ **to** n
5. **do** (* Insert p_r into $\mathcal{T}$: *)
6. Find a triangle $p_ip_jp_k \in \mathcal{T}$ containing p_r.
7. **if** p_r lies in the interior of the triangle $p_ip_jp_k$
8. **then** Add edges from p_r to the three vertices of $p_ip_jp_k$, thereby splitting $p_ip_jp_k$ into three triangles.
9. LEGALIZEEDGE($p_r, \overline{p_ip_j}, \mathcal{T}$)
10. LEGALIZEEDGE($p_r, \overline{p_jp_k}, \mathcal{T}$)
11. LEGALIZEEDGE($p_r, \overline{p_kp_i}, \mathcal{T}$)
12. **else** (* p_r lies on an edge of $p_ip_jp_k$, say the edge $\overline{p_ip_j}$ *)
13. Add edges from p_r to p_k and to the third vertex p_l of the other triangle that is incident to $\overline{p_ip_j}$, thereby splitting the two triangles incident to $\overline{p_ip_j}$ into four triangles.
14. LEGALIZEEDGE($p_r, \overline{p_ip_l}, \mathcal{T}$)
15. LEGALIZEEDGE($p_r, \overline{p_lp_j}, \mathcal{T}$)
16. LEGALIZEEDGE($p_r, \overline{p_jp_k}, \mathcal{T}$)
17. LEGALIZEEDGE($p_r, \overline{p_kp_i}, \mathcal{T}$)
18. Discard p_{-1}, p_{-2}, and p_{-3} with all their incident edges from $\mathcal{T}$.
19. **return** $\mathcal{T}$

4.2 Delaunaytriangulierung

Da illegale Kanten auftreten können, müssen Diagonalen vertauscht werden. Der Algorithmus deutet bereits an, dass dies genau für die drei oder vier Kanten um den Punkt p_r herum auftreten kann, da p_r innerhalb entsprechender Umkreise sein kann. Für eine Kante $\overline{p_i p_j}$ hat man folgende Situation.

