Algorithmische Geometrie
(Computational Geometry)

VO: 250048, Mo.,Mi.,Do. 10:00-10:45, 2A310

UE: 250049, Di. 10:00-10:45, 2A310

Andreas Kriegl

Wien, Sommersemester 2010

Inhaltsverzeichnis

Vorbemerkung

Tag 3.01

Beschreibung

Algorithmische Geometrie (Computational Geometry):
Effektive Algorithmen zum Lösen geometrischer Probleme,
siehe wiki(de) und wiki(en)

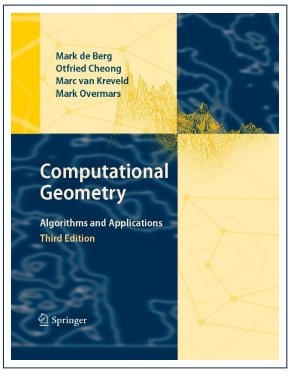
Entstehung

Diese wurde seit den späten 1970'ern entwickelt.

Anwendungen in:

- Computer Graphik
- **G**eographische **I**nformations-**S**ysteme
- **C**omputer-**A**ided-**D**esign and **C**omputer-**A**ided-**M**anufacturing
- Robotik
- Mathematische Visualisierung

Literatur



[dBvKOS97]:

Mark de **Berg**,
Otfried **Cheong**
(**Schwarzkopf**),
Marc van **Kreveld**,
Mark **Overmars**:
Computational Geometry:
Algorithmus and Applications
Springer-Verlag, 1997, 2000,
2008

http://www.cs.uu.nl/geobook

google: Computational_Geometry_algorithms_and_applications_2d_ed_-_De_berg.pdf

Vorkenntnisse

- Elementare Geometrie (und Wahrscheinlichkeitstheorie);
- Sortier-Algorithmen, Such-Algorithmen, balanzierte Such-Bäume.

Inhaltsverzeichnis

5. Orthogonale Rang-Suche (Datenbankabfrage)[dBvKOS97, S.101]
6. Ortsbestimmung [dBvKOS97, S.127]
7. Voronoi Diagramme (Postamtproblem) [dBvKOS97, S.153]
8. Anordnungen und Dualität (Supersampling)[dBvKOS97, S.170]
9. Delaunay Triangulierungen (Höheninterpolation)[dBvKOS97, S.188]
10. Weitere geometrische Datenstrukturen (Windowing)[dBvKOS97, S.216]
11. Konvexe Hüllen [dBvKOS97, S.239]
12. Binäre Raum-Partitionen (Malalgorithmus)[dBvKOS97, S.255]
13. Roboter-Bewegungsplanung [dBvKOS97, S.270]
14. Quadbäume (Netzerzeugung)[dBvKOS97, S.294]
15. Sichtbarkeitsgraphen (Kürzeste Verbindungen) [dBvKOS97, S.310]
16. Simplex-Rang-Suche (Windowing revisited)[dBvKOS97, S.321]

Weitere Links:

- Andreas Kriegl, Uni Wien, 2010
  http://www.mat.univie.ac.at/∼kriegl/LVA.html
- Bernd Gärtner, FU Berlin, 1996
  http://www.inf.ethz.ch/personal/gaertner/texts/agskript/agskript.pdf
- Oliver Karch, Uni Würzburg, 2000
  http://users.informatik.uni-halle.de/∼schenzel/ss07/Uebung-D/vorlesung.pdf
- Olaf Delgado-Friedrichs, Uni Tübigen, 2002
  http://www-ab.informatik.uni-tuebingen.de/teaching/ws02/cg/script-finished.pdf

1. Konvexe Hülle

Tag 3.02

Typische Probleme [dBvKOS97, S.10]

- Bestimmung der nächsten Apotheke liefert eine Unterteilung, ein sogenanntes Voronoi-Diagramm.
- Kürzeste Verbindung mit Hindernissen (Robotik, Routenplaner).
- Schnittpunkte von Kartenüberlagerungen.

Anwendungsbereiche [dBvKOS97, S.19]–[dBvKOS97, S.20]

- Computer Graphik (von Vektorgraphik bis Photorealistische Darstellungen), siehe Kapitel 6, 10 und 16 für $2-D$, Kapitel 12 für $3-D$.
- Robotik (Bewegungs Planung) siehe Kapitel 13 und 15.
- GIS, siehe Kapitel 6, 10 und 16 sowie 2, 7 und 9.
- CAD und CAM, siehe Kapitel 14 und 4.
- Muster Erkennung
- Nicht-geometrische Probleme, siehe Kapitel 5.

- ...

Konvexe Menge K [dBvKOS97, S.11]

Mit $x, y \in K$ ist $\overline{xy} := \{x + \lambda(y - x) : 0 \leq \lambda \leq 1\} \in K$.

Konvexe Hülle einer Menge P [dBvKOS97, S.11]

- Durchschnitt aller konvexen Obermengen
- Kleinste konvexe Obermenge
- Menge aller endlichen Konvexkombinationen $\sum_k \lambda_k p_k$ mit $p_k \in P$, $\lambda_k \geq 0$ und $\sum_k \lambda_k = 1$.
- Gummibandmethode (im Zweidimensionalen).

Konvexes Polygon [dBvKOS97, S.11]

Sei $P \subseteq \mathbb{R}^2$ endlich. Dann ist die konvexe Hülle von P ein konvexes Polygon (beschrieben durch Folge der Ecken) zusammen mit der inneren Komponente.

Naheliegende Algorithmus [dBvKOS97, S.12]

Die Seitenkanten des orientierten konvexen Polygons sind genau jene orientierte Strecken, so daß alle anderen Punkte links von ihnen liegen. Schließlich sortiere man diese. Für Pseudocode, siehe [dBvKOS97, S.12].

Korrektheit [dBvKOS97, S.12]

- Jede Kante des Randes der konvexen Hülle hat diese Eigenschaft.
- Jede Strecke mit dieser Eigenschaft liegt am Rand der konvexen Hülle.

Laufzeit [dBvKOS97, S.13]

- Anzahl der orientierten Strecken ... $n(n-1)$
- Anzahl der Tests pro Strecke ... $n-2$
- Sortieren ... $0 + 1 + 2 + \cdots + (n-1) = \binom{n-1}{2}$ (besser $n \cdot \log(n)$)
- Insgesamt $O(n^3)$.

Degenerierte Fälle [dBvKOS97, S.14]

Was passiert, wenn ein Punkt auf der Verbindungsgerade liegt? - Dann sollte der Test auch erlauben, daß er auf der Verbindungsstrecke liegt.

Robustheit [dBvKOS97, S.14]

Was passiert, wenn Rundungsfehler auftreten:
Z.B. wenn $\overrightarrow{pq}$, $\overrightarrow{qr}$ und $\overrightarrow{pr}$ alle (oder keiner) den Test bestehen.

Verbesserter rekursiver Algorithmus und Pseudocode [dBvKOS97, S.15]

1: Sortiere Punkte nach der x-Koordinate.

2: Bestimme oberen und unteren Rand getrennt jeweils rekursiv.
3: Induktionsschritt für oberen Rand: Füge nächsten Punkt hinzu (dieser gehört zum Rand) und entferne vorige Punkte des alten Randes solange dort ein links-Knick ist.
4: Analog für unteren Rand.
5: Füge beiden Ränder aneinander.

Korrektheit [dBvKOS97, S.16]

- Keine lineare Ordnung, darum Drehung oder lexikographische Ordnung.
- Falls aufeinanderfolgende Kanten kollinear sind, so entferne mittleren Punkt.

Robustheit [dBvKOS97, S.16]

Ergebnis ist ein (fast konvexes) Polygon und alle Punkte liegen (fast) im Inneren. Achtung bei (vertikale)Spitzen.

Tag 3.03

1.1 Theorem [dBvKOS97, S.17]**.**
Die konvexe Hülle von n Punkten kann in $O(n \log(n))$-Zeit bestimmt werden.

Beweis [dBvKOS97, S.17]

Korrektheit durch Induktion nach n. Laufzeit: Lexikographisch sortieren in $O(n \log(n))$-Zeit, for-Schleife in $O(n)$-Zeit, while-Schleife insgesamt $\leq n$-mal, da jedesmal ein Punkt entfernt wird. □

Nachbemerkungen zur Vorgehensweise [dBvKOS97, S.17]

1. Vernachlässige vorerst degenerierte Fälle um den Algorithmus zu finden.
2. Modifiziere den Algorithmus, so daß er auch diese Fälle inkludiert. Besser nicht durch Fallunterscheidungen sondern wenn möglich durch direkte Adaption [dBvKOS97, S.18].
3. Tatsächliche Implementation: U.a. geometrische Eigenschaften in berechenbare Ausdrücke übersetzen (Software libraries).
4. Robustheits Überlegungen (Exakte Arithmetik, Absturz-Vermeidung).

Weiterführende Bemerkungen [dBvKOS97, S.22]

- Obige Algorithmus ist von [Graham, 1972] modifiziert durch [Andrew, 1979].
- Untere Schranke ist $O(n \log(n))$.
- Verbesserung durch [Kirkpatrick and Seidel, 1986] sowie durch [Chan, 1995] auf $O(n \log(h))$ wobei h die Komplexität (Anzahl der Punkte) der konvexen Hülle bezeichnet.

Sortieralgorithmen

Insertions Sort

Sortiere rekursive die Elemente nach und nach in die bereits sortierten ein. Runtime: $O(n^2)$ Animation

Quick Sort

Methode: Teile und Erobere (divide and conquer)!

1. Wähle ein Element (pivot).
2. Teile die übrigen in jene die kleiner und jene die größer sind.
3. Fahre mit diesen Teilen rekursiv fort bis diese 1-elementig sind.

Benötigt $O(n)$ extra Speicherplatz. Ist stabil. Im Mittel $O(n \log n)$ Laufzeit. Im schlechtesten Fall $O(n^2)$ Laufzeit.

Verbesserte Version

Subroutine PARTITION(array, left, right, pivot):

```
1: swap array[pivot] and array[right]
2: newpos = left
3: for i from left to right-1 do
4:     if array[i] < pivot then
           swap array[i] and array[newpos] increment newpos
5: swap array[newpos] and array[right]
6: return newpos
```

Dies bringt array[pivot] an die korrekte Stelle und sortiert die kleineren nach links und die größeren nach rechts. Benötigt nur $O(\log n)$ extra Speicherplatz.

Merge Sort

1. Teile in zwei annähernd gleich große Teile.
2. Verfahre mit den Teilen rekursiv.
3. Sortiere die beiden (sortierten) Teile zusammen. Laufzeit $O(n)$.

Gesamtlaufzeit maximal $O(n \log(n))$ folgt aus $T(n) = 2T(n/2) + n$, siehe Master Theorem:

$$T(n) = a\,T(\frac{n}{b}) + f(n) \text{ mit } f(n) = O(n^{\log_b(a)}),\ a \geq 1,\ b > 1 \Rightarrow$$
$$\Rightarrow T(n) = O(n^{\log_b(a)} \log_b(n)).$$

Zusätzlicher Speicherbedarf $O(n)$.

Binary Tree

Ein Baum wo jeder Knoten höchstens zwei Kinder hat. Dieser heißt balanziert, falls die Tiefe der Teilbäume je zweier Geschwister sich höchstens um 1 unterscheiden. Vorteilhaft für Suche.

Binary Heap (Haufen)

Ein (balanzierter) binärer Baum wo alle Ebenen bis auf die letzte vollständig sind und in der letzte von links beginnend ohne Lücken ist. Jeder Knoten ist größer (oder gleich) als jedes seiner beiden Kinder. Kann in einem Array gespeichert werden. **Einfügen** in einen Heap durch sift-up (hinauf sichten) (Laufzeit $O(\log(n))$): Ans Ende anfügen. Tausche rekursiv wenn nötig mit Eltern

Tag 3.08

Entfernen der Wurzel durch sift-down (Laufzeit $O(\log(n))$): Tausche mit letztem Element und dann rekursiv wenn nötig mit größten Kind. Step by Step Animation. Laufzeit für das **Erzeugen** eines binären Heaps naive $O(n \log(n))$; Verbessert $O(n \sum_{k=0}^{\infty} \frac{h}{2^h}) = O(n)$ durch rekursives Heapifizieren der unteren Teilbäume.

Sei $2^k \leq n < 2^{k+1}$. Anzahl der unteren Teilbäume der Tiefe j ist maximal 2^{k+1-j}. Stamm sift-down braucht jeweils $j-1$. Gesamtschritte

$$\sum_{j=1}^{k}(j-1)2^{k+1-j} = 2^k \sum_{j=0}^{k-1} \frac{j}{2^j} = 2^k(2 - \frac{k+1}{2^{k-1}}) \leq 2^{k+1} \leq 2n.$$

Heap Sort

1. Verwandle Liste in Heap. $O(n)$
2. Bewege Wurzel an den Anfang des sortierten Teils. $O(1)$
3. Verfahre mit unsortierten Teil rekursiv. n-mal sift-down a $O(\log(n))$

Stabil. Laufzeit $O(n \log(n))$. Konstanter zusätzlicher Speicher.

2. Schneiden von Geradensegmenten

Information in Layer[dBvKOS97, S.27]

GIS organisieren die Daten in Layer (Schichten). Jeder Layer für eine Art von Information (z.B. Straßen, Städte, Flüsse, ...,Bevölkerungsdichte, Vegetationstyp,...). Benutzer wählt die gewünschten Kategorien. Wichtig sind die Schnittpunkte der verschiedenen Layer, z.B. Straßen $\cap$ Flüsse = Brücken oder Höhenbereich $\cap$ Vegetationsform.

Schnittpunkte von Geradensegmenten [dBvKOS97, S.28]

Bestimme alle Schnittpunkte von jeweils einem Segment des einen Layers mit einem Segment des anderen. Zwecks Vereinfachung vereinigen wir die beiden Layer zu einem. Suchen also auch Durchschnitte von Segmenten des gleichen Layers. Diese filtern wir nachher aus.

Naiver Algorithmus[dBvKOS97, S.29]

Wähle alle $\binom{n}{2}$ Paare von Strecken und bestimme (so vorhanden) deren Schnittpunkt. Laufzeit ist $O(n^2)$. Nicht verbesserbar, wenn sich alle schneiden.

Verbesserung [dBvKOS97, S.29]

Oft werden sich viele Segmente gar nicht schneiden. Brauchen Verbesserung in Abhängigkeit der Komplexität der Ausgabe (Anzahl der Schnittpunkte). Dazu vermeiden wir jene Segmente zu testen die weit voneinander entfernt sind.

Projektion [dBvKOS97, S.29]

Segmente die disjunkte Projektion auf einer Achse haben scheiden aus. Betrachten nur jene Paare die beide von horizontalen Geraden getroffen werden.

Plane sweep (fegen) algorithm [dBvKOS97, S.30]

Verschieben eine Gerade (sweep line) über die Ebene. Untersuchen den Status, d.h. die sie treffenden Segmente. Dieser ändert sich nur bei Randpunkten der Segmente (Eventpunkte).

Tag 3.10

Testen bei Eventpunkten [dBvKOS97, S.30]

Falls der Eventpunkt ein oberer Randpunkt eines Segments ist, so testen wir dieses auf Durchschnitt mit den anderen Segmenten des Status und fügen es zum Status hinzu. Falls der Eventpunkt ein unterer Randpunkt eines Segments ist, so entfernen wir es aus dem Status.

Das ist nicht gut genug, also verbessern [dBvKOS97, S.30]

Laufzeit ist noch immer $O(n^2)$, z.B. wenn alle Segmente die x-Achse treffen. Sollten nur horizontal nahe liegende testen. Dazu ordnen wir die Segmente des Status nach der x-Koordinate des Schnittes mit der sweep line. Diese Ordnung ändert sich in Schnittpunkten. Wir müssen also auch die Schnittpunkte als Eventpunkte berücksichtigen. Um die Schnittpunkte zu erhalten testen wir in oberen Randpunkten nur benachbarte Segmente des Status auf darunterliegende Schnittpunkte und fügen diese (wenn nicht schon dort) den Eventpunkten hinzu. In diesen Schnittpunkten tauschen wir die Position der Segmente im Status und testen beide auf Schnitt mit ihren neuen Nachbarn. Bei unteren Randpunkten müssen wir deren beiden Nachbarn auf Schnittpunkt prüfen.

Nicht-Degeneriertheitsannahme: [dBvKOS97, S.30]

Segmente nicht horizontal und nicht überlappend.

2.1 Lemma [dBvKOS97, S.31]**.**
Falls sich zwei nicht-horizontale Segmente in einem einzigen Punkt p treffen und kein weiteres Segment durch p geht, so gibt es einen oberhalb liegenden Eventpunkt in welchem die beiden Segmente benachbart sind.

Beweis. Wenn die Sweepline knapp oberhalb p ist, so schneiden die beiden Segmente diese in benachbarten Punkten. Somit existiert ein Eventpunkt q in welchem sie getestet werden. □

Erlauben Degeneriertheit [dBvKOS97, S.32]

Merken uns zu jedem Schnittpunkt alle Segmente die ihm treffen. Überlappung schließen wir nach wie vor aus.

Datenstruktur des Algorithmus [dBvKOS97, S.33]

- Eventqueue (Warteschlange) Q: Eventpoints (gespiegelt) lexikographisch sortiert (horizontale Segmente erlaubt) und in binären Suchbaum gespeichert. Zu jeden Eventpoint die Liste der Segmente, die ihm enthalten (z.B. in einen gemeinsamen oberer Endpunkt). Suchen des Nachfolgers, Einfügen und Entfernen eines Eventpoints benötigt bei hinreichend balanzierten Suchbaum $O(\log n)$ Zeit, siehe AVL-Baum und AVL-Tree.
- Status T: Die geordnete Folge der Segmente welche die Sweepline treffen. Ebenfalls als Blätter in einem (balanzierten) binären Suchbaum gespeichert. Bei den inneren Knoten speichern wir das Segment des am weitesten rechts liegenden Blattes des Teilbaums.
- Suche in T nach unmittelbar links von Eventpunkt liegenden Segment: Folgen in inneren Knoten den li/re Ast falls Segment li/re von dem im Knoten ist. Dann ist das Endblatt oder das unmittelbar links liegende das gesuchte. Suche in T nach unmittelbar rechts von Eventpunkt liegenden Segment ist analog.
- Nachbarsuche somit ebenfalls in $O(\log n)$ Zeit.