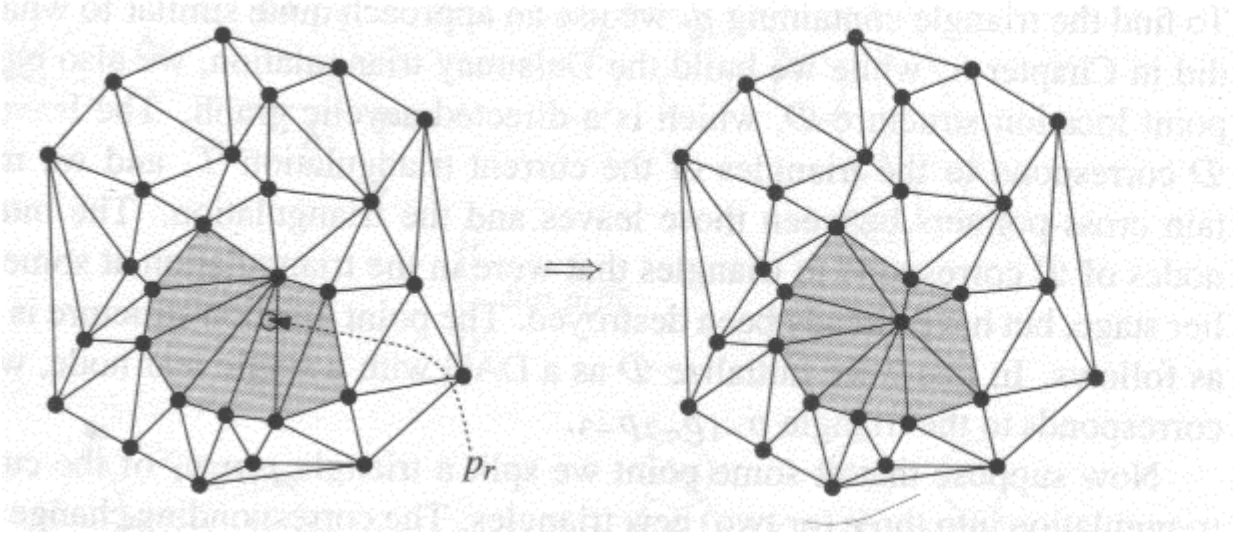


4.2 Delaunaytriangulierung

Da nach dem Vertauschen einer Diagonale weitere illegale Kanten auftreten können, sind diese ebenfalls zu prüfen. Da wir aber Verbesserungen auf dem Weg zu einer Delaunaytriangulierung von p_{-3}, p_{-2}, p_{-1}, p_1 , ... , p_r erreichen, bricht die Kaskade von Veränderungen nach einiger Zeit ab. Eine genaue Analyse erfolgt später.

LEGALIZEEDGE($p_r, \overline{p_i p_j}, \mathcal{T}$)
1. (∗ The point being inserted is p_r, and $\overline{p_i p_j}$ is the edge of $\mathcal{T}$ that may need to be flipped. ∗)
2. **if** $\overline{p_i p_j}$ is illegal
3. **then** Let $p_i p_j p_k$ be the triangle adjacent to $p_r p_i p_j$ along $\overline{p_i p_j}$.
4. (∗ Flip $\overline{p_i p_j}$: ∗) Replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$.
5. LEGALIZEEDGE($p_r, \overline{p_i p_k}, \mathcal{T}$)
6. LEGALIZEEDGE($p_r, \overline{p_k p_j}, \mathcal{T}$)

4.2 Delaunaytriangulierung



All edges created are incident to p_r.

4.2 Delaunaytriangulierung

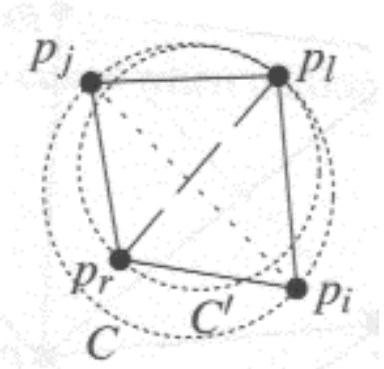
Wir zeigen noch , dass jede vertauschte Diagonale zum Delaunaygraph gehört.

Lemma 4.24: Jede erzeugte Kante in DELAUNAYTRIANGULATION und LEGALIZEEDGE während des Einfügens von p_r ist eine Kante des Delaunaygraphen von $\{p_{-3}, p_{-2}, p_{-1}, \cdots, p_r\}$.

Beweis: Wir betrachten die Kanten $\overline{p_r p_i}, \overline{p_r p_j}, \overline{p_r p_k}$ aus dem Aufteilen des Dreieckes p_i p_j p_k durch p_r. Da wir vor dem Einfügen von p_r eine Delaunaytriangulierung hatten, gibt es einen von anderen Punkten freien Umkreis C zum Dreieck p_i p_j p_k. In diesem liegt nun ein Kreis C' durch p_r , p_i, so dass $\overline{p_r p_i}$ im Delaunaygraph ist. Analog $\overline{p_r p_j}$ und $\overline{p_r p_k}$, sowie im Sonderfall $\overline{p_r p_l}$. Für die Kanten von LEGALIZEEDGE gehen wir ähnlich vor. Hier wird die Kante $\overline{p_i p_j}$ eines Dreieckes p_i p_j p_l durch eine Kante $\overline{p_r p_l}$ ersetzt. Auch hier gibt es den (bis auf p_r) leeren Umkreis