Algorithmus FindIntersections [dBvKOS97, S.33]

Input: Liste von Geradensegmenten (Paare von Randpunkten)
Output: Liste von Schnittpunkten jeweils mit Liste der involvierten Segmente.

1: Q := Liste der Endpunkte und für die oberen jeweils mit dem zugehörigen Segment.
2: T := {}
3: **while** Q nicht leer **do**
4: Entnehme nächsten Eventpunkt p von Q und HandleEventPoint(p).

Subroutine HandleEventPoint(p) [dBvKOS97, S.34]

1: Sei $U(p)$ die Liste der Segmente mit oberen Randpunkt p.
2: Finde (benachbarte) Segmente in T die p enthalten und liste diese in $L(p)$ falls p unterer Randpunkt und in $C(p)$ falls p innerer Punkt ist.
3: **if** $L(p) \cup U(p) \cup C(p)$ mindestens 2-elementig **then**
Speichere diesen Schnittpunkt p in Q zusammen mit $L(p)$, $U(p)$, $C(p)$.
4: Entferne die Segmente in $L(p) \cup C(p)$ aus T.
5: Ordne die Segmente aus $U(p) \cup C(p)$ in T ein, nach ihren Schnittpunkt mit einer unmittelbar unterhalb liegender Sweepline. // Die Ordnung von $C(p)$ wir dadurch umgekehrt

6: **if** $U(p) \cup C(p) = \emptyset$ **then**
FindNewEvent(s_-,s_+,p) mit linken und rechten Nachbarn s_- und s_+.
7: **else**
8: FindNewEvent(s_l,s',p) mit dem linkesten Segment s' von $U(p) \cup C(p)$ und dessen linken Nachbarn s_l in T.
9: FindNewEvent(s",s_r,p) mit dem rechtesten Segment s'' von $U(p) \cup C(p)$ und dessen rechten Nachbarn s_r in T.

Subroutine FindNewEvent(s_l,s_r,p) [dBvKOS97, S.35]

1: **if** s_l und s_r treffen sich lexikographisch nach p (d.h. unterhalb oder auf gleicher Höhe rechts) und der Schnittpunkt q ist noch nicht in Q **then**
Füge q zu Q hinzu.

2.2 Lemma. Korrektheit [dBvKOS97, S.35]**.**
Der Algorithmus FindIntersections *findet alle Schnittpunkte mit zugehörigen Segmenten korrekt.*

Beweis. Induktion nach der Reihenfolge der Eventpunkte: Sei p ein Eventpunkt (Rand- oder Schnittpunkt), sodaß alle vorigen Schnittpunkte richtig bestimmt wurden. Sei $U(p)$, $L(p)$ und $C(p)$ wie oben. Falls p ein Randpunkt eines Segments ist, dann ist $p \in Q$ beim Start und die Segmente in $U(p)$ sind bereits gespeichert. Die Segmente von $L(p)$ und $C(p)$ sind in T zum Zeitpunkt p, werden also in Zeile 2 von HandleEventPoint identifiziert. Somit ist der Algorithmus in diesem Fall auch bei p korrekt. Falls p kein Randpunkt ist, so müssen wir zeigen, daß p irgendwann nach Q kommt. Seien s' und s'' zwei benachbarte Segmente durch p. Nach Lemma 2.1 (verallgemeinert) werden diese benachbart in T in einem Eventpunkt vor p, also ihr Schnittpunkt p gefunden und damit zu Q hinzugefügt. □

2.3 Lemma. Laufzeit [dBvKOS97, S.36]**.**
Die Laufzeit von FindIntersections *bei n Segmenten und i Schnittpunkten ist* $O((n+i)\log n)$

Beweis.

- Erzeugen von Q (balanzierter binär-Baum) benötigt $O(n \log n)$ Zeit.
- Events:
  - Delete p von Q in Zeile 3 von FindIntersections in $O(\log n)$ Zeit.
  - 1-2 Aufrufe von FindNewEvent in Zeile 6,8,9 von HandleEventPoint fügen maximal 2 Eventpunkte zu Q hinzu (á $O(\log n)$ Zeit)
  - Einfügen, Löschen, Nachbarn finden für T in Zeile 4,5,6,8,9 von HandleEventPoint (á $O(\log n)$). Anzahl der Operationen ist linear in der Anzahl $m(p)$ von $L(p) \cup U(p) \cup C(p)$.
  - Sei $m := \sum_{p \in Q} m(p)$, dann ist die Gesamtlaufzeit $O(m \log n)$.
  - Klarerweise ist $m = O(n+k)$, k die Größe des Outputs, denn Segmente die bei anderen als den beiden Eventpunkten auftreten werden dort in Output ausgegeben. Wollen aber $m = O(n+i)$: Betrachte Segmente als Graph,

$m(p) \leq \deg p \Rightarrow m \leq \sum_p \deg(p) = 2n_e$, wo n_e die Kanten. Höchstens $n_v \leq 2n + i$ viele Ecken. Mittels Euler's Formel (siehe 🌍) erhalten wir (siehe [dBvKOS97, S.36]):

$$2 \leq n_v - n_e + n_f \leq 2n + i - n_e + \frac{2}{3}n_e \Rightarrow m \leq 2n_e \leq 2 \cdot 3(2n + i - 2).$$

□

Beweis der Eulerformel. Induktion nach der Anzahl der Kanten n_e.
$(n_e = 0) \Rightarrow n_f = 1$ und $n_v \geq 1$ (Graph$\neq \emptyset$) $\Rightarrow n_v - n_e + n_f \geq 2$. $(n_e > 0)$ Entferne eine Kante $n_e \to n_e - 1$. Falls diese zwei Endpunkte hat, so identifiziere diese. Also dann $n_v \to n_v - 1$. Andernfalls schließt sie eine Fläche ein. Also $n_f \to n_f - 1$. Dies zeigt auch, daß wenn der Graph zusammenhängend ist, so gilt Gleichheit. □

Tag 3.11

2.4 Theorem. Speicherbedarf [dBvKOS97, S.37]**.**
Der Speicherbedarf von FindIntersections *ist* $O(n)$ *bei* n *Segmenten.*

Beweis.

- In T kommt jedes Segment höchstens 1x, also $O(n)$ Speicher.
- $Q = O(n + i)$, aber wollen besseres: Speichern nur jene Schnittpunkte von gerade benachbarten Segmenten. Damit ist der Speicherbedarf O(n). Allerdings müssen Schnittpunkt entfernt werden, wenn ihre Segmente nicht mehr benachbart sind, was nicht an der Laufzeit-Abschätzung ändert. Diese Schnittpunkte werden automatisch rechtzeitig wieder aufgenommen.

□

Binäre Suchbäume

Binärer Suchbaum 🌐 🌐 📄 🌍 j-Algo

Ein binärer Baum in welchem der Wert jedes Knoten größer aller Werte im linken Teilbaum und kleiner (oder gleich) aller Werte im rechten Teilbaum ist. Die Höhe h eine Baums ist die maximale Länge seiner Zweige.

Suche im binären Suchbaum

Ausgehend vom Stamm gehe jeweils zum linken/rechten Kind eines Knotens, wenn der Suchwert kleiner/größer als jener des Knoten ist. Benötigt maximal $O(h)$-Zeit. Ziel ist es also h möglichst klein zu halten (Balanziertheit).

Adelson-Velsky Landis-Baum 🌐 🌐

Die Höhe des linken und des rechten Astes unterscheiden sich höchstens um 1. Dementsprechend markieren wir den Knoten mit $\swarrow$, $\leftrightarrow$ und $\searrow$, je nachdem welcher Teilbaum länger ist.

Höhenabschätzung $h = O(\log n)$ [Jia09, S.42]

(Fibonacci-)Bäume F_h der Höhe h mit minimal vielen n Knoten erhalten wir rekursiv durch Anhängen von F_{h-2} und F_{h-1} an den neuen Stamm. Sei n_h die Anzahl der Knoten von F_h, d.h. $n_1 = 1$, $n_2 = 2$ und rekursiv $n_h = n_{h-2} + 1 + n_{h-1}$. Setzen wir $f_h := n_{h-2} + 1$ so ist $f_3 = 2$, $f_4 = 3$ und $f_{h+2} = f_h + f_{h+1}$, also f_h die Fibonacci-Folge mit $f_0 = 0$ und $f_1 = 1$.

Einfügen und Löschen

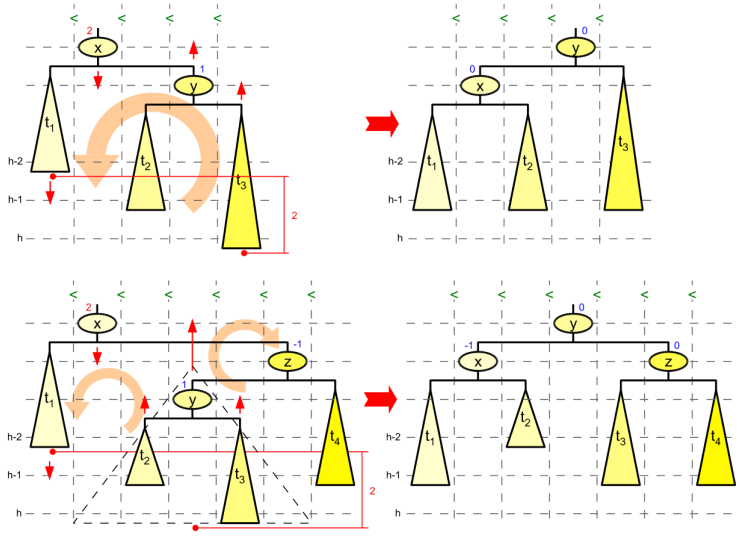
Einfügen: Suche nach dem einzufügenden Element im Baum. Falls dieses nicht gefunden wird, so füge man es als entsprechendes Kind and den letzten Knoten an. **Entfernen**: Falls dieser Knoten ein Blatt ist, so entferne ihn. Besitzt er genau ein Kind, so ersetze ihn durch dieses. Besitzt er 2 Kinder, so ersetze ihn durch das größte Element des linken Teilbaums. Da dieses seinerseits höchstens ein Kind hat kann es nach dem zuvor Gesagten entfernt werden.

Rebalanzieren

Dazu verfolgen wir den Ast beginnend beim eingefügten oder gelöschten Blatt Richtung Stamm bis wir auf einen Knoten x stoßen, welcher die AVL-Bedingung verletzt. Sei y dessen Kind mit dem längeren Teilbaum und z das Kind von y mit dem längeren Teilbaum. Falls y und z beide linke (oder beide rechte) Kinder sind, so genügt eine einfache "Rotation" um die AVL-Bedingung wiederherzustellen. Andernfalls benötigen wir eine doppelte "Rotation". Im Fall des Einfügens ist damit die AVL-Bedingung des gesamten Baums wiederhergestellt, da sich height(x) dann nicht ändert. Im Falle des Löschens kann height(x) um 1 abnehmen und wir müssen den Ast weiter verfolgen. Die Laufzeit ist jedenfalls $O(h)$.

Tag 3.15

Restrukturieren, einfache und doppelte Rotation [Jia09, S.20]



Überlagerungen von Zerlegungen

Gemeinsame Verfeinerung [dBvKOS97, S.37]

Wir wollen zwei Unterteilungen überlagern, also die gemeinsame Verfeinerung bestimmen.

Unterteilung [dBvKOS97, S.38]

Betrachten Unterteilung der Ebene durch lineare Graphen (mit Ecken und (offenen) Kanten). Flächen der Unterteilung sind Zusammenhangskomponenten des Komplements, also (möglicherweise unbeschränkte) offene "polygonale Bereiche" mit Kanten und Ecken der Unterteilung als Rand. Komplexität der Unterteilung ist die Summe der Anzahlen der Ecken, der Kanten und der Flächen.

Methoden [dBvKOS97, S.39]

Wir wollen z.B. den Rand einer Fläche orientiert durchlaufen können, oder die (beiden) an eine Kante angrenzenden Flächen bestimmen, oder alle Kanten mit gegebener Ecke bestimmen können.

Doppelt-verbundene Liste [dBvKOS97, S.39]

Diese soll Eintragungen für jede Ecke, jede (orientierte) Kante und jede Fläche enthalten; und für jeden Eintrag einerseits Attribut-Informationen und andererseits geometrisch-topologische Informationen.

- Zu jeder Kante $\vec{e}$ haben wir den umgekehrt orientierten Zwilling Twin($\vec{e}$), sowie die links liegende Fläche IncidenceFace($\vec{e}$) und Next($\vec{e}$) und Prev($\vec{e}$) für die nächste bzw. vorige Kante des Randes von IncidenceFace($\vec{e}$). Sowie einen Pointer zum Anfangspunkt Origin($\vec{e}$) der Kante.
- Achtung: Der Rand muß nicht zusammenhängend sein (bei Löchern in der Fläche) darum orientiert man den Rand bei Löchern umgekehrt. Für jede Fläche f benötigen wir Pointer in jede Zusammenhangskomponente des Randes: (OuterComponent(f) in die äußere oder nil falls unbeschränkt) und die Liste InnerComponents(p) der übrigen. Annahme: Keine isolierten Ecken.
- Für jede Ecke v speichern wir die Koordinaten und einen Pointer IncidenceEdge(v) zu einer Kante mit diesem Anfangspunkt.

Der Speicherbedarf hierfür ist linear in der Komplexität. Achtung die Länge von InnerComponents(f) ist nicht konstant in f, aber jede orientierte Kante kann höchstens für ein f durch InnerComponents(f) referenziert werden.

Tag 3.17

Bemerkung [dBvKOS97, S.41]

Falls die Ecken keine Attribut-Information haben, können wir sie weglassen. Wir dürfen auch annehmen, daß der Graph zusammenhängend ist, andernfalls verbinden wir die Komponenten durch Kanten und dann sind die InnerComponents unnötig.

Gemeinsame Verfeinerung zweier Unterteilungen [dBvKOS97, S.41]

Wir wollen aus zwei doppelt-verbundenen Unterteilungen S_1 und S_2 die gemeinsame Verfeinerung $O(S_1, S_2)$ bestimmen, die als Flächen f die Zusammenhangskomponenten von $f_1 \cap f_2$ für beliebige Flächen $f_i \in S_i$ hat. Dabei soll zu jede ihrer Flächen $f = f_1 \cap f_2$ die zugehörigen Flächen f_1 und f_2 zwecks Attribut-Information ausgegeben werden.

Wiederverwertung von Information [dBvKOS97, S.42]

Alle Kanten bis auf die sich-schneidenden können weiter verwendet werden. Wir beginnen also mit der Vereinigung der Kantenlisten von S_1 und S_2 als neue Kantenliste. Ziel ist es daraus eine gültige Unterteilung zu machen.

Schneidende Kanten [dBvKOS97, S.42]

Verwenden den Plane Sweep Algorithmus für $S_1 \cup S_2$. Neben den dort beschriebenen Daten T und Q sei D die Vereinigung der beiden gegebenen doppelt-verbundenen Listen. Bei Eventpunkten p wird T und Q upgedated. Falls p nur auf Kanten einer Zerlegung liegt ist nichts weiter zu tuen. Andernfalls zerlege Kanten e in die beiden Teile und adaptiere die Twin, Next und Prev Pointer (Details siehe [dBvKOS97, S.43]). Braucht alles konstante Zeit, bis auf Lokalisieren der Nachbarn der Kante $\vec{e}$ wofür lineare Zeit im Grad m der Ecke benötigt wird.

Laufzeit [dBvKOS97, S.44]

Laufzeit für Bestimmung der Kanten-Daten ist $O((n+k)\log n)$ wobei k die Komplexität des Overlays ist, siehe Lemma 2.3.

Flächenbestimmung [dBvKOS97, S.44]

Anzahl der Flächen ist Anzahl der äußeren Ränder + 1. Diese Zyklen werden an Linksknick bei der linkesten Ecke erkannt.

2.5 Lemma [dBvKOS97, S.45]**.**
Jeder Zykel definiert eine Ecke eines neuen Graphens G. Eine weitere Ecke steht für den Rand bei ∞. Kante von C nach C' in G, falls eine Kante von C unmittelbar links von der linkesten Ecke von C' und C' ein Loch (als im Uhrzeigersinn orientiert ist) beschreibt. Die (initialen Ecken der) Zusammenhangskomponenten von G entsprechen genau den Zyklen welche äußerer Rand eine Fläche sind.

Beweis. Offensichtlich begrenzen die Zyklen, welche Ecken einer Zusammenhangskomponente von G entsprechen, die gleiche Fläche. Es kann also höchstens einen äußeren Zykel pro Zusammenhangskomponente geben. Umgekehrt beschreibt jede initiale Ecke offensichtlich einen äußeren Rand. □

Tag 3.18

Bestimmung von G [dBvKOS97, S.46]

Wir brauchen also für die Flächendaten des Overlays nur G zu bestimmen: Ecken in $O(k)$-Zeit sind klar. Kanten indem wir die ursprünglichen Kanten unmittelbar links von der linkesten Ecke beim Plane-Sweep bestimmen. Dazu Pointer von allen ursprünglichen Kanten zu der zugehörigen Ecke in G in $O(n)$-Zeit. Flächenbestimmung benötigt also zusätzlich zum plane-sweep $O(n+k)$ Zeit.

Pointer auf ursprüngliche Flächen [dBvKOS97, S.46]

Wir wählen eine Ecke der Fläche. Falls die angrenzenden Randkanten von beiden Zerlegungen stammen, sind die ursprünglichen Flächen klar. Andernfalls nur eine Fläche klar und wir brauchen noch die Fläche der anderen Zerlegung welche die Ecke enthält. Dies wird uns auch durch den plane-sweep geliefert! Dies benötigt also insgesamt $O\big((n+k)\log n\big)$-Zeit nach Lemma 2.3.

Zusammenfassung: Algorithmus MapOverlay [dBvKOS97, S.46]

Input: Zwei Zerlegungen S_i
Output: Overlay von S_1 und S_2

1: Kopiere die doppel-verbundenen Listen von S_1 und S_2 nach D. // $O(n)$
2: Berechne die Kantenschnitte mittels plane sweep. Zusätzlich:
   - Verändere D wie oben beschreiben, falls sich Kanten von verschiedenen S_i schneiden.
   - Speichere die unmittelbar links von Eventpunkt liegende Kante zu dieser Ecke in D.

   // $O((n+k)\log n)$ nach 2.3. // Nun ist D eine doppelt-verbundene Liste mit Ausnahme der Flächen.
3: Bestimme die Randzyklen durch Abgehen von D. // $O(k)$
4: Konstruiere den Graphen G und seine Zusammenhangskomponenten. // $O(k)$
5: **for** Zusammenhangskomponenten von G **do** // $O(k)$
   Sei C der eindeutige äußere Randzykel der Komponente und f die zugehörige Fläche. Erzeuge einen record für f. Setze OuterComponent(f) und konstruiere die Liste InnerComponent(f). Setze die IncidenceFace pointers der involvierten Kanten.
6: Beschrifte jede Fläche mit den Namen der Flächen in S_1 und S_2 die sie als Durchschnitt beschreiben. // $O\big((n+k)\log n\big)$

2.6 Theorem. Laufzeit [dBvKOS97, S.47]**.**
Seien s_i Unterteilungen der Komplexität n_i, weiters $n = n_1+n_2$ und k die Komplexität des Overlays. Dieses kann in $O\big((n+k)\log n\big)$-Laufzeit bestimmt werden.

Beweis. Zeile 1 braucht $O(n)$ Zeit; Zeile 2 braucht $O\big((n+k)\log n\big)$ Zeit nach 2.3. Zeile 3–5 braucht $O(k)$ Zeit. Zeile 6 braucht $O\big((n+k)\log n\big)$ Zeit. □

Boolsche Operatoren [dBvKOS97, S.47]

Um Boole'sche Operationen auf Polygone anzuwenden betrachten wir diese als Unterteilungen und filtern aus dem Overlay die (polygonalen) Flächen mit der entsprechenden Operation heraus.

2.7 Folgerung [dBvKOS97, S.48]**.**
Seien P_i einfach geschlossene Polygone mit n_i Ecken. Sei $n = n_1 + n_2$, dann läßt sich $P_1 \cap P_2$, $P_1 \cup P_2$ und $P_1 \setminus P_2$ in $O\big((n+k)\log n\big)$ Laufzeit bestimmen, wobei k die Komplexität des Outputs ist.

Nachbemerkungen [dBvKOS97, S.48]

Der Algorithmus zur Overlay-Bestimmung mit $O\big((n+k)\log n\big)$ Laufzeit stammt von [BO79]. Kann zu $O(n \log n + k)$ verbessert werden, siehe [Bal95]. Die Reduktion des Speicherbedarfs von $O(n+k)$ auf $O(n)$ stammt von [PS91].

3. Polygon-Triangulierungen (Galerieüberwachung)

Galerieüberwachung [dBvKOS97, S.52]

Wieviele drehbare Kameras sind notwendig um alle Wände einer Gallerie zu überwachen und wo müssen diese positioniert werden? Der Einfachheit halber modellieren wir die Wände der Gallerie als einfach geschlossenes Polygon, also keine unzusammenhängenden Wände (wie z.B. Säulen). Kamerapositionen sind Punkte p im Inneren K des Polygons und der jeweils überwachte Bereich ist die maximale sternförmige Teilmenge mit diesen Zentrum, d.h. $\{x : \overline{xp} \subseteq K\}$. Wollen die Anzahl der nötigen Kameras als Ausdruck in der Anzahl der Ecken des schlimmsten Polygons (bei konvexen Polygonen reicht eine Kamera). Das (schwierigere) Problem zu einem gegebenen Polygon die minimale Anzahl der Kameras zu finden ist **NP-hart**.

P=NP ?

Komplexitätsklassen

Probleme der Klasse P sind jene Entscheidungsprobleme, welche durch einen Algorithmus (genauer eine deterministische Turingmaschine) in polynomialer Zeit, d.h. in $O(n^k)$-Schritten für ein $k \in \mathbb{N}$ in Abhängigkeit von der Eingabelänge n, lösbar sind.
Probleme der Klasse NP sind jene, deren Lösungen in polynomialer Zeit durch einen Algorithmus überprüft werden können, z.B. subset sum problem.
NP-vollständige Probleme sind die "schwierigsten" in NP, also jene Probleme in NP auf die sich jedes andere in polynomialer Zeit reduzieren läßt, z.B. Erfüllbarkeitsproblem der Aussagenlogik Satz von Cook(-Levin), Liste, Lateinische Quadrate & Sudokus, Problem des Handlungsreisenden, Rucksackproblem (Maximierung mit Nebenbedingung).
NP-harte Probleme sind jene (nicht notwenig in NP liegende Probleme), für die ein NP-vollständiges Problem existiert welches auf sie in polynomialer Zeit reduzierbar ist (und somit auch jedes Problem aus NP), z.B. sind alle NP-vollständigen auch NP-hart. Das Halteproblem ist NP-hart aber nicht in NP, da algorithmisch nicht entscheidbar [Tur36]. Siehe auch .

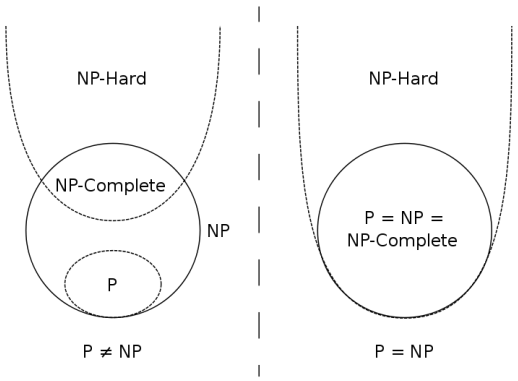
Beweis, siehe auch . Wäre das Halteproblem algorithmisch entscheidbar, so auch das Selbstanwendbarkeitsproblem und dann könnten wir das Programm P_0 betrach-

ten, welches auf ein P genau dann anwendbar ist, wenn P nicht auf sich selbst anwendbar ist. Dieses P_0 kann nun weder auf sich anwendbar noch nicht anwendbar sein. □

Folgerung. Schwache Version von Gödels Unvollständigkeitssatz.
Die gültigen Sätze (der Aussagenlogik 1. Stufe) über die natürlichen Zahlen lassen sich nicht vollständig axiomatisieren. Vgl. [Göd31].

Beweis. Andernfalls könnten wir die gültigen Sätze (= herleitbaren Sätze) algorithmisch aufzählen. Die Aussage, daß P auf i anwendbar ist, kann als Aussage über Paare natürlicher Zahlen aufgefaßt werden, also kommt entweder sie oder ihre Negation in der Liste der gültigen Sätze vor, und somit können wir sie entscheiden. □

Tag 3.22



Millenium Prize Problem
(von 1 Million US-Dollar)
des Clay Institutes

Triangulierung

[dBvKOS97, S.53] Wir versuchen des Innere des Polygons zu triangulieren, d.h. in Dreiecke zu zerlegen, denn jedes Dreieck kann offensichtlich durch eine Kamera überwacht werden.

3.1 Theorem [dBvKOS97, S.54]**.**
Jedes einfach geschlossene Polygon mit n Ecken besitzt eine Triangulierung mit $n-2$ Dreiecken.

Beweis. Induktion nach n. $(n = 3)$ ist trivial. $(n > 3)$ eine Diagonale existiert, denn sei v die lexikographisch kleinste Ecke und v_- und v_+ seine beiden Nachbarn. Falls $\overline{v_-v_+}$ in Inneren des Polygons liegt so ist es eine Diagonale, andernfalls ist die Strecke von v zu der am weitesten von $\overline{v_-v_+}$ entfernten Ecke im Dreieck vv_-v_+ eine Diagonale. Nun wenden wir die Induktionsannahme auf die beiden Teilpolygone mit n_1 und $n - n_1 + 2$ viele Ecken und erhalten somit eine Triangulierung mit $(n_1 - 2) + (n - n_1 + 2 - 2) = n - 2$ vielen Dreiecken.

Daß jede Triangulierung aus genau $n - 2$ Dreiecken besteht, folgt ebenso da obige Rechnung für jede gewählte Diagonale funktioniert. □

Folgerung [dBvKOS97, S.54]**.**
Jedes einfach geschlossene Polygon mit n Ecken kann durch $n-2$ Kameras (im Inneren der Dreiecke plaziert) überwacht werden. Wenn wir die Kameras auf Diagonalen plazieren, sollten wir mit $\lceil \frac{n-2}{2} \rceil$ auskommen. Noch besser ist sie in den Ecken zu plazieren. Dazu brauchen wir eine Auswahl von Ecken, sodaß jedes Dreieck eine Ecke in dieser Menge hat. Dazu färben wir die Ecken mit 3 Farben, so daß die Randpunkte von Kanten der Triangulierung verschiedene Farben haben. Die $\lfloor n/3 \rfloor$ Ecken (der seltensten) Farbe können dann als Kamerapositionen verwendet werden. Besser geht es nicht [dBvKOS97, S.55]*, wie das Beispiel eines Kammes mit $\lfloor n/3 \rfloor$ Zähnen zeigt.*

Existenz einer 3-Färbung [dBvKOS97, S.54]

Betrachten dualen Graphen $\mathcal{T}^d$, d.h. jeder (beschränkten) Fläche f von $\mathcal{T}$ entspricht eine Ecke f^d von $\mathcal{T}^d$. Gemeinsame Kanten (also in unserem Fall Diagonalen) von f_1 und f_2 entsprechen Kanten zwischen f_1^d und f_2^d. Es ist $\mathcal{T}^d$ ein Baum (d.h. keine geschlossenen Wege), denn Entfernung eine Kante läßt ihm zerfallen. Tiefensuche: Beginnen bei einer Ecke f^d von $\mathcal{T}^d$ und Färben die Ecken von f. Wenn wir f_2^d von f_1^d über eine Kante erreichen, dann haben f_1 und f_2 eine gemeinsame Kante und somit die Farbe für den 3. Punkt klar. Da $\mathcal{T}^d$ ein Baum ist, sind die anderen mit f_2^d verbundenen Ecken noch nicht besucht worden.

Tiefensuche – Depth-first Search

Algorithmus DFS(v)

1: Markiere v als besucht.
2: **for** Kante e von v zu unbesuchten v' **do**
   DFS(v')
3: Markiere v als erledigt.

Benötigt $O(n_e + n_v)$-Laufzeit (besucht eventuell für jeden Knoten alle Kanten) und $O(n_e)$-Speicherplatz (für die Ecken am aktuellen Suchpfad und die bereits gefundenen).

3.2 Art Gallery Theorem [dBvKOS97, S.55]**.**
Eine einfach geschlossenes Polygon mit n-Ecken kann durch $\lfloor n/3 \rfloor$ Kameras überwacht werden, und im allgemeinen nicht mit weniger. □

Effektiver Algorithmus [dBvKOS97, S.55]

Brauchen eine effektiven Algorithmus der die Triangulierung (als doppelt-verbundene Liste) liefert. Wir werden im Folgenden zeigen, daß dies in $O(n \log n)$-Zeit geht. Danach benötigen wir lineare Zeit (für die Tiefensuche) um die Färbung vorzunehmen. Somit folgt dann:

3.3 Theorem [dBvKOS97, S.56]**.**
Eine Liste von $\lfloor n/3 \rfloor$ Kamera-Positionen zur vollständigen Überwachung eines einfach geschlossenes Polygon mit n-Ecken kann in $O(n \log n)$-Zeit bestimmt werden.

Partitionieren eines Polygons in monotone Teile

Finden einer Diagonale [dBvKOS97, S.56]

FindDiagonal

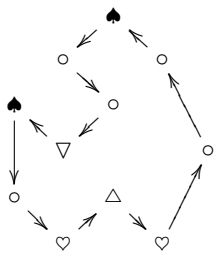
1: Suche linkeste (genauer lexikographisch kleinste) Ecke, $O(n)$.
2: Bestimme Nachbar Ecken, $O(1)$.
3: Finde wenn nötig passende Ecke im Inneren des so erhaltenen Dreiecks, $O(n)$

Rekursionsschritt für Bestimmung der Triangulierung: Spalte Polygon längs der Diagonale in zwei Teile. Schlimmstenfalls $n \to (3, n-1)$, also insgesamt $O(n^2)$-Zeit. Bei konvexen Polygonen hingegen geht das Triangulieren in $O(n)$-Zeit. Idee: Polygon in konvexe Teile spalten – aber das ist genauso schwierig wie das Triangulieren. Darum spalten wir es in sogenannte monotone Teile. Ein Polygon heißt **monoton** bzgl. y-Geraden (kurz y-monoton) wenn die orthogonalen Geraden das innere des Polygons in zusammenhängenden Mengen schneiden. Dies ist genau dann der Fall wenn wir von der höchsten Ecke zur niedrigsten niemals ein Schritt hinauf machen. Ein **Umdrehpunkt** (turn vertex) ist eine Ecke wo die Kantefolge zwischen auf und ab wechselt. Diese müssen wir loswerden indem wir dort eine Diagonale einfügen. Verwenden der lexikographische Ordnung statt auf/ab vermeidet Degeneriertheitsprobleme.

Eckenklassifizierung [dBvKOS97, S.57]

- ♠ Start-Ecke ... beide Nachbarn unterhalb und links-Knick (Innere ist unten).
- △ Split-Ecke ... beide Nachbarn unterhalb und rechts-Knick (Innere ist oben).
- ◯ Reguläre-Ecke ... ein Nachbar unterhalb und einer oberhalb.
- ▽ Merge-Ecke ... beide Nachbarn oberhalb und rechts-Knick (Innere ist unten).
- ♡ End-Ecke ... beide Nachbarn oberhalb und links-Knick (Innere ist oben).

Wenden plane sweep an. In Split-Ecken wird der Algorithmus einen Teil des Polygons abspalten und in Merge-Ecken vereinen.



3.4 Lemma [dBvKOS97, S.57]**.**
Polygone sind genau dann y-monoton, wenn sie weder Split- noch Merge-Ecken besitzen.

Beweis. ($\Rightarrow$) denn bei einer Split- oder Merge-Ecke liegen benachbart zwei Zusammenhangskomponenten.

($\Leftarrow$) Falls es nicht y-monoton ist, so existiert eine sweepline mit nicht zusammenhängenden Durchschnitt mit dem Inneren. O.B.d.A. ist die linkeste Zusammenhangskomponente des Durchschnitts kein isolierter Punkt. Seien $p < q$ die beiden Randpunkte

dieser Komponente. Wir folgen dem Polygon von q an (aufwärts) bis wir wieder auf die Sweepline in einen Punkt r treffen, siehe [dBvKOS97, S.58]. Falls $r > q$ so ist der höchste Punkt dazwischen eine Split-Ecke (und keine Start-Ecke), andernfalls ist $r = p$ und wir finden einen Merge-Punkt von q aus abwärts. □

Tag 3.24

Entfernen von Split-Punkten [dBvKOS97, S.58]

Speichern die Ecken (Eventpoints) in einer event queue lexikographisch geordnet nach der Höhe in $O(n \log n)$-Zeit. Sei $\triangle$ eine Split-Ecke und e_- und e_+ die benachbarten Kanten auf der sweepline. Wählen die Diagonale zu dem niedrigsten Punkt zwischen e_- und e_+ oberhalb der sweepline so dieser existiert, oder den oberen Endpunkt von e_- oder von e_+. Mit helper(e) bezeichnen wir die niedrigste Ecke oberhalb der sweepline für welche das horizontale Segment welches sie mit der Kante e verbindet im Polygon liegt.

Entfernen von Merge-Punkten [dBvKOS97, S.59]

Sei $v = \triangledown$ ein Merge-Punkt und e_- und e_+ die benachbarten Kanten auf der sweepline. Beachte, daß $v = \text{helper}(e_-)$. Sobald wir auf eine Ecke v' stoßen, die neuer helper(e_-) wird, können wir $\overline{vv'}$ als Diagonale verwenden. D.h. wann immer der Helfer einer Kante e_- sich ändert, so verbinden wir die beiden Helfer durch eine Diagonale falls der alten Helfer eine Merge-Ecke oder der neue eine Split-Ecke ist. Falls ein Helfer sich nicht mehr ändert, so verbinden wir ihn mit dem unteren Endpunkt von e_-.

Implementierung [dBvKOS97, S.59]

Status: Speichern die Kanten welche die Sweepline treffen zusammen mit ihren jeweiligen Helfer in einem binären Suchbaum T und (da wir nur an Kanten links von Split- und Merge-Ecken interessiert sind) zwar nur jene, die das Innere des Polygons rechts (im Sinne der lexikographischen Ordnung) haben. Speichern die (Teil-)Polygone in doppelt-verbundener Liste D. Weiters Pointer zwischen Kanten in T und in D.