zu p_i p_j p_l, der innerhalb eines Kreises um $\overline{p_r p_l}$ liegt,

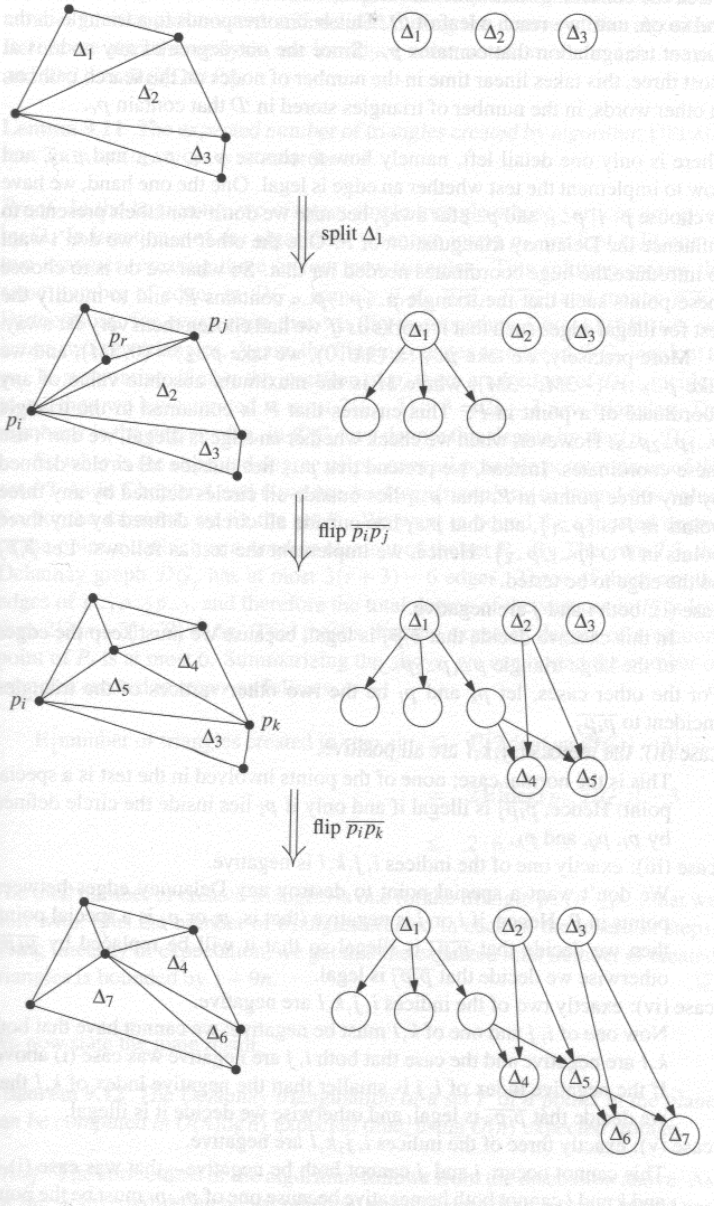
so dass diese Kante im Delaunaygraphen sein muss. QED



4.2 Delaunaytriangulierung

Um schnell die in Zeile 6 nötige Suche nach dem Dreieck, in dem p_r liegt, durchführen zu können, bauen wir eine den trapezförmigen Kanten nachempfundene Suchstruktur *D* in Form eines gerichteten, azyklischen Graphen auf. Die Blätter sind die Dreiecke in *T* und die inneren Knoten entsprechen Dreiecken, die es zu einem frühen Zeitpunkt im Algorithmus einmal gab. Suche und Anpassung der Suchstruktur *D* erläutern die nächsten Skizzen.

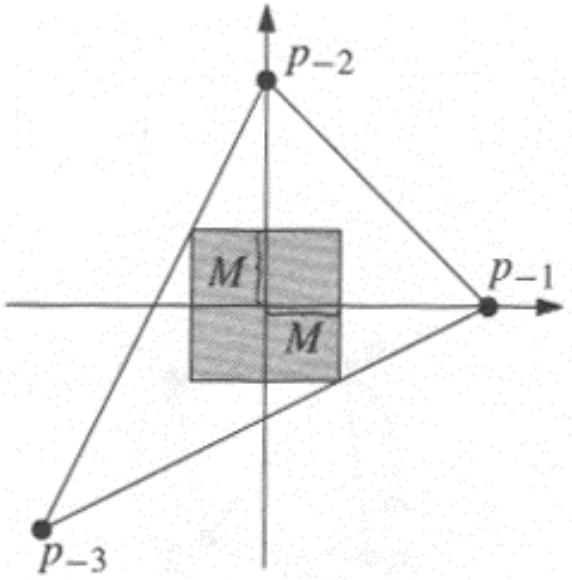
4.2 Delaunaytriangulierung



The effect of inserting point p_r into triangle Δ_1 on the data structure $\mathcal{D}$ (the part of $\mathcal{D}$ that does not change is omitted in the figure)