Algorithmus MakeMonotone(D) [dBvKOS97, S.60]

Input: Ein einfach geschlossenes Polygon als doppelt-verbundene Kantenliste D.
Output: Eine Zerlegung in monotone Teilpolynome (gespeichert in D)
Sortiere die Ecken lexikographisch in eine Queue Q.
Binäre Suchbaum $T := \{\}$.
while $Q \neq \emptyset$ **do**
Entferne die oberste Ecke v von Q.
Rufe die den Typ von v entsprechende Routine HandleSplitVertex, HandleMergeVertex, HandleStartVertex, HandleEndVertex, HandleRegularVertex auf.

Subroutine HandleStartVertex(v) [dBvKOS97, S.60]

1: Füge die bei v startende Kante e in T ein und setze helper$(e) := v$.

Subroutine HandleEndVertex(v) [dBvKOS97, S.60]

1: Sei e die bei v endende Kante.
2: **if** helper(e) ist MergeEcke **then**
   Füge die Diagonale von v nach helper(e) in D ein.
3: Entferne e von T.

Subroutine HandleSplitVertex(v) [dBvKOS97, S.61]

1: Suche den linken Nachbarn e_- von v in T.
2: Füge die Diagonale von v nach helper(e_-) zu D hinzu.
3: helper$(e_-) := v$.
4: Füge die bei v startende Kante e in T ein und setze helper$(e) := v$.

Subroutine HandleMergeVertex(v) [dBvKOS97, S.61]

1: Sei e' die bei v endende Kante.
2: **if** helper(e') ist Merge-Ecke **then**
3: Füge die Diagonale von v nach helper(e') zu D hinzu.
4: Entferne e' von T.
5: Suche den linken Nachbarn e_- von v in T.
6: **if** helper(e_-) ist Merge-Ecke **then**
7: Füge die Diagonale von v nach helper(e_-) zu D hinzu.
8: helper$(e_-) := v$.

Subroutine HandleRegularVertex(v) [dBvKOS97, S.61]

1: **if** Innere des Polygons liegt rechts von v **then**
2: Sei e' die bei v endende Kante.
3: **if** helper(e') ist Merge-Ecke **then**
4: Füge die Diagonale von v nach helper(e') zu D hinzu.
5: Entferne e' von T.
6: Füge die bei v startende Kante e in T ein und setze helper$(e) := v$.
7: **else**
8: Suche den linken Nachbarn e_- von v in T.
9: **if** helper(e_-) ist Merge-Ecke **then**
10: Füge die Diagonale von v nach helper(e_-) zu D hinzu.
11: helper$(e_-) := v$

3.5 Lemma [dBvKOS97, S.61]**.**
Der Algorithmus MakeMonotone *zerlegt das Polygon durch Hinzufügen von sich nicht-schneidenden Diagonalen in y-monotone Teil-Polygone.*

Beweis. Klarerweise besitzen die Teil-Polygone keine Split-Ecken. Auch die Merge-Ecken v wurden alle entfernt sobald sich $\text{helper}(e_-) = v$ ändert. Also sind sind diese Polygone nach Lemma 3.4 monoton. Bleibt zu zeigen das jede hinzugefügte Diagonale weder Kanten des Polygons noch zuvor bestimmte Diagonalen echt schneidet. Für HandleSplitVertex (und analog für andere): Sei $\overline{v'v}$ die aktuelle Diagonale und e_- die links benachbarte und e_+ die rechts benachbarte Kante von v. Somit ist $\text{helper}(e_-) = v'$ wenn wir v erreichen. Die Endpunkte von e liegen außerhalb des Vierecks Q mit den Seiten e_-, e_+ und den Horizontalen durch v und v'. Angenommen eine Kante e träfe die Diagonale und somit auch eine der beiden horizontalen von e_- nach v oder v', was ausgeschlossen ist. Jede zuvor bestimmte Diagonale muß beide Endpunkte oberhalb von v und nicht in Q haben und trifft nach den eben gezeigten e_- und e_+ nicht, kann also auch die Diagonale nicht treffen. □

Laufzeit [dBvKOS97, S.62]

- Erzeugen von Q aus D: $O(n \log n)$.
- Erzeugen von $T = \emptyset$: $O(1)$.
- In jedem (der n) Eventpunkt(e): Eine Operation in Q, höchstens eine Nachbarbestimmung, ein Einfügen und ein Entfernen von T (á $O(\log n)$). Höchstens 2 Diagonalen einfügen ($O(1)$).

Insgesamt also $O(n \log n)$ Laufzeit.

Speicherbedarf [dBvKOS97, S.62]

- Jede Ecke wird höchstens einmal (in Q) gespeichert.
- Jede Kante (mit ihren Helper) wird höchstens einmal (in T) gespeichert.
- Anzahl der zu D hinzugefügten Diagonalen ist $\leq n - 2$.

Insgesamt also $O(n)$ Speicherbedarf.

3.6 Theorem [dBvKOS97, S.62]**.**
Ein einfach geschlossenes Polygon mit n-Ecken kann in y-monotone Polygone in $O(n \log n)$-Laufzeit mit $O(n)$-Speicher algorithmisch zerlegt werden. □

Triangulierung eines monotonen Polygons [dBvKOS97, S.63]

Ordne die Ecken lexikographisch nach der Höhe. Einrichten eines lifo-stack (Stapel) mit den Ecken die bereits behandelt wurden aber noch Diagonalen benötigen als Zusätzliche Datenstruktur. Der noch zu behandelnde Teil des Stapels schaut dabei wie eine Angel aus, d.h. eine Kante (die Angelrute) gefolgt mit einem konvexen Winkel von konkav verbundenen Kanten (der Angelschnur). Bei jeder Ecke v versuchen wir so viele Diagonalen wie möglich zu Punkten im Stapel zu machen, dabei gibt es zwei Fälle: Falls v gegenüber dem konkaven Teil liegt, also die untere Ecke der Angelrute ist, so verbinden wir diese mit allen Punkten auf der Angelschnur und entfernen letztere vom Stapel. Die unterste Ecke auf der Angelschnur und danach v kommen auf den Stack. Andernfalls verbinden wir v solange wir möglich mit Ecken die wir vom Stapel pop'en. Die letzte verbundene Ecke und danach v kommen auf den Stack.

Algorithmus TriangulateMonotonePolygon [dBvKOS97, S.64]

Input: Ein monotones Polygon als doppelt-verbundene Liste.
Output: Eine Triangulierung des Polygons als doppelt-verbundene Liste.

1: Sortiere die Ecken des Polygons lexikographisch: $v_1, v_2, \ldots, v_n$. // $O(n)$
2: Stapel $S := \emptyset$. Push'e die ersten beiden Ecken v_1 und v_2 auf S. // $O(1)$
3: **for** $j := 3$ to $n-1$ **do**
   // Gesamtzeit $O(n)$: Pro Schleife jeweils zwei push.
   // Insgesamt (mit Schritt 2) $2(n-3)+2$ push (and höchstens so viele pop).
4: **if** v_j und letzte auf den Stapel ge'push'te Ecke auf verschiedenen Seiten des Randes **then**
5: Pop'e alle Ecken von S und füge Diagonale von v_j zu jeder ge'pop'ten Ecke bis auf die letzte zu D hinzu.
6: Push'e v_{j-1} und v_j auf S.
7: **else**
8: Pop'e eine Ecke von S.
9: Pop'e Ecken von S so lange die Diagonalen von v_j zu ihnen im Polygon liegen. Füge diese Diagonalen zu D hinzu.
10: Push'e die letzte ge'pop'te Ecke und v_j auf S.
11: Füge Diagonalen zu allen Ecken am Stapel bis auf der ersten und letzten zu D hinzu. // $O(n)$

3.7 Theorem [dBvKOS97, S.65]**.**
Ein monotones Polygon mit n Ecken kann in $O(n)$-Zeit trianguliert werden. □

Tag 3.25

3.8 Folgerung [dBvKOS97, S.65]**.**
Ein einfach geschlossenes Polygon kann in $O(n \log n)$-Zeit und mit $O(n)$-Speicherplatz trianguliert werden.

Beweis. Zuerst zerlegen wir nach Theorem 3.6 das Polygon in monotone Teil in $O(n \log n)$-Zeit. Dann triangulieren wir jeden monotonen Teil in linear Zeit. Die Summe der Ecken aller Teile ist $O(n)$: Denn (nach der Eulerschen Formel $n-(n+(n-3))+(n-2)=1$) wurden höchstens $n-3$-Diagonalen zum Teilen (in höchstens $n-2$ Teile nach Theorem 3.1) verwendet, und jede Ecke gehört zu 1+Anzahl der dort endenden Diagonalen vielen Teilen, also ist die Summe all dieser Vielfachheiten $\leq n+2(n-3)$. Somit die Gesamtzeit dieses Schrittes ebenfalls $O(n)$. □

Verallgemeinerung [dBvKOS97, S.65]

Für Polygone mit Löchern gilt Theorem 3.8 genauso.

Beweis. Beachte, daß MakeMonoton und seine Subroutinen nirgends verwendet haben, daß der Rand zusammenhängend ist also auch für Polygone mit inneren Randkomponenten funktionieren. Da dabei die oberste/unterste Ecke jeder inneren Randkomponente ein split- bzw. ein merge-Punkt ist werden diese durch Diagonalen miteinander und schließlich mit dem äußeren Rand verbunden. Also haben die entstehenden Teilpolygone keine Löcher mehr und wir können auf diese jeweils TriangulateMonotonPolygon anwenden. Abschätzung der Anzahl der Diagonalen: Dazu triangulieren wir den polygonalen Bereich und auch jedes seiner Löcher. Nach

der Eulerschen Formel ist $1 = n - (n + d) + n_f$, wobei d die Anzahl der Diagonalen bezeichnet und bei jeder Ecke genau eine der (passend orientierten) ursprünglichen Seiten endet. Weiters ist $3n_f = 2(n + d) - n_0$, wobei n_0 die Anzahl der Kanten am äußeren Rand ist. Also erhalten wir $1 = -d + \frac{2(n+d)-n_0}{3}$ und somit $d = 2n - n_0 - 3$ und $n_f = \frac{2(n+2n-n_0-3)-n_0}{3} = 2n - n_0 - 2$ (Dies verallgemeinert Theorem 3.1). Analog zum Beweis von Theorem 3.8 ist die Anzahl der verwendeten Diagonalen ein $O(n)$ und somit die Gesamtanzahl der Ecken wieder ein $O(n)$. Damit gelten auch hier obige Abschätzungen für Laufzeit und Speicherbedarf. □

Noch allgemeiner können wir damit sogar Zerlegungen triangulieren (wobei wir diese in ein großes Rechteck einschreiben).

3.9 Theorem [dBvKOS97, S.66]**.**
Eine Zerlegung mit n Ecken der Ebene kann in $O(n \log n)$-Zeit und mit $O(n)$-Speicherbedarf algorithmisch erhalten werden.

Beweis. Aus der Zerlegung erhalten wir die (durch Polygone berandeten) Zusammenhangskomponenten auf die wir die vorige Verallgemeinerung anwenden können. Die Abschätzungen für Laufzeit und Speicherbedarf gelten wie zuvor, denn die Anzahl der Kanten einer Triangulierung ist wie eben $3n - n_0 - 3$ und somit die Gesamtanzahl der bei allen Zusammenhangskomponenten verwendeten Ecken $\leq n + 2(3n - n_0 - 3)$ also wieder ein $O(n)$. □

Bemerkungen und Kommentare [dBvKOS97, S.66]

Dieses Kunst-Gallerie-Problem wurde von Victor Klee 1973 gestellt und [Chv75] gab den ersten Beweis für $\lfloor n/3 \rfloor$ Kameras. Der viel einfachere Beweis hier ist von [Fis78] und basiert auf dem zwei-Ohren Theorem von [Mei75]. Der hier beschriebene Algorithmus für die Triangulierung monotoner Polygone stammt von [LP77]. In [TVW88] wurde ein $O(n \log \log n)$-Algorithmus für einfach geschlossene Polygone beschrieben und in [Cha90a] und [Cha91] wurde schließlich ein komplizierter $O(n)$-Algorithmus beschrieben. Daß die Fragestellung nach der Minimalanzahlvon Überwachungskameras für gegebene einfachgeschlossene Polygone NP-hart ist, stammt von [Agg84] und [LL86]. Das 3-dimensionale Problem einen Polyeder in Tetraeder zu zerlegen ist viel komplizierter. Man benötigt bisweilen zusätzliche Ecken nach [Cha84], einfachstes Beispiel ist das Schönhardt Polyeder. Und die Frage, ob diese bei einem allgemein gegebenen Polyeder wirklich nötig sind, ist NP-vollständig nach [RS92].

Tag 4.12

4. Lineares Programmieren (Gußformen)

Gießen [dBvKOS97, S.69]

Dabei geht es um das Problem, ob und wie ein gegossenes Objekt aus seiner Gußform entfernt werden kann ohne diese zu zerstören. **Voraussetzungen:** Objekte sind Polyeder. Gußformen bestehen nur aus einen Teil. Entfernen soll durch eine einzelne Translation geschehen. Oberste Fläche muß horizontal sein, also so viele mögliche Positionierungen wie Seiten des Polyeders.

Definition [dBvKOS97, S.70]

Wir nennen ein Polyeder gießbar, wenn es eine Positionierung gibt, sodaß er wie oben beschrieben aus seiner Gußform entfernt werden kann. Das Polyeder soll im Halbraum $z \leq 0$ positioniert werden. Die Gußform ist das Komplement des Objekts im Halbraum. Alle Seiten bis auf die in $z = 0$ liegenden heißen gewöhnliche Seiten.

Eigenschaften der Translationsrichtung [dBvKOS97, S.70]

Gesucht ist eine Richtung $d = (x, y, z)$ im Halbraum $z > 0$, sodaß das Polyeder in Richtung d beliebig verschoben werden kann ohne die Gußform zu treffen. Für jede gewöhnliche Seite f muß das Objekt von der entsprechenden Seite $\hat{f}$ der Gußform entfernt oder entlang dieser bewegt werden.

4.1 Lemma [dBvKOS97, S.71]**.**
Ein Polyeder kann genau dann von seiner Gußform in Richtung d entfernt (bzw. ein wenig in dieser bewegt werden), wenn der Winkel zwischen d und der nach außen weisenden Flächennormale auf jede gewöhnliche Seite ein stumpfer ist.

Beweis.
($\Rightarrow$) Wäre einer dieser Winkel spitz, so würde schon eine kleine Translation in Richtung d die entsprechende Seite in die Gußform verschieben.

($\Leftarrow$) Angenommen eine Translation um $t \cdot d$ mit $t > 0$ würde zu einer Kollision führen. Sei $p+t\,d$ mit $t > 0$ im Inneren der Form F und (o.B.d.A.) p im (Inneren des) Objekt(s) P. Sei q der zugehörige Eintrittspunkt am Rand der Form, d.h. $q = p + t_0\,d \in \partial P$, sodaß $p + t_0\,d \in F$ für alle $t > t_0$ nahe t_0. Da P 3-dimensional ist dürfen wir o.B.d.A. annehmen, daß q im Inneren einer Seite $\hat{f}$ von F liegt. Dann muß d mit der ins Innere von F weisenden Flächennormale an $\hat{f}$, also der nach außen weisenden Flächennormale an f, einen spitzen Winkel bilden. □

Folgerung [dBvKOS97, S.71]**.**
Wenn das Objekt mittels endlich vielen kleinen Translationen entfernt werden kann, dann auch mit einer einzigen.

Beweis. Fänden wir keine einzelne Translations, so wäre nach 4.1 für jedes d einer der Winkel spitz und somit könnten wir das Objekt nicht einmal ein wenig bewegen. □

Bestimmen einer möglichen Richtung [dBvKOS97, S.71]

Sei o.B.d.A. $d = (d_x, d_y, 1)$. Dann ist der Winkel zwischen d und einer Flächennormale $n = (n_x, n_y, n_z)$ genau dann stumpf, wenn $\langle d|n\rangle = d_x n_x + d_y n_y + n_z \leq 0$ ist. Dies beschreibt eine Halbebene (d.h. die Punkte die auf einer Seite einer Geraden liegen) in der Ebene $d_z = 1$. Ausnahme ist $n_x = 0 = n_y$, dann ist die Bedingung automatisch erfüllt ($n_z \leq 0$) oder unerfüllbar ($n_z > 0$). Die möglichen Richtungen d sind also genau jene im (konvexen) Durchschnitt dieser Halbebenen in $d_z = 1$.

Wir werden folgendes zeigen:

4.2 Theorem [dBvKOS97, S.72]**.**
Für Polyeder mit n Seiten können in erwarteter $O(n^2)$-Zeit und $O(n)$-Speicherbedarf entschieden werden, ob sie gießbar sind und falls ja auch eine geeignete Richtung berechnet werden.

Durchschnitte von Halbebenen [dBvKOS97, S.72]

Sei $H = \{h_1, \ldots, h_n\}$ eine endliche Menge linearer Ungleichungen der Form $a_i x + b_i y \leq c_i$ mit $(a_i, b_i) \neq 0$. Gesucht sind die Punkte (x, y) die gleichzeitig alle diese Bedingungen erfüllen. Dieser konvexe Bereich (so er nicht leer ist) wird von höchstens n Kanten berandet. Er kann auch unbeschränkt sein oder zu einen Punkt oder einer Kante degenerieren.

Mittels "Teilen und Erobern" beschreiben wir einen Algorithmus zur Bestimmung dieses Durchschnitts.

Algorithmus IntersectHalfPlanes [dBvKOS97, S.73]

Input: Eine endliche nicht-leere Liste H von n Halbebenen.
Output: Den konvexen polygonalen Bereich $C = \bigcap_{h \in H} h$.

1: **if** n=1 **then**
2: $C := h$ wobei $H = \{h\}$.
3: **else**
4: Zerlege H in Teile H_1 und H_2 mit $\lceil \frac{n}{2} \rceil$ und $\lfloor \frac{n}{2} \rfloor$ vielen Elementen.
5: $C_1 :=$IntersectHalfPlanes(H_1).
6: $C_2 :=$IntersectHalfPlanes(H_2).
7: $C :=$IntersectConvexRegions(C_1, C_2).

Algorithmus IntersectConvexRegions [dBvKOS97, S.73]

In Korollar 2.7 haben wir gezeigt, daß der Durchschnitt zweier Polygone in $O((n + k) \log n)$-Zeit bestimmt werden kann, wobei n die Gesamtanzahl der Ecken der beiden Polygone und k jene des Durchschnitts ist. Hier nun ist der Rand der konvexen Bereiche nicht notwendig ein Polygon, weshalb einfache Modifikationen notwendig sind.

[dBvKOS97, S.74]

Der Algorithmus MapOverlay aus Kapitel 2 berechnet nach 2.6 die gemeinsame Verfeinerung in $O((n+k) \log n)$-Zeit wobei k die Anzahl der Schnittpunkte von Kanten von C_1 mit solchen von C_2 ist. Da die Schnittpunkte Ecken von $C_1 \cap C_2$ sein müssen, dies aber höchstens n Kanten und somit auch höchsten n Ecken haben kann, ist $k \leq n$. Und somit die Gesamtlaufzeit rekursiv durch $T(1) = O(1)$ und $T(n) = O(n \log n) + 2T(\frac{n}{2})$ für $n \geq 1$ gegeben. Dies ergibt $T(n) = O(n \log(n)^2)$.

Verbesserung [dBvKOS97, S.74]

Wir wollen das verbessern, da wir ja zusätzlich wissen, daß C_1 und C_2 konvex sind. Dazu schließen wir die degenerierten Fälle, wo C_i nicht 2-dimensional ist, aus. Wir speichern den jeweils linken Left(C_i) und den rechten Rand Right(C_i) (d.h. die Kanten die den Bereich zur rechten bzw. linken haben) als von oben nach unten geordnete Halbebenen (O.B.d.A. seien keine horizontalen Randteile vorhanden). Die Ecken erhalten wir als Schnittpunkte aufeinanderfolgender Halbebenen.

[dBvKOS97, S.74]

Wir verwenden wieder einen plane sweep. Wegen Konvexität schneiden höchstens 4 Kanten die sweep line. Status T sind also nur 4 (möglicherweise nil-) Pointer $LeftC1$, $RightC1$, $LeftC2$ und $RightC2$. Initialisierung: Sei y_i die y-Koordinate der höchsten Ecke von C_i oder $+\infty$ falls diese nicht existiert. Sei $y_{\text{start}} := \min(y_1, y_2)$ der Anfangswert für die sweep line. Damit sind auch die Anfangswerte der 4 Pointer klar. Eine event queue ist unnötig, denn der nächste Eventpunkt ist der (lexikographisch) höchstgelegene der unteren Endpunkte der Segmente des Status. Bei jedem Eventpunkt startet eine neue Kante e eines Randes. Je nachdem welcher der 4 Pointer auf ihn zeigt rufen wir die entsprechende Subroutine auf. Z.B.:

Subroutine HandleLeftC1[dBvKOS97, S.75]

Sei der Eventpunkt p der obere Endpunkt von e und $C = C_1 \cap C_2$. Falls p zwischen LeftC2 und RightC2 liegt, dann ist e eine Kante von C und wir fügen die Halbebene die e beschreibt zu Left(C) hinzu. Falls e und RightC2 sich schneiden, dann ist der Schnittpunkt eine Ecke von C. Falls p rechts von RightC2 liegt dann sind beide vom Schnittpunkt nach unten ausgehenden Kantenteil Kanten von C und wir müssen diese Left(C) und Right(C) hinzufügen) und andernfalls sind bei nach oben gehenden Kantenteil Kanten von C (und wir müssen nichts machen). Falls schließlich e und LeftC2 sich schneiden, dann ist der Schnittpunkt wieder eine Ecke von C. Falls e links von LeftC2 liegt, dann ist e die beim Schnittpunkt startende Kante von C andernfalls ist es LeftC2 und wir fügen die jeweils entsprechende Halbebene zu Left(C) hinzu. Es kommen also möglicherweise die Halbebenen von e und von LeftC2 zu Left(C) und zwar jedenfalls jene von e zuerst.

Laufzeit [dBvKOS97, S.76]

Für jede neue Kante benötigen wir konstante Zeit und somit insgesamt $O(n)$.

Korrektheit [dBvKOS97, S.76]

Wir müssen zeigen, daß die Kanten in korrekter Reihenfolge zu C hinzugefügt werden. Sei p oberer Endpunkt einer Kante e von C. Dann ist p entweder ein oberer Endpunkt einer Kante von C_1 oder C_2, oder der Durchschnitt zweier Kanten e_i von C_i. Im ersten Fall wird e identifiziert, wenn wir p erreichen, und im zweiten Fall, wenn wir den niedereren oberen Endpunkt von e_1 oder e_2 erreichen. Somit werden alle C definierenden Halbebenen hinzugefügt. Das dies in der richtigen Reihenfolge geschieht ist offensichtlich.

4.3 Theorem [dBvKOS97, S.76]**.**
Der Durchschnitt zweier konvexer polygonaler (durch die geordneten linken und rechten Ränder gegebene) Bereiche der Ebene kann in $O(n)$-Zeit bestimmt werden.

Tag 4.14

[dBvKOS97, S.77]

Somit können wir den Kombinationsschritt in IntersectHalfPlanes in linearer Zeit durchführen und die Rekursionsvorschrift für die gesamte Laufzeit lautet $T(n) = O(n) + 2T(n/2)$, also ist $T(n) = O(n \log n)$.

4.4 Folgerung [dBvKOS97, S.77]**.**
Der Durchschnitt von n Halbebenen kann in $O(n \log n)$-Zeit bestimmt werden. □

Es gibt natürlich einen engen Zusammenhang zwischen der Bestimmung des Durchschnitts von Halbebenen und der Bestimmung der konvexen Hülle, siehe Abschnitt 8.2 zusammen mit 11.4.

Inkrementelles lineares Programmieren

[dBvKOS97, S.79]

Wir sind gar nicht an allen Lösungen interessiert (wofür wir $O(n \log n)$-Zeit brauchen) sondern nur an einer einzigen. Dies ist verwandt mit dem Problem der linearen Optimierung, d.h. der Suche nach jenem Punkt $x = (x_1, \ldots, x_d)$, der gegebene lineare Ungleichungen

$$h_i : \sum_{j=1}^{d} a_{i,j}\, x_j \leq b_i \text{ für } i \in \{1, \ldots, n\}$$

erfüllt und dabei ein lineares Funktional $f : x \mapsto \sum_{j=1}^{d} c_j\, x_j$ maximiert, also am weitesten in Richtung $c = (c_1, \ldots, c_d)$ liegt. Dabei nennt man jene Punkte, welche alle Ungleichungen h_i erfüllen, **mögliche Punkte**. Im Unterschied zu dem allgemeinen Problem ist bei uns die Anzahl ($d = 2$) der Variablen klein und die allgemeinen standard-Algorithmen (wie z.B. der Simplex-Algorithmus) nicht sehr effektiv.

[dBvKOS97, S.79]

Wir wählen eine beliebige Richtung $c \in \mathbb{R}^2 \setminus \{0\}$. Sei $C := \bigcap_i h_i$ die Menge der möglichen Punkte. Dann kann einer der folgenden Fälle eintreten:

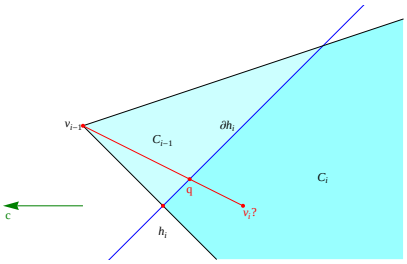
1. $C = \emptyset$.
2. C is unbeschränkt in Richtung c, d.h. es existiert eine Halbgerade $p + \mathbb{R}_+ \cdot c \subseteq C$. Als Lösung wollen wir diese angeben.
3. Es gibt eine Kante e von C, deren nach außenweisende Normale in Richtung c zeigt. Dann ist jeder Punkt auf der Kante Lösung.
4. Andernfalls existiert eine eindeutige Lösung, nämlich jene Ecke von C, für welche das Funktional $f = \langle c, \_ \rangle$ maximal wird.

[dBvKOS97, S.79]

Bei der Bestimmung von C wollen wir rekursiv vorgehen, indem wir schrittweise weitere Ungleichungen hinzufügen. Dabei wollen wir sicherstellen, daß die Lösung (sofern sie existiert) in jedem Einzelschritt eindeutig ist: Um Fall (2) zu vermeiden, fügen wir zwei weitere Ungleichungen hinzu, z.B. $m_i : \operatorname{sgn}(c_i)\, x_i \leq M$ für $i \in \{1, 2\}$ mit geeignet großem M. Um Eindeutigkeit im Fall (3) zu erreichen, nehmen wir die lexikographisch (bzgl. c) größte Ecke, d.h. wir sortieren nach der Größe der Projektion auf c und bei Gleichheit nach der auf $c^\perp$. Damit hat jedes lösbare Problem eine eindeutige lexikographisch größte Lösung und diese ist eine Ecke des Randes von C. Sei $H_i := \{m_1, m_2, h_1, \ldots, h_i\}$ und $C_i := \bigcap_{h \in H_i} h$. Sei v_i die eindeutige Lösung in C_i. Klarerweise ist $C_0 \supseteq C_1 \supseteq \ldots \supseteq C_n = C$. Falls also $C_i = \emptyset$ für ein i, so ist $C_j = \emptyset$ für alle $j \geq i$, und somit das Problem ohne Lösung.

4.5 Lemma. Induktionsschritt [dBvKOS97, S.80]**.**
Falls $v_{i-1} \in h_i$ dann ist $v_i := v_{i-1}$. Andernfalls ist entweder $C_i = \emptyset$ oder $v_i \in \partial h_i$, wobei ∂h_i die h_i begrenzende Gerade ist.

Beweis. Sei $v_{i-1} \in h_i$. Dann ist $v_{i-1} \in C_{i-1} \cap h_i = C_i$, also erst recht optimal für C_i, d.h. $c_i = c_{i-1}$.
Andernfalls ist $v_{i-1} \notin h_i$ und sei $C_i \neq \emptyset$ und $v_i \notin \partial h_i$. Sei e das Segment von v_{i-1} nach v_i. Wegen $v_i \in C_i \subseteq C_{i-1}$ und $v_{i-1} \in C_{i-1}$ ist $e \subseteq C_{i-1}$. Da v_{i-1} optimal ist, ist das Funktional f monoton fallend längs e. Da $v_{i-1} \notin h_i$ aber $v_i \in C_i \subseteq h_i$ existiert ein Schnittpunkt $q \neq v_i$ von $e \subseteq C_{i-1}$ mit $\partial h_i \subseteq h_i$. Also ist $q \in C_{i-1} \cap h_i = C_i$ und $f(v_{i-1}) \geq f(q) \geq f(v_i)$ und Gleichheit gilt nur, wenn v_{i-1} und damit q lexikographisch nach v_i kommt. Dies ist somit ein Widerspruch zur Maximalität von $v_i \in C_i$. □



1-dimensionale lineare Optimierung [dBvKOS97, S.81]

Wie finden wir v_i? Sei dazu $v_{i-1} \notin h_i$. Gesucht ist $p \in \partial h_i$ mit $p \in h$ für alle $h \in H_{i-1}$ und $f(p)$ maximal. Wir parametrisieren die Gerade ∂h_i mittels $t \in \mathbb{R}$ so, daß f in t wächst. Nun bestimmen wir die Parameter, welche den Schnittpunkten von ∂h_{i-1} mit ∂h_j entsprechen. Dann liegt $h_j \cap \partial h_i$ auf einer Seite des Schnittpunkts. Somit müssen wir nur den größten Parameter t_u finden, der einen Schnittpunkt, für welchen h_j oberhalb liegt, entspricht, und ebenso den kleinsten Parameter t_o, der einen Schnittpunkt, für welchen h_j unterhalb liegt, entspricht. Wegen m_1 und m_2 existieren t_u und t_o. Falls $t_u > t_o$ so ist die Aufgabe unlösbar. Andernfalls ist t_o der optimale Punkt v_i.

Algorithmus 2dBoundedLP [dBvKOS97, S.82]

Input: Ein lineares Optimierungsproblem $(H \cup \{m_1, m_2\}, f)$, wobei $h \in H$ Halbebenen, f ein lineares Funktional und $\{m_1, m_2\}$ die möglichen Punkte lexikographisch bzgl. f beschränkt.
Output: Entweder die Unlösbarkeit oder den eindeutig bestimmten lexikographisch bzgl. f maximalen Punkt.

```
1: Sei v_0 die (einzige) Ecke von C_0 = m_1 ∩ m_2 und H = {h_1, ..., h_n}.
2: for i := 1 to n do
3:   if v_{i-1} ∈ h_i then
4:     v_i := v_{i-1}
5:   else
6:     if ∂h_i ∩ C_{i-1} ≠ ∅ then
7:       Sei v_i ∈ ∂h_i ∩ C_{i-1} der lexikographisch bzgl. f maximale Punkt.
8:     else
9:       return "Problem unlösbar"
```

10: **return** v_n

4.7 Lemma. Laufzeit [dBvKOS97, S.82]**.**
Der Algorithmus 2dBoundedLP *berechnet die Lösung linearer Optimierungsaufgaben mit n Bedingungen und 2 Variablen in $O(n^2)$-Zeit und bei linearen Speicherbedarf.*

Beweis.
Korrektheit: Aus Lemma 4.5 folgt, daß v_i jeweils der optimale Punkt von C_i ist. Falls das 1-dimensionale Problem auf ∂h_i unlösbar ist, so ist $C_i = \emptyset$ und damit auch $C_n \subseteq C_i$.

Speicherbedarf ist klar, da wir in n Schritten jeweils eine weitere Halbebene hinzufügen.

Laufzeit: Das 1-dimensionale Problem im i-ten Schritt hat Laufzeit $O(i)$ und somit ist die Gesamtzeit $\sum_{i=1}^{n} O(i) = O(n^2)$. □

[dBvKOS97, S.83]

Die Laufzeit ist enttäuschend im Vergleich zu der mittels MapOverlay erreichten Laufzeit $O(n \log(n)^2)$. Die Abschätzung $O(i)$ für den i. Schritt ist recht grob, da nur im Fall $v_{i-1} \notin h_i$ notwendig. Allerdings kann passieren, daß die optimale Ecke sich wirklich in jedem Schritt ändert (z.B. falls alle $h_2, \ldots, h_n$ parallel liegen).

Zufallsgesteuertes lineares Optimieren [dBvKOS97, S.83]

Im letzten Beispiel liegt das Problem in der Anordnung der h_i. Es gibt immer eine "gute" Ordnung der h_i, aber diese ist schwer zu finden! Wir wollen nun zeigen, daß die Wahrscheinlichkeit, daß eine zufällig gewählte Ordnung "schlecht" ist, klein ist. Dazu ergänzen wir im Algorithmus 2dBoundedLP den Schritt (1) durch Anwenden einer zufälligen Permutation RandomPermutation$(h_1, \ldots, h_n)$ um den neuen Algorithmus 2dRandomizedBoundedLP zu erhalten.

Algorithmus RadomPermutation, [dBvKOS97, S.84]

Input: Ein Array $(h[1], \ldots, h[n])$.
Output: Eine Umordnung $(h[\sigma(1)], \ldots, h[\sigma(n)])$.
1: **for** $k = n$ downto 2 **do**
2: Vertausche $h[k]$ mit $h[\text{Random}(k)]$.

Wobei Random(k) eine ganze Zufallszahl in $\{1, \ldots, k\}$ liefern soll. Die Laufzeit dieses Algorithmus ist $O(n)$.

Laufzeit von 2dRandomizedBoundedLP [dBvKOS97, S.84]

Da die $n!$ vielen Permutationen gleich wahrscheinlich sind, ergibt sich die wahrscheinliche Laufzeit aus dem Mittel der Laufzeiten für jede Permutation.

4.8 Lemma [dBvKOS97, S.84]**.**
Das 2-dimensionale lineare Optimierungsproblem mit n-Bedingungen kann in erwarteter $O(n)$ Laufzeit und maximal $O(n)$ Speicherbedarf gelöst werden.

Beweis. Wir haben bereits gezeigt, daß der Speicherbedarf linear ist und die Laufzeit von RandomPermutation $O(n)$ ist. Hinzufügen einer Halbebene h_i benötigt konstante Zeit, falls sich die optimale Ecke nicht ändert.
Andernfalls müssen wir ein 1-dimensionales lineares Optimierungsproblem lösen. Wir wollen nun die Zeit für all diese 1-dimensionalen Probleme abschätzen. Sei dazu X_i eine Zufallsvariable, die für eine Anordnung der $h \in H$ gleich 1 ist, falls $v_{i-1} \notin h_i$ und 0 sonst. Das 1-dimensionale Problem mit i Bedingungen kann in $O(i)$ Zeit gelöst werden. Die Gesamtzeit für Zeile 5-9 ist somit $\sum_{i=1}^{n} O(i) \cdot X_i$. Da der Erwartungswert $E(X) = \sum_\sigma X(\sigma)\, P(\sigma) = \sum_i x_i \cdot P(X = x_i)$ (selbst für abhängige Zufallsvariablen) linear ist, gilt für den Erwartungswert der Laufzeit:

$$E\Big(\sum_i O(i) \cdot X_i\Big) = \sum_i O(i) \cdot E(X_i),$$

wobei $E(X_i) = 0 \cdot P(X_i = 0) + 1 \cdot P(X_i = 1)$ die Wahrscheinlichkeit $P(v_{i-1} \notin h_i)$ für $v_{i-1} \notin h_i$ ist.