4.2 Delaunaytriangulierung

Um nun noch p_{-1}, p_{-2}, p_{-3} festzulegen, berechnen wir das Maximum M aller Beträge aller Koordinaten und setzen p_{-1} = (3M, 0), p_{-2} =(0, 3M), p_{-3} = (-3M, -3M). Bei der Berechnung der Kreisbedingung ist dann zu berücksichtigen, ob einer der drei Punkte beteiligt ist.



4.2 Delaunaytriangulierung

Analyse

Wir betrachten zunächst die strukturellen Änderungen durch den Algorithmus. Dazu sei $P_r = \{p_1 , ... , p_r\}$, $DG_r = DG\ (\{p_{-3} , p_{-2} , p_{-1}\} \cup P_r)$

Lemma 4.25: Die erwartete Anzahl erzeugter Dreiecke von DELAUNAYTRIANGULATION ist maximal *9n+1.*

Beweis: Zu Beginn wird 1 Dreieck erzeugt. In Schritt r werden ein oder zwei Dreiecke zerlegt, so dass 3 oder 4 neue Dreiecke entstehen. Es entsteht eine entsprechende Anzahl Kanten in DG_r, die bei p_r starten. Ferner erzeugt jeder Diagonalentausch in LEGALIZEEDGE 2 neue Dreiecke. Dabei entsteht stets eine Kante, die bei p_r startet. Ist k der Grad von p_r in DG_r, so werden $2(k - 3) + 3 = 2k - 3$ neue Dreiecke erzeugt.

4.2 Delaunaytriangulierung

Mittels Rückwärtsanalyse stellen wir fest, dass p_r zufällig aus P_r ausgewählt ist. Also müssen wir den Erwartungswert für k bestimmen. Nach 4.14 gibt es in DG_r $3(r + 3) – 6$ Kanten. Die drei Kanten außen enden nicht in P_r, also bleibt für die Summe der Grade

$$\sum_{p \in P_r} deg\,(p) \leq 2 \cdot (3(r+3) - 6 - 3)$$

$$= 6r$$

also E (deg(p)) = 6 und es werden 2 * 6 – 3 = 9 neue Dreiecke pro Punkt erwartet. Dazu kommt noch das eine Dreieck zu Beginn. QED

4.2 Delaunaytriangulierung

Satz 4.26: Die Delaunaytriangulierung einer Menge P von n Punkten in der Ebene kann in $O(n \log n)$ erwarteter Zeit mit $O(n)$ erwartetem Speicher berechnet werden.

Beweis: Die Korrektheit wissen wir schon. Aus dem Aussagen über die Triangulierung wissen wir, dass $O(n)$ Speicherplätze für die Struktur ausreichen. Das letzte Lemma beschränkt ferner den Umfang der Suchstruktur D, da jeder Knoten zu einem Dreieck gehört, das im Laufe des Algorithmus erzeugt wird. Also ist $O(n)$ Speicher korrekt.

4.2 Delaunaytriangulierung

Ferner wissen wir aus dem Lemma 4.25, dass bis auf die Punktsuche alle Schritte $O(n)$ Zeit benötigen. Der Aufwand der Suche ist linear in der Anzahl Dreiecke, die in irgendeiner Zeit erzeugt wurden und in denen p_r liegt. Betrachten wir die beiden Fälle zur Zerstörung von Dreiecken:

- Ein neuer Punkt p_l wird innerhalb eines Dreieckes $p_i p_j p_l$ eingefügt.
- Ein Diagonalentausch ersetzt $p_i p_j p_k$ und $p_i p_j p_l$ durch $p_k p_i p_j$ *und* $p_k p_j p_l$.