Tag 4.15

Wir versuchen $P(X_i = 1)$ mittels rückwärts-Analyse zu bestimmen. Sei also v_n die optimale Ecke nach dem letzten Schritt. Dann wird die Ecke $v_n \in C_n$ durch 2 Halbebenen aus $\{m_1, m_2, h_1, \ldots, h_n\}$ beschrieben. Betrachten wir nun C_{n-1}. Dies unterscheidet sich von C_n durch Weglassen von h_n. Dabei ändert sich das Optimum genau dann, wenn v_n keine Ecke von C_{n-1} ist welche extremal in Richtung c ist. Dies ist genau dann der Fall, wenn h_n eine der v_n beschreibenden Halbebenen ist. Da die Halbebenen in zufälliger Reihenfolge sind, ist die Wahrscheinlichkeit dafür höchstens $2/n$, denn es könnte m_1 und/oder m_2 zur Festlegung von v_n notwendig gewesen sein. Analog gilt für den i. Schritt, daß $P(X_i = 1) \leq 2/i$. Also ist

$$\sum_i O(i) \cdot E(X_i) \leq \sum_i O(i) \cdot \frac{2}{i} = O(n).$$

Wie bereits gesehen benötigt der Rest des Algorithmus ebenfalls $O(n)$-Zeit. □

Unbeschränkte lineare Optimierungsprobleme [dBvKOS97, S.86]

Wir wollen nun die künstlich durch $\{m_1, m_2\}$ erzwungene Beschränktheitsbedingung loswerden. Dazu stellen wir zuerst fest, wann ein gegebenes Problem (H, c) unbeschränkt, d.h. die möglichen Punkte einen Halbstrahl $\rho = \{p + \lambda\delta : \lambda > 0\}$ enthalten, längs dem $f = \langle c, \_ \rangle$ unbeschränkt wächst (also $\langle \delta, c \rangle > 0$ ist). Damit $\rho \subseteq H$ gilt, muß $\langle \delta, n(h) \rangle \geq 0$ für alle $h \in H$ gelten, wobei $n(h)$ den nach innen zeigende Normalvektor auf den Rand von h bezeichnet.

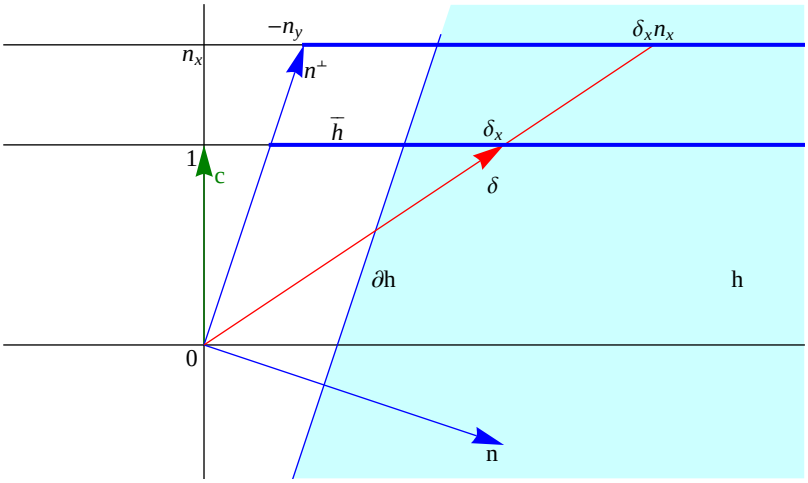
4.9 Lemma. Test für Unbeschränktheit [dBvKOS97, S.86]**.**
Ein lineares Optimierungsproblem (H, c) ist genau dann unbeschränkt, wenn ein Vektor δ existiert mit $\langle \delta, c \rangle > 0$ und $\langle \delta, n(h) \rangle \geq 0$ $\forall h \in H$ und weiters $\bigcap_{h \in H'} h \neq \emptyset$, wobei $H' := \{h \in H : \langle \delta, n(h) \rangle = 0\}$.

Beweis. ($\Rightarrow$) haben wir zuvor gezeigt, denn $H' \subseteq H$. ($\Leftarrow$) Sei δ wie im Lemma, $p_0 \in \bigcap_{h \in H'} h$ und $\rho_0 := \{p_0 + \lambda\delta : \lambda > 0\}$. Wegen $\forall h \in H' : \langle \delta, n(h) \rangle = 0$ ist

$\rho_0 \subseteq \bigcap_{h\in H'} h$ und wegen $\langle \delta, c\rangle > 0$ wächst f unbeschränkt längs ρ_0. Für $h \in H \setminus H'$ ist $\langle \delta, n(h)\rangle > 0$ und somit $\exists \lambda_h > 0$ mit $p_0 + \lambda\delta \in h\ \forall \lambda \geq \lambda_h$. Sei $\lambda' = \max_{h\in H\setminus H'} \lambda_h$ und $p := p_0 + \lambda'\delta$. Dann ist $\rho := \{p + \lambda\delta : \lambda > 0\}$ der gesuchte Strahl. □

Bestimmung von δ für 4.9, [dBvKOS97, S.87]

O.B.d.A. (nach Drehung) ist $c = (0, 1)$ und δ mit $\langle \delta, c\rangle > 0$ von der Form $\delta = (\delta_x, 1)$. Somit ist $\langle \delta, n(h)\rangle \geq 0$ mit $n(h) = (n_x, n_y)$ genau dann, wenn $\delta_x n_x \geq -n_y$. Die Existenz so eines δ ist somit das 1-dimensionales Problem $\delta_x \in \bigcap \bar{H}$, wobei $\bar{H} := \{\bar{h} : h \in H\}$ und $\bar{h} := \{t : t\, n_x(h) \geq -n_y(h)\}$.



Nach der Bestimmung von δ, [dBvKOS97, S.87]

Falls $\exists \delta_x \in \bigcap \bar{H}$, dann müssen wir wegen Lemma 4.9 nur noch die Lösbarkeit des Problems $H' := \{h \in H : \langle \delta, n(h)\rangle = 0\}$ bestimmen. Da $\forall h \in H' : \partial h \parallel \delta$, genügt es den Schnitt der $h \in H'$ mit der x-Achse (oder $\delta^\perp$) zu betrachten und das damit beschriebene 1-dimensionale Problem $\overline{H'}$ zu untersuchen. Falls $\bigcap \overline{H'} \neq \emptyset$, so ist das ursprüngliche Problem (H, c) unbeschränkt und wir erhalten einen entsprechenden Halbstrahl ρ in $O(n)$ Zeit (für die Berechnung der λ_h). Falls nicht, so ist $\bigcap H' = \emptyset$ und wir finden $h_1, h_2 \in H'$ mit $\emptyset = h_1 \cap h_2 \supseteq \bigcap H$. Andernfalls ist $\bigcap \bar{H} = \emptyset$ und somit (wegen dim = 1) insbesonders $\bar{h}_1 \cap \bar{h}_2 = \emptyset$ (für ein gewisses nach links beschränktes $\bar{h}_1$ und ein nach rechts beschränktes $\bar{h}_2$) und wegen 4.9 ist folglich $(\{h_1, h_2\}, c)$ und erst recht (H, c) beschränkt. Deshalb nennt man h_1, h_2 Zertifikate für die Beschränktheit von $(\{h_1, h_2\}, c)$ bzw. von (H, c).

Tag 4.19

Degeneriertheit [dBvKOS97, S.87]

Um $\{h_1, h_2\}$ anstelle von $\{m_1, m_2\}$ in 2dRandomizedBoundedLP zu verwenden müssen wir noch sicherstellen, daß $\{h_1, h_2\}$ sogar lexikographisch bzgl. c beschränkt ist. Falls $\bar{h}_1 = \emptyset$ schon für ein einzelnes $h_1 \in H$ ist (also ist $n(h_1) \parallel -c$), so suchen wir unter allen Halbebenen $h_2 \in H$, eine mit $\partial_1 \cap h_2$ in Richtung $c^\perp$ beschränkt (d.h. $\langle n(h_2), c^\perp\rangle < 0$). Falls solch eine existiert, so ist $\{h_1, h_2\}$ lexikographisch bzgl. c beschränkt. Andernfalls ist dann ein ganzer Halbstrahl $\rho \subseteq \partial h_1$ in Richtung $c^\perp$

in (H, c) bzgl. c maximal, also H lexikographisch unbeschränkt. Diesen Halbstrahl erhalten wir durch Lösen des 1-dimensionale Problems $\{h \cap \partial h_1 : h \in H\}$.

Subroutine 2dFindCertificate, [dBvKOS97, S.87]

Input: Ein lineares Optimierungsproblem (H, c).
Output: Halbstrahl falls lexikographisch unbeschränkt und Zertifikate andernfalls.

1: **if** $\exists \delta_\perp \in \bigcap_{h \in H} \bar{h} \neq \emptyset$ mit $\bar{h} := \{\delta_\perp : \langle \delta_\perp, n_\perp(h) \rangle \geq -\langle c, n(h) \rangle\} \subseteq c^\perp$ **then**
2: $H' := \{h : \langle n(h), \delta \rangle = 0\} \subseteq H$.
3: **if** $\exists p \in \bigcap_{h \in H'} h' \neq \emptyset$ mit $h' := h \cap \delta^\perp$ **then**
4: **return** H enthält unbeschränkten Halbstrahl $p + \mathbb{R}_+ \delta$.
5: **else**
6: **return** H ist unlösbar wegen $h_1, h_2 \in H$ mit $h_1 \cap h_2 = \emptyset$
7: **else**
8: **if** $\exists h_1 : \bar{h}_1 = \emptyset$, d.h. $n(h_1) \parallel c$ **then**
9: Untersuche $(\{h'' : h \in H\}, c^\perp)$, wobei $h'' := h \cap \partial h_1$.
10: **if** $\exists h_2 : h_2 \cap \partial h_1$ ist in Richtung $c^\perp$ beschränkt (oder sogar leer) **then**
11: **return** H ist lexikographisch beschränkt durch $h_1, h2$.
12: **else**
13: $\bigcap_{h \in H} h'' = \partial h_1 \cap \bigcap H$ ist lexikographisch unbeschränkten Halbstrahl.
14: **return** H ist lexikographisch unbeschränkt wegen dieses Halbstrahls.
15: **else**
16: **return** H ist lexikographisch beschränkt wegen h_1, h_2 mit $\bar{h}_1 \cap \bar{h}_2 = \emptyset$.

Tag 4.21

Laufzeit von 2dFindCertificate

Diese ist $O(n)$, denn:

(1) Lösen von $\delta \in \bigcap_{h \in H} \bar{h} \subseteq c^\perp$ in $O(n)$ Zeit.
(3) Lösen von $\bigcap_{n(h) \perp \delta} h \cap d^\perp \subseteq \delta^\perp$ in $O(n)$ Zeit.
(10) Lösen von $(\{h \cap \partial h_1 : h \in H\}, c^\perp)$ in $O(n)$ Zeit.

Zusammenfassung:

Algorithmus 2dRandomizedLP [dBvKOS97, S.88]

Input: Ein lineares Optimierungsproblem (H, c), wobei H eine endliche Liste von Halbebenen und $0 \neq c \in \mathbb{R}^2$ ist.
Output: Falls (H, c) lexikographisch unbeschränkt ist, wird ein entsprechender Halbstrahl ausgegeben. Falls H unlösbar ist (d.h. $\bigcap H = \emptyset$), werden 2 oder 3 Zertifikate $h \in H$ dafür ausgegeben. Andernfalls wird der lexikographisch bzgl. c größte Punkt in $\bigcap_{h \in H} h$ ausgegeben.

Algorithmus 2dRandomizedLP fortgesetzt

1: 2dFindCertificate(H, c)

2: Entweder liefert dies einen Halbstrahl, der H als lexikographisch unbeschränkt identifiziert, oder Zertifikate $h_1, h_2 \in H$ für die lexikographische Beschränktheit. Diese sind entweder schon widersprüchlich, oder es existiert eine lexikographisch maximale Lösung.
3: Sei $\{v_2\} = \partial h_1 \cap \partial h_2$ die maximale Lösung von $(\{h_1, h_2\}, c)$.
4: Sei $h_3, \dots, h_n$ eine zufällige Permutation der restlichen $h \in H$.
5: **for** $i = 3$ to n **do**
6: **if** $v_{i-1} \in h_i$ **then**
7: $v_i := v_{i-1}$
8: **else**
9: **if** $\partial h_i \cap C_{i-1} = \emptyset$, d.h. $C_i = \emptyset$ **then**
10: Finde $j, k < i$ mit $h_j \cap h_k \cap \partial h_i = \emptyset$.
11: **return** H ist unlösbar wegen h_i, h_j, h_k.
12: **else**
13: Sei $v_i \in \partial h_i \cap C_{i-1}$ der lexikographisch bzgl. c maximale Punkt.
14: **return** Maximale Lösung v_n.

4.10 Theorem [dBvKOS97, S.88]**.**
2-dimensionale lineare Optimierungsprobleme mit n Bedingungen können in erwarteter $O(n)$-Laufzeit mit maximal $O(n)$ Speicherbedarf gelöst werden.

Beweis. Die Schritte (3)–(14) von 2dRandomizedLP sind ident mit (1)–(10) von 2dRandomizedBoundedLP (siehe 2dBoundedLP) mit erwarteter Laufzeit $O(n)$ und maximal $O(n)$ Speicherbedarf. Jene von 2dFindCertificate sind maximal $O(n)$. □

Lineares Programmieren in höheren Dimensionen [dBvKOS97, S.88]

[dBvKOS97, S.89]

Wir verallgemeinern den Algorithmus für das 2-dimensionale lineare Optimieren nun auf höhere Dimensionen. Sei also H eine endliche Menge von Halbräumen $h \subseteq \mathbb{R}^d$ und $0 \neq c \in \mathbb{R}^d$. Wir suchen $p \in C := \bigcap_{h \in H} h$ mit $f(p) := \langle c, p \rangle$ maximal. Zwecks Eindeutigkeit im beschränkten Fall suchen wir den lexikographisch bzgl. c größten Punkt $p \in C$ (o.B.d.A. sei $c = (1, 0, \dots)$). Wie im 2-dimensionalen Fall gehen wir rekursiv vor, indem wir nach und nach Bedingungen $h \in H$ hinzufügen. Zwecks Eindeutigkeit stellen wir wie zuvor fest, ob das Problem lexikographisch unbeschränkt bzgl. c ist. Andernfalls erhalten wir (wie wir gleich zeigen werden) Zertifikate $h_1, \dots, h_k \in H$ mit $k \leq d$, welche die möglichen Punkte lexikographisch beschränken. Falls es überhaupt mögliche Punkte gibt, so muß $k = d$ sein. Seien $h_1, \dots, h_d$ Zertifikate, welche die Beschränktheit garantieren, und $h_{d+1}, \dots, h_n$ eine zufällige Anordnung der übrigen $h \in H$. Sei $C_i := \bigcap_{j=1}^{i} h_j$. Sofern existent sei v_i die optimale Ecke in C_i. Das folgende Lemma ist eine direkte Verallgemeinerung von Lemma 4.5:

4.11 Lemma [dBvKOS97, S.89]**.**
Für C_i und v_i wie gerade beschrieben gilt:

1. *Falls $v_{i-1} \in h_i$, dann ist $v_i = v_{i-1}$.*
2. *Falls $v_{i-1} \notin h_i$, dann ist $C_i = \emptyset$ oder $v_i \in \partial h_i$, der Hyperebene die h_i begrenzt.*

□

Behandlung von Fall 2 in 4.11, [dBvKOS97, S.89]

Im 2. Fall müssen wir also den optimalen Punkt in $\partial h_i \cap C_{i-1}$ finden. Dies ist ein lineares Optimierungsproblem in Dimension $\dim(\partial h_i) = d-1$. Für die lexikographische Ordnung auf ∂h_i verwenden wir die nicht verschwindenden Projektionen der Basisvektoren der ursprünglichen lexikographischen Ordnung.

Beschränktheitstest, [dBvKOS97, S.90] **I**

Wir müssen noch feststellen, ob das Problem unbeschränkt ist und andernfalls die gesuchten Zertifikate finden. Die Verallgemeinerung von Lemma 4.9 ist ohne Probleme, d.h. für die Beschränktheit eines d-dimensionales lineares Optimierungsproblem (H, c) müssen wir zuerst ein entsprechendes $(d-1)$-dimensionales Problem $\bar{H}$ beantworten. Falls $\bigcap \bar{H} \neq \emptyset$, so erhalten wir eine Richtung δ und das d-dimensionale lineare Problem ist entweder unbeschränkt in diese Richtung δ oder unlösbar. Dies kann festgestellt werden, wenn man $(H' := \{h \in H : \langle n(h), \delta\rangle = 0\}, c)$ auf Lösbarkeit überprüft. Es ist δ parallel zu den Rändern der Halbräume $h \in H'$, also muß dafür ebenfalls ein $(d-1)$-dimensionales Problem behandelt werden.

Tag 4.22

Beschränktheitstest, [dBvKOS97, S.90] **II**

Falls $\bar{H} = \emptyset$, dann erhalten wir $k \leq d$ Zertifikate $h_1, \ldots, h_k \in H$ mit $\bigcap_{i=1}^{k} \bar{h}_i = \emptyset$. Es ist $\bigcap_{i=1}^{k} h_i$ in Richtung c beschränkt, denn andernfalls würde der Richtungsvektor eines entsprechenden Halbstrahls ein Lösung δ für $\bar{H}$ liefern. Betrachtet man nun den echten affinen Teilraum, der von den bzgl. c maximalen Lösungen von $h_1 \cap \cdots \cap h_k$ erzeugt wird. Diese werden durch den Durchschnitt gewisser der Seitenflächen ∂h_i beschrieben. Wenn die Dimension dieses affinen Raums d' (Im Buch wird $d' = d-k$ fälschlich behauptet) ist, so auch jene des Durchschnitts der zugehörigen linearen Teilräume $n(h_i)^{\perp}$. Also ist die Dimension des linearen Erzeugnisses dieser $n(h_i)$ gerade $d-d'$, die Dimension des orthogonalen Komplements dieses Durchschnitts. Es reichen also $d-d'$ viele der h_i aus um den Durchschnitt zu beschreiben. Allerdings ist mir nicht klar, wie man diese in vernünftiger Laufzeit findet. Wir überprüfen nun, ob das Problem (H, c) auf diesem affinen Teilraum bzgl. lexikographischer Ordnung beschränkt ist. Falls nicht, dann geben wir einen entsprechenden Halbstrahl aus, andernfalls bilden die entsprechenden höchstens d' vielen Zertifikate zusammen mit den $d-d'$ vielen der h_i höchstens d viele Zertifikate für die lexikographische Beschränktheit in $\mathbb{R}^d$.

Subroutine FindCertificate, [dBvKOS97, S.87]

Input: Ein lineares Optimierungsproblem (H, c) der Dimension d.
Output: Einen entsprechenden Halbstrahl falls $\bigcap H$ lexikographisch unbeschränkt ist und höchstens d viele Zertifikate andernfalls.

1: **if** $\exists \delta_\perp \in \bigcap_{h\in H} \bar{h} \neq \emptyset$ mit $\bar{h} := \{\delta_\perp : \langle \delta_\perp, n_\perp(h)\rangle \geq -\langle c, n(h)\rangle\} \subseteq c^{\perp}$ **then**
2: $H' := \{h : \langle n(h), \delta\rangle = 0\} \subseteq H$.
3: **if** $\exists p \in \bigcap_{h\in H'} h' \neq \emptyset$ mit $h' := h \cap \delta^{\perp}$ **then**
4: **return** H enthält unbeschränkten Halbstrahl $p + \mathbb{R}_+ \delta$.
5: **else**
6: **return** H ist unlösbar wegen $h_1, h_2 \in H$ mit $h_1 \cap h_2 = \emptyset$

7: **else**
8: $\exists h_1, \ldots, h_k \in H : \bigcap_{i=1}^{k} \bar{h}_i = \emptyset$ (mit $k \leq (d-1)+1$).
9: Sei E der affine echte Teilraum der bzgl. c maximalen Punkte in $\bigcap_{i=1}^{k} h_i$.
10: Sei $H'' := \{h \cap E : h \in H\}$.
11: FindCertificate(H'')
12: **if** H'' ist lexikographisch beschränkt **then**
13: **return** H ist lexiko. beschränkt durch diese Zertifikate und $\{h_1, \ldots, h_k\}$.
14: **else**
15: **return** H ist lexiko. unbeschränkt wegen dem entsprechenden Halbstrahl in H''.

Zusammenfassung:

Algorithmus RandomizedLP [dBvKOS97, S.90]

Input: Ein lineares Optimierungsproblem (H, c), wobei H eine Menge von n Halbebenen im $\mathbb{R}^d$ und $0 \neq c \in \mathbb{R}^d$ ist.
Output: Falls (H, c) lexiko. unbeschränkt ist, einen entsprechender Halbstrahl. Falls es unlösbar ist, dann höchstens $d+1$ viele Zertifikate dafür. Andernfalls den lexikographisch bzgl. c größten Punkt in $\bigcap_{h \in H} h$.

[dBvKOS97, S.91]

1: FindCertificate(H, c)
2: Entweder liefert dies einen Halbstrahl, der H als lexikographisch unbeschränkt identifiziert, oder Zertifikate $h_1, \ldots, h_k \in H$ für die lexikographische Beschränktheit. Diese sind entweder schon widersprüchlich, oder besitzen eine lexikographisch maximale Lösung (dann ist $k = d$).
3: Seien $h_1, \ldots, h_d \in H$ Zertifikate für die lexikographische Beschränktheit und v_k die lexikographisch maximale Lösung von $h_1 \cap \cdots \cap h_d$.
4: Sei $h_{d+1}, \ldots, h_n$ eine zufällige Permutation der restlichen $h \in H$.
5: **for** $i = d+1$ to n **do**
6: **if** $v_{i-1} \in h_i$ **then**
7: $v_i := v_{i-1}$
8: **else**
9: v_i sei der lexikographisch bzgl. c maximale Punkt aus $\partial h_i \cap \bigcap_{j=1}^{i-1} h_j$.
10: **if** v_i existiert nicht **then**
11: Seien H^* Zertifikate für die Unlösbarkeit dieses Problems.
12: **return** Problem ist unlösbar wegen der Zertifikate $H^* \cup \{h_i\}$.
13: **return** v_n als eindeutige Lösung.

4.12 Theorem [dBvKOS97, S.91]**.**
Für fixe Dimension d können d-dimensionale lineare Optimierungsprobleme mit n Bedingungen in erwarteter $O(n)$ Laufzeit gelöst werden.

Beweis. Mittels Induktion nach d zeigen wir die Existenz einer Konstante K_d, s.d. die erwartete Laufzeit $\leq K_d n$ ist. Für $(d = 2)$ ist dies Theorem 4.10. Nun für (d). In FindCertificate müssen wir in Schritt (1) und Schritt (3) lineare Optimierungsprobleme der Dimension $d-1$ bzw. in Schritt (11) einen Aufruf von FindCertificate der Dimension $\leq d-1$ durchführen, also läßt sich die Laufzeit von Schritt (1) in

RandomizedLP mit $K_{d-1} \cdot n$ abschätzen. In Schritt (3) müssen wir ein lineares Gleichungssystem lösen, was angeblich $O(d)$ Zeit, meines Wissen nach aber $O(d^3)$ Zeit, benötigt. Ebenso benötigt Schritt (6) $O(d)$ Zeit. In Schritt (9) projezieren wir c auf ∂h_i in $O(d)$ Zeit und schneiden i Halbräume mit ∂h_i in $O(d\,i)$ Zeit. Weiters einen Aufruf von RandomizedLP für Dimension $d-1$ und $i-1$ Halbräume mit $K_{d-1}(i-1)$ Laufzeit. Sei wieder X_i die Zufallsvariable mit $X_i = 1$ falls $v_{i-1} \notin h_i$ und 0 andernfalls. Die erwartete Laufzeit des kompletten Algorithmus ist laut Buch somit beschränkt durch

$$O(d\,n) + d\,n\,K_{d-1} + \sum_{i=d+1}^{n} \Big(O(d\,i) + (i-1)K_{d-1} \Big) \cdot E(X_i).$$

Um $E(X_i)$ zu bestimmen verwenden wir wieder rückwärts-Analyse. Der optimale Punkte im i. Schritt ist eine Ecke $v_i \in C_i$ und somit durch d Halbräume beschrieben. Im Vergleich zum Schritt $(i-1)$ ändert sich v_i nur, wenn wir einen der v_i beschreibenden Halbräume entfernen. Die Wahrscheinlichkeit dafür ist $d/(i-d)$. Somit ist die erwartete Laufzeit beschränkt durch

$$O(d\,n) + d\,n\,K_{d-1} + \sum_{i=d+1}^{n} \Big(O(d\,i) + (i-1)K_{d-1} \Big) \cdot \frac{d}{i-d} \leq K_d\,n.$$

Im Buch wird daraus $K_d \leq O(K_{d-1}d)$ gefolgert (also $K_d = O(c^d\,d!)$ für ein $c > 0$ unabhängig von d) was wegen

$$\begin{aligned}
\sum_{i=d+1}^{n} \frac{i-1}{i-d} &= \sum_{j=1}^{n-d} \frac{d+j-1}{j} = n - d + (d-1) \sum_{j=1}^{n-d} \frac{1}{j} \\
&= n - 1 + (d-1) \sum_{j=2}^{n-d} \frac{1}{j} \geq n - 1 + (d-1) \int_2^{n-d+1} \frac{1}{x}\,dx \\
&= n - 1 + (d-1)(\log(n-d+1) - log(2)).
\end{aligned}$$

nicht stimmt, wie die z.B. die Wahl $d - 1 = n/2$ zeigt, denn dann wächst dieser Ausdruck wie $\frac{n}{2}\log(\frac{n}{2})$ und ist kein $O(n)$.

Kleinste umfassende Scheibe [dBvKOS97, S.92]

[dBvKOS97, S.92]

Betrachten wir einen Roboterarm, der Gegenstände an verschiedenen gegebenen Punkte aufnehmen und an anderen absetzen soll. Um mit einen möglichst kurzen Arm das Auslangen zu finden müssen wir die Kreisscheibe mit kleinsten Radius finden, die all die Punkte enthält. Für deren Existenz und Eindeutigkeit siehe Lemma 4.14. Wir verwenden wieder einen radomisierten inkrementellen Algorithmus. Dazu bilden wir zuerst ein Zufallspermutation $p_1, \ldots, p_n$ der Punkte $P := \{p_1, \ldots, p_n\}$. Sei $P_i := \{p_1, \ldots, p_i\}$ und D_i die kleinste Scheibe die P_i umfaßt. Das Analogon zu Lemma 4.5 ist folgendes:

4.13 Lemma [dBvKOS97, S.93]**.**
Sei $2 < i \leq n$ sowie P_i und D_i wie gerade definiert. Dann gilt:

1. *Falls $p_i \in D_{i-1}$, so ist $D_i = D_{i-1}$.*
2. *Falls $p_i \notin D_{i-1}$, so ist $p_i \in \partial D_i$.*

Dies wird aus Lemma 4.14 folgen.

Algorithmus MiniDisc[dBvKOS97,]

Input: Eine Menge P von n verschiedenen Punkten in $\mathbb{R}^2$.
Output: Die kleinste Scheibe D, welche P umfaßt.
1: Bestimme eine Zufallspermutation $p_1, \ldots, p_n$ von P.
2: Sei D_2 die kleinste Scheibe die $\{p_1, p_2\}$ umfaßt. // Diese hat Mittelpunkt in $\frac{p_1+p_2}{2}$ und Radius $\frac{1}{2}|p_2 - p_1|$.
3: **for** $i = 3$ to n **do**
4: **if** $p_i \in D_{i-1}$ **then**
5: $D_i := D_{i-1}$
6: **else**
7: $D_i :=$MiniDiscWithPoint(P_{i-1}, p_i)
8: **return** D_n

Für die Subroutine MiniDiscWithPoint gehen wir ganz analog vor:

Subroutine MiniDiscWithPoint[dBvKOS97, S.93]

Input: Eine Menge P von n verschiedenen Punkten in $\mathbb{R}^2$ und ein Punkt q, für welche eine Scheibe $D \supseteq P$ existiert mit $q \in \partial D$.
Output: Die kleinste Scheibe $D \supseteq P$ mit $q \in \partial D$.
1: Bestimme eine Zufallspermutation $p_1, \ldots, p_n$ von P.
2: Sei D_1 die kleinste Scheibe die $\{p_1, q\}$ umfaßt.
3: **for** $i = 2$ to n **do**
4: **if** $p_i \in D_{i-1}$ **then**
5: $D_i := D_{i-1}$
6: **else**
7: $D_i :=$MiniDiscWith2Points(P_{i-1}, p_i, q)
8: **return** D_n

Auch für die Subroutine MiniDiscWith2Points gehen wir ganz analog vor:

Subroutine MiniDiscWith2Points[dBvKOS97, S.94]

Input: Eine Menge P von n verschiedenen Punkten in $\mathbb{R}^2$ und zwei weitere Punkte q_1 und q_2, für welche eine Scheibe $D \supseteq P$ existiert mit $\{q_1, q_2\} \subseteq \partial D$.
Output: Die kleinste Scheibe $D \supseteq P$ mit $\{q_1, q_2\} \in \partial D$.
1: Sei D_0 die kleinste Scheibe, die $\{q_1, q_2\}$ umfaßt.
2: **for** $i = 1$ to n **do**
3: **if** $p_i \in D_{i-1}$ **then**
4: $D_i := D_{i-1}$
5: **else**
6: D_i ist die Scheibe deren Rand der Kreis durch q_1, q_2 und p_i ist.
7: **return** D_n

4.14 Lemma [dBvKOS97, S.94]**.**
Seien P und R zwei disjunkte endliche Mengen von Punkten mit $P \cup R \neq \emptyset$. Dann gilt:

1. *Falls eine Scheibe $D \supseteq P$ mit $R \subseteq \partial D$ existiert. Dann existiert auch die kleinste solche (bezeichnet als $\mathrm{md}(P, R)$) und ist eindeutig.*

Sei $p \in P$ und P, R erfülle die Voraussetzungen von (1) sowie $R \cup (P \setminus \{p\}) \neq \emptyset$. Dann gilt weiters:

2. $p \in \mathrm{md}(P \setminus \{p\}, R) \Rightarrow \mathrm{md}(P, R) = \mathrm{md}(P \setminus \{p\}, R)$.
3. $p \notin \mathrm{md}(P \setminus \{p\}, R) \Rightarrow \mathrm{md}(P, R) = \mathrm{md}(P \setminus \{p\}, R \cup \{p\})$.

Beweis. (1) Wir bezeichnen mit $D_r(p)$ die Scheibe mit Mittelpunkt p und Radius $r \geq 0$. Sei $p \in P \cup R$ fix gewählt. Sei r_∞ das Infimum der Radien aller Scheiben $D \supseteq P$ mit $R \subseteq \partial D$. Dieses Infimum ist ein Minimum, denn wenn $D_{r_n}(p_n)$ solche Scheiben mit Radien $r_n \searrow r_\infty$ sind, dann besitzen die Mittelpunkte p_n eine Häufungspunkt p_∞ da $|p - p_n| \leq r_n \leq r_0$. Es ist $D_\infty := D_{r_\infty}(p_\infty)$ eine Scheibe mit $P \subseteq D_\infty$, denn für jeden Punkt $p' \in P$ gilt $|p' - p_n| \leq r_n$ und somit durch Grenzübergang $|p' - p_\infty| \leq r_\infty$. Und ganz analog folgt $R \subseteq \partial D_\infty$. Angenommen es gäbe 2 solche Schreiben $D_0 := D(M_0, r_\infty)$ und $D_1 := D(M_1, r_\infty)$ mit $M_0 \neq M_1$. Da $p \in P \subseteq D_0 \cap D_1$ ist, ist $\emptyset \neq \partial D_0 \cap \partial D_1 =: \{z, \bar{z}\}$ und wir betrachten für $\lambda \in [0, 1]$ die Mittelpunkte $M_\lambda := M_0 + \lambda(M_1 - M_0)$, Radien $r_\lambda := |z - M_\lambda|$ und Scheiben $D_\lambda := D(M_\lambda, r_\lambda)$. Dann ist $P \subseteq D_0 \cap D_1 \subseteq D_\lambda$ für alle $\lambda \in [0, 1]$. Wegen $R \subseteq \partial D_0 \cap \partial D_1 = \{z, \bar{z}\}$ ist $R \subseteq \partial D_\lambda$, aber $r_\lambda < r_\infty$ für $0 < \lambda < 1$, ein Widerspruch.

Tag 4.26

Beachte, daß damit auch $\mathrm{md}(P', R')$ für alle $P' \subseteq P$ und $R' \subseteq R$ mit $P' \cup R' \neq \emptyset$ existiert. (2) Sei $p \in D := \mathrm{md}(P \setminus \{p\}, R)$. Dann ist $P = (P \setminus \{p\}) \cup \{p\} \subseteq D$ und $R \subseteq \partial D$. Und D ist minimal mit diesen beiden Eigenschaften, den eine kleinere solche Scheibe würde auch $P \setminus \{p\}$ enthalten, im Widerspruch zur Definition von D, also ist $D = \mathrm{md}(P, R)$. (3) Sei $p \notin D_0 := \mathrm{md}(P \setminus \{p\}, R)$. Sei $D_1 := \mathrm{md}(P, R)$. Wir betrachten D_λ wie in (1). Nach Voraussetzung ist $p \in D_1 \setminus D_0$. Somit existiert ein $0 < \lambda_0 \leq 1$ mit $p \in \partial D_{\lambda_0}$. Wie in (1) ist $P \setminus \{p_0\} \subseteq D(\lambda_0)$ und $R \subseteq \partial D_{\lambda_0}$. Wegen der Minimalität von D_1 ist $D_{\lambda_0} = D_1$, also $p \in \partial D_1$, d.h. $\mathrm{md}(P, R) = \mathrm{md}(P, R \cup \{p\}) \overset{(2)}{=} \mathrm{md}(P \setminus \{p\}, R \cup \{p\})$. □

Beweis von Lemma 4.13

Beachte, daß D_i aus Lemma 4.13 gerade $\mathrm{md}(\{p_1, \ldots, p_i\}, \emptyset)$ ist, und somit dieses folgt. □

4.15 Theorem [dBvKOS97, S.94]**.**
Die kleinste eine gegebene n-elementige Menge von Punkten in der Ebene umfassende Scheibe kann in erwarteter $O(n)$ Laufzeit und mit maximal $O(n)$ Speicherbedarf berechnet werden.

Beweis. Aus Lemma 4.14 folgt, daß MiniDiscWith2Points, MiniDiscWithPoint und schließlich MiniDisc die gestellten Aufgaben korrekt lösen. Nun zur Laufzeit und zum Speicherbedarf:

- MiniDiscWith2Points benötigt $O(n)$ Zeit (Konstante Zeit für jede Schleife) und $O(n)$ Speicher.
- MiniDiscWithPoint und MiniDisc haben $O(n)$ Speicher-Bedarf.
- MiniDiscWithPoint hat abgesehen von the Aufrufen von MiniDiscWith2Points $O(n)$ Laufzeit.
- Zur Bestimmung der Wahrscheinlichkeit von Aufrufen von MiniDiscWith2Points verwenden wir wieder Rückwärts-Analyse: Sei D_i die kleinste Scheibe mit $p \in \partial D_i$, welche $\{p_1, \ldots, p_i\}$ enthält. Bei Entfernen eines der Punkte ändert sich die kleinste Scheibe höchstens dann, wenn dies ein Randpunkt ist, und zwar nur dann, wenn außer p nur zwei Randpunkte vorliegen. Das heißt, die Wahrscheinlichkeit dafür ist wieder $\leq 2/i$ und somit die erwartete Laufzeit von MiniDiscWithPoint beschränkt durch

$$O(n) + \sum_{i=2}^{n} O(i)\frac{2}{i} = O(n).$$

- Das selbe Argument liefert für die erwartete Laufzeit von MiniDisc ebenfalls $O(n)$.