Stets ist ein Dreieck Δ durch einen neuen Punkt zerstört worden, wobei p_r im Umkreis des Dreiecks liegt. Sei nun $K(\Delta)$ die Anzahl der Punkte in P, die im Umkreis von Δ liegen. Da Δ beim Auftreten des ersten Punktes im Umkreis zerstört wird, ergibt sich

$$O(n) + \sum_{\Delta} card\,(K(\Delta))$$

als Aufwand, wobei Δ über alle im Algorithmus erzeugten Dreiecke läuft. Im nächsten Lemma zeigen wir

$$O\left(n + \sum_{\Delta} card\left(K(\Delta)\right)\right) = O(n \log n).$$

QED. (Eigentlich reicht als Aufwand für die Punktsuche die Anzahl der Dreiecke, in denen der Punkt liegt. Diese Zahl ist aber sicher kleiner oder gleich der Anzahl der Dreiecke, in deren Umkreis der Punkt liegt. Wir nutzen diese Zahl, weil sie einfacher zu berechnen ist.)

4.2 Delaunaytriangulierung

Lemma 4.27: Wenn *P* eine Punktmenge in allgemeiner Lage ist, gilt

$$\sum_{\Delta} card\,(K(\Delta)) = O(n \log n)$$

wobei die Summe über alle vom Algorithmus erzeugten Dreiecke Δ läuft.

Beweis: Wenn *P* in allgemeiner Lage ist, so auch die Teilmenge P_r. Damit gibt es einen eindeutigen Graphen $DG(\{p_{-3}, p_{-2}, p_{-1}\} \cup P_r)$, dessen Menge von Dreiecken wir T_r nennen. Die Dreiecke aus Schritt r des Algorithmus sind dann $T_r - T_{r-1}$. Also gilt

$$\sum_{\Delta} card\,(K(\Delta)) = \sum_{r=1}^{n} \sum_{\Delta \in T_r - T_{r-1}} card\,(K(\Delta))$$

Zu einem Punkt *q* sei $k(P_r, q)$ die Anzahl Dreiecke Δ in T_r, so dass *q* in $K(\Delta)$ liegt, und davon $k(P_r, q, p_r)$ die Dreiecke inzident zu p_r. Da alle neuen Dreiecke im Schritt *r* inzident zu p_r sind, folgt

$$\sum_{\Delta \in T_r - T_{r-1}} card\,(K(\Delta)) = \sum_{q \in P - P_r} k\,(P_r, q, p_r)$$

4.2 Delaunaytriangulierung

Sei nun P_r^* eine feste Wahl von P_r. Wir berechnen den Erwartungswert für alle Permutationen von P mit diesem festen P_r^*. Da ein Dreieck $\Delta \in T_r^*$ mit Wahrscheinlichkeit $3/r \; zu \; p \in P_r^*$ inzident ist , gilt

$$E(k(P_r^*, q, p_r)) \leq \frac{3k(P_r^*, q)}{r}$$

Wenn wir über alle $q \in P - P_r^*$ summieren, folgt

$$E[k(P_r^*, p_{r+1})] = \frac{1}{n-r} \sum_{q \in P - P_r} k\,(P_r^*, q)$$

und somit

$$E\left[\sum_{\Delta \in T_r^* - T_{r-1}^*} card\,(K(\Delta))\right] \leq 3\,(\frac{n-r}{r})\,E[k(P_r^*, p_{r+1})]$$

4.2 Delaunaytriangulierung

Nun ist k (P_r^*, p_{r+1}) die Anzahl Dreiecke $\Delta \in T_r^* mit \quad p_{r+1} \in K(\Delta).$ Genau diese Dreiecke werden im Schritt *r* + 1 verändert, also

$$E\left[\sum_{\Delta \in T_r^* - T_{r-1}^*} card\,(K(\Delta))\right] \leq 3\,(\frac{n-r}{r})\,E[card(T_r^* - T_{r+1}^*)]$$