□

[dBvKOS97, S.96]

Der Algorithmus MiniDisc kann noch vereinfacht werden, indem man nur einmal ein zufällige Permutation durchführt und diese dann auch in der Subroutine MiniDiscWithPoint verwendet. Weiters kann man anstelle der 3 Routinen auch einen einzigen Algorithmus MiniDiscWithPoints(P, R) schreiben, welcher, das in Lemma 4.14 definierte $\mathrm{md}(P, R)$, bestimmt.

Nachbemerkungen

[dBvKOS97, S.96]

Den Durchschnitt von d-dimensionalen Halbräumen für große d zu bestimmen ist viel aufwendiger, da die Anzahl der Seiten des Durchschnitts ein $O(n^{\lfloor d/2 \rfloor})$ ist, siehe [Ede87a]. Für lineare Optimierungsprobleme wurde der Simplex-Algorithmus (mit maximal exponentieller Laufzeit) entwickelt, siehe [Dan63], und es gibt auch Algorithmen mit polynomialer Laufzeit von [Kha80] und von [Kar84]. In [Meg84] wurde erstmals gezeigt, daß die Überprüfung, ob ein Durchschnitt von Halbräumen leer ist, echt einfacher als die Bestimmung des Durchschnitts ist. Sein (deterministischer) Algorithmus hat Laufzeit $O(K_d\, n)$ mit $K_d = 2^{2^d}$. Dies wurde in [CJY95] und [Dye86] auf $K_d = 3^{d^2}$ verbessert. Die hier gegebenen randomizierten inkrementellen Algorithmen stammen von [Sei91b], wo $K_d = d!$ gezeigt wird. Der Algorithmus für kleinste Scheiben stammt von [Wel91], der das auch auf höher-dimensionale Sphären übertragen hat. Es ist klarerweise umstritten, ob randomizierte Algorithmen in Zeit-kritischen Situationen wirklich eingesetzt werden sollen (z.B. Flugüberwachung, intensiv Station im Krankenhaus, Kernkraftwerke, etc.)

5. Orthogonale Bereichs-Suche (Datenbankabfrage)

[dBvKOS97, S.102]

Wir wollen nun gewisse Datenbankabfragen geometrisch interpretieren. Die Records der Datenbank sollten dazu Punkten in einem $\mathbb{R}^d$ entsprechen. Und die Abfrage damit Abfragen über solche Punktmengen sein, also z.B. die Frage nach allen Angestellten mit Geburtsdatum in einem gewissen Intervall und Einkommen in einem zweiten Intervall. Wir wollen also jene Punkte der Datenbank bestimmen, die in einem achsenparallelen Rechteck (oder höherdimensionalen Quader) liegen. Wir nennen das eine orthogonale Bereichsabfrage.

1-dimensionale Bereichsabfrage, [dBvKOS97, S.102]

Wir suchen zuerst jene Punkte(=Zahlen) aus $P := \{p_1, \ldots, p_n\}$ die in einem Intervall $[x, x']$ liegen. Dies können wir effektiv lösen indem wir eine balanzierten Such-Baum T verwenden, wobei dessen Blätter die Zahlen p_i enthalten und dessen Knoten v Werte x_v speichern, welche die Suche dirigieren, d.h. der linke Teilbaum alle Werte $p_i \leq x_v$ und der rechte alle übrigen enthält. Allerdings verallgemeinert sich das nicht auf höhere Dimensionen und erlaubt auch nicht effektive Updates für P.

Aufspaltungspunkt v_{split}, [dBvKOS97, S.102]

Um nun die Punkte im Intervall $[x, x']$ zu erhalten, suchen wir nach x und x' in T. Seien μ und μ' jene Blätter, wo die entsprechende Suche endet. Die gesuchten Punkte finden sich dann in den Blättern zwischen μ und μ' eventuell zusammen mit jenen in μ und/oder μ'. Erstere liegen auf Teilbäumen deren Wurzel zwischen den beiden Suchpfaden liegt. Um diese zu bestimmen, suchen wir den letzten gemeinsamen Knoten v_{split} der beiden Suchpfade. Sei $\mathrm{l}(v)$ und $\mathrm{r}(v)$ die beiden Kinder des Knotens v.

Algorithmus FindSplitNode[dBvKOS97, S.103]

Input: Ein binärer Baum T und zwei Werte $x \leq x'$.
Output: Den letzten gemeinsamen Knoten der Suchpfade zu x und x'.

1: Sei v die Wurzel von T.
2: **while** v ist kein Blatt und nicht $x \leq x_v < x'$ **do**
   // Die Suchpfade trennen sich hier nicht
3: **if** $x' \leq x_v$ **then** // $x \leq x' \leq x_v$
4: $v := \mathrm{l}(v)$
5: **else** // $x' > x > x_v$
6: $v := \mathrm{r}(v)$
7: **return** v

Tag 4.28

Ausgabe, [dBvKOS97, S.103]

Startend bei v_{split} folgen wir den Suchpfad zu x und geben in jenen Knoten, wo der Pfad nach links geht, alle Blätter des rechten Teilbaums aus. Analog folgen wir den Suchpfad zu x' und geben in jenen Knoten, wo der Pfad nach rechts geht, alle Blätter des linken Teilbaums aus. Schließlich überprüfen wir jedes der beiden Endblätter, ob es dazugehört. Im Detail benötigen wir dazu eine Subroutine ReportSubtree, die einen (Teil-)Baum durchläuft und alle seine Blätter ausgibt. Deren Laufzeit ist linear in der Größe der Ausgabe, da bei binären balanzieren Bäumen die Anzahl der inneren Knoten kleiner als jene der Blätter ist.

Algorithmus 1dRangeQuery [dBvKOS97, S.103]

Input: Ein binärer Baum T und zwei Werte $x \leq x'$.
Output: Alle Blätter in T, welche im Bereich $[x, x']$ liegen.

```
1:  v_split :=FindSplitNode(T, x, x').
2:  if v_split ist Blatt then
3:    print v_split falls v_split ∈ [x, x'].
4:  else
      // Folge dem Suchpfad zu x und berichte Punkte in rechten Teilbäumen
5:    v := l(v_split)
6:    while v ist nicht Blatt do
7:      if x ≤ x_v then
8:        ReportSubtree(r(v))
9:        v := l(v)
10:     else
11:       v := r(v)
12:   print v falls v ∈ [x, x'].
13:   Verfahre analog mit dem Suchpfad von v_split nach x'
14: return
```

5.1 Lemma.[dBvKOS97, S.104]**.**
Der Algorithmus 1dRangeQuery *liefert genau die Punkte im angegebenen Bereich.*

Beweis. Wir zeigen zuerst, daß jeder gelieferte Punkt im Abfragebreich liegt. Falls p das Endblatt des Suchpfads zu x oder x' ist, so wird dieses explizit geprüft. Andernfalls wird p bei einem Aufruf von ReportSubtree ausgegeben. Sei v im Suchpfad von x (analog für x') und somit betrachten wir ReportSubtree(r(v)). Damit liegt v und r(v) im linken Teilbaum von v_{split}, also ist $x \leq x_v < p \leq x_{v_{\text{split}}} < x'$. Sei nun umgekehrt $p \in P \cap [x, x']$ und μ das Blatt mit Wert p. Sei v der tiefste Vorfahre von p, welcher im Algorithmus 1dRangeQuery ausgegeben wurde. Angenommen $v \neq \mu$. Es kann v kein besuchter Knoten in einem Aufruf von ReportSubtree sein, den dann wären alle seine Nachfahren (also auch μ) bei diesem Aufruf ausgegeben worden. Somit liegt v auf dem Suchpfad zu x oder zu x'. Betrachten wir o.B.d.A. den ersten Fall. Da der Pfad zu x und p bei v aufspaltet (da v tiefstgelegen ist), liegen p und x auf verschiedenen Teilbäumen von v und falls p nicht berichtet wurde, muß der Pfad zu x bei v nach rechts gehen, also $p \leq x_v < x$ gelten, ein Widerspruch. □

Speicherbedarf [dBvKOS97, S.104]

Für das Einrichten des balanzierten Suchbaums benötigen wir $O(n)$ Speicher und $O(n \log(n))$ Laufzeit. Die Abfragezeit ist im schlimmsten Fall n, falls alle Punkte im

Suchbereich liegen, deshalb versuchen wir die Laufzeit in Abhängigkeit von der Größe k der Ausgabe zu bestimmen: ReportSubtree ist linear in seiner Ausgabe. Also die Gesamtzeit all dieser Aufrufe $O(k)$. Die übrigen besuchten Knoten sind jene auf den $O(\log(n))$ langen Suchpfaden zu x und x'. Bei jedem solchen Knoten benötigen wir $O(1)$ Zeit. Und somit ist die gesamte Abfragezeit $O(k + \log n)$

5.2 Theorem [dBvKOS97, S.105]**.**
Sei P eine n-elementige Teilmenge von $\mathbb{R}$. Diese kann in einem balanzierten Suchbaum in $O(n \log(n))$ Zeit gespeichert werden und benötigt dazu $O(n)$ Speicher. Die Abfragezeit ist dann $O(k+\log n)$, wobei k die Anzahl der ausgegebenen Punkte ist. □

Kd-Bäume [dBvKOS97, S.105]

2-dimensionaler orthogonale Bereichssuche [dBvKOS97, S.105]

Sei P eine n-elementige Teilmenge von $\mathbb{R}^2$. Wir setzen vorerst voraus, daß keine zwei Punkte die gleiche x- bzw. die gleiche y-Koordinate haben. Eine 2-dimensionale orthogonale Bereichssuche bedeutet jene Punkte in $p \in P$ zu finden, welche in einem Rechteck der Form $[x, x'] \times [y, y']$ liegen. Das bedeutet also zwei 1-dimensionalen Bereichsabfragen. Im 1-dimensionalen Fall wird der binären Suchbaum rekursiv beschrieben, indem man die Punkte in zwei annähernd gleich große Teile teilt, wobei ein Teil aus den Punkten mit Wert kleiner-gleich dem Aufspaltungs-Wert und der andere aus jenen mit Wert größer als der Aufspaltungswert besteht. Im 2-dimensionalen spalten wir zuerst nach x-Koordinate, dann die beiden entstandenen Teile nach y-Koordinate, als nächstes deren Teile wieder nach der x-Koordinate u.s.w.. Wir teilen also die Menge rekursiv durch abwechselnd vertikale und horizontale Geraden. Der entstehende Baum heißt kd-Baum (steht für k-dimensional, also eigentlich $2d$-Baum)

Algorithmus BuildKdTree [dBvKOS97, S.106]

Input: Eine endliche Menge $\emptyset \neq P \subseteq \mathbb{R}^2$ und eine Tiefe t.
Output: Die Wurzel eines kd-Baums für P.

1: **if** $P = \{p\}$ **then**
2: **return** Ein neues Blatt mit Inhalt p
3: **else**
4: **if** t ist gerade **then**
5: Zerlege P mittels vertikaler Gerade durch die mittlere x-Koordinate m der Punkte aus P in $P_1 := \{p \in P : p_x \leq m\}$ und $P_2 := \{p \in P : p_x > m\}$.
6: **else**
7: Zerlege P mittels horizontaler Gerade durch die mittlere y-Koordinate m der Punkte aus P in $P_1 := \{p \in P : p_y \leq m\}$ und $P_2 := \{p \in P : p_y > m\}$.
8: Erzeuge neuen Knoten mit Wert m und Kindern
9: $v_l :=$BuildKdTree$(P_1, t + 1)$
10: $v_r :=$BuildKdTree$(P_2, t + 1)$
11: **return** v

Dabei bedeutet mittlere Koordinate m die $\lfloor \frac{n}{2} \rfloor$ kleinste Zahl unter den gegebenen. Bei 2 Werten bedeutet dies den kleineren der beiden und gewährleistet das der Algorithmus endet.

Laufzeit [dBvKOS97, S.107]

Am zeitaufwendigsten ist in jedem Rekursionsschritt das Finden von m. Dies ist (kompliziert) in linearer Zeit möglich. Besser ist die Punkte sowohl nach x als auch unabhängig nach y zu sortieren. Danach findet man m einfach in linearer Zeit. Die sortierten Listen für P_1 und P_2 findet man ebenso einfach in linearer Zeit. Also ist die Laufzeit T für das Erzeugen des Baums rekursiv gegeben durch

$$T(n) = \begin{cases} O(1) & \text{für } n = 1, \\ O(n) + 2T(\lceil \frac{n}{2} \rceil) & \text{sonst,} \end{cases}$$

also $T(n) = O(n \log(n))$ (auch inklusive dem Sortieren).

Speicherbedarf [dBvKOS97, S.107]

Jedes Blatt speichert einen Punkt aus P, also haben wir n Blätter. Der kd-Baum ist ein binärer Baum mit $O(1)$ Speicherbedarf für jeden internen Knoten, also mit Gesamtspeicherbedarf $O(n)$.

5.3 Lemma [dBvKOS97, S.107]**.**
Für den Aufbau eines kd-Baum für n Punkte benötigt man $O(n)$ Speicherplatz und $O(n \log n)$ Zeit. □

Rechteckige Region eines Knotens [dBvKOS97, S.108]

Jedem Knoten v entspricht ein möglicherweise auf manchen Seiten unbeschränktes Rechteck region(v), wobei die Randlinien durch die Werte der Vorgänger von v beschrieben werden. Für die Wurzel v des Baumes ist region(v) $= \mathbb{R}^2$. Ein Punkt kommt genau dann im Teilbaum mit Wurzel v vor, wenn er zu region(v) gehört. Somit wüssen wir einen Teilbaum nur dann durchsuchen, wenn region(v) den Suchbereich trifft.

Abfrage-Algorithmus [dBvKOS97, S.108]

Wir durchlaufen den kd-Baum, aber besuchen dabei nur Knoten v deren region(v) den Suchbereich trifft. Falls diese ganz im Suchbereich enthalten sind, so geben wir alle Blätter dieses Teilbaums aus. Wenn wir beim Durchlaufen auf ein Blatt stoßen, müssen wir überprüfen, ob der entsprechende Punkt im Suchbereich enthalten ist, und falls ja diesen ausgeben.

Unter Benutzung der Subroutine ReportSubtree erhalten wir:

Algorithmus SearchKdTree [dBvKOS97, S.109]

Input: Die Wurzel v eines kd-Baums und einen Suchbereich R.
Output: Die Werte aller Blätter des Baums, die im Bereich R liegen.

```
1: if v ist ein Blatt then
2:    Teste, ob der Wert von v in R liegt, und falls ja gebe diesen aus.
3: else // v ist innerer Knoten
4:    if region(l(v)) ⊆ R then
5:       ReportSubtree(l(v))
6:    else // region(l(v)) ∩ (ℝ² \ R) ≠ ∅
```

```
7:      if region(l(v)) ∩ R ≠ ∅ then
8:        SearchKdTree(l(v), R)
9:    if region(r(v)) ⊆ R then
10:     ReportSubtree(r(v))
11:   else // region(r(v)) ∩ (ℝ² \ R) ≠ ∅
12:     if region(r(v)) ∩ R ≠ ∅ then
13:       SearchKdTree(r(v), R)
```

[dBvKOS97, S.109]

Für die Tests region$(v) \cap R \neq \emptyset$ könnten wir vorweg region(v) für alle v bestimmen und speichern. Besser ist es die aktuelle Region zu speichern und mittels der, in den internen Konten gespeicherten Begrenzungslinien, bei den rekursiven Aufrufen zu adaptieren, z.B. ist für Knoten v von gerader Tiefe

$$\text{region}(\text{l}(v)) = \text{region}(v) \cap \{p : p_x \leq m_v\}.$$

Beachte, daß R bei SearchKdTree nicht ein Rechteck zu sein braucht.

Tag 4.29

5.4 Lemma. Laufzeit[dBvKOS97, S.110]**.**
Eine Abfrage für ein achsenparalleles Rechteck in einem kd-Baum mit n Punkten kann in $O(k+\sqrt{n})$ Laufzeit durchgeführt werden, wobei k die Anzahl der ausgegebenen Punkte ist.

Beweis. Die Zeit um einen Teilbaum zu durchlaufen und die Punkte in den Blättern auszugeben ist linear in den ausgegebenen Punkten. Also ist die Gesamt-Zeit $O(k)$ für die Schritte (5) und (10). Bleibt die Zeit für jene besuchten Knoten v zu bestimmen, die nicht in einem der durchlaufenen Teilbäume liegen, d.h. für welche region(v) den Abfragebereich trifft aber nicht in diesem enthalten ist, also der Rand des orthogonalen Abfragebereichs R region(v) trifft, d.h. die Bedingungen (7) bzw. (12) erfüllen. Dazu bestimmen wir die Anzahl $Q(n)$ der v, für die region(v) eine (der R begrenzenden) vertikalen/horizontalenGeraden ℓ trifft. Sei also T ein kd-Baum und ℓ eine vertikale Gerade. Sei $\ell(\text{root}(T))$ die teilende Gerade bei der Wurzel root(T) des Baums. Dann trifft ℓ genau eine der beiden Bereiche region(l(root(T))) bzw. region(r(root(T))). Achtung: Dies hat nicht die Rekursion $Q(n) = 1 + Q(n/2)$ zur Folge, denn im nächsten Schritt wird ℓ beide