Nach Theorem 4.14 sind in $T_m^* \; genau \; 2(m+3) - 2 - 3 = 2\mathrm{m} + 1$ Dreiecke, also hat $T_{r+1}^* 2\; Dreiecke\; mehr\; als\; T_r^*:$

$$E\left[\sum_{\Delta \in T_r^* - T_{r-1}^*} card\,(\mathrm{K}(\Delta))\right] \leq 3\,(\frac{n-r}{r})\,(E[card(T_{r+1}^* - T_r^*)] - 2)$$

4.2 Delaunaytriangulierung

Nun nehmen wir das Mittel über alle *P** auf beiden Seiten, was die Aussage nicht ändert. Dann können wir noch nutzen, dass im Mittel max. 6 Dreiecke in T_{r+1} inzident zu p_{r+1} sind und genau diese card ($T_{r+1} – T_r$) ausmachen. Also

$$E\left[\sum_{\Delta\in T_r^* - T_{r-1}^{\circ *}} card\left(K(\Delta)\right)\right] \le 3\left(\frac{n-r}{r}\right)\left(E\left[card\left(T_{r+1} - T_r\right)\right]-2\right) = 3\left(\frac{n-r}{r}\right)(6-2) = 12\left(\frac{n-r}{r}\right).$$

$$\sum_{r=1}^{n} 12\frac{n-r}{r} = 12n\sum_{r=1}^{n}\frac{1}{r} - 12n = O(n\log n).$$

QED.

4.2 Delaunaytriangulierung

Literatur:

Aus der Numerik ist bekannt, dass gute Interpolationen durch Vermeiden langer, schmaler Dreiecke erzeugt werden [R. E. Barnhill. Representation and approximation of surfaces. In J. R. Rice, editor, math, Software III, pages 69-120. Academic Press, New York, 1997]. Rippa [S. Rippa. Minimal roughness, property of the Delaunay triangulation. Comput. Aided Geom. Design, 7:489-497, 1990] hat gezeigt, dass Delauaytriangulierungen stets die Rauhheit eines Terrains reduzieren, und zwar unabhängig von der Höhenverteilung. (Datenunabhängiger Zugang ist hier gut!)

Die Aussage, dass das Vertauschen von Diagonalen jede Triangulierung in eine beliebige andere überführt, stammt von Lawson [C. L. Lawson. Transforming triangulations. Discrete Math., 3:365-372, 19972], Sibson [R. Sibson. Locally equiangular triangulations. Computer Journal 21:243-245,1978] hat die Eindeutigkeit der Delaunaytriangulierung für Punkte in allgemeiner Lage gezeigt.

4.2 Delaunaytriangulierung

Unser Algorithmus stammt von Guibas et al. [L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica, 7:381-413,1992] und die Analyse von Mulmuley [K. Mulmuley. Computational Geometry: An Introduction Through Randomized Algorihms. Prentice Hall, Englewood Cliffs, NJ, 1994]. Die Verallgemeinerung der Analyse auf beliebige Punktlage findet man in [M. De Berg, M. Van Kreveld, M. Overmars, O. Schwarzkopf. Computational Geometry. 3. Auflage, Springer, Berlin, 2008]. Es gibt weitere zufallsbasierte Algorithmen.

Weitere wichtige geometrische Graphen sind der Euclidean minimum spanning tree (EMST) [M. I. Shamos. Computational Geometry. Ph.D. thesis, Dept. Comput. Theory Appl., 1:51-64, 1991], der Gabrielgraph [K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. Systematic Zoology, 18:259-278, 1969] und der relative Nachbarschaftsgraph [G. T. Toussaint. The relative neighbourhood graph of a finite planar set. Pattern Recogn., 12:261-268, 1980]. Eine weitere wichtige Triangulierung ist die minimum weight triangulation. [M. T. Dickerson, S. A. McElfresh, and M. H. Montague. New algorithms and empirical findings on minimum weight triangulation heuristics. In Proc. 11th Annu. ACM Sympos. Comput. Geom., pages 238–247, 1995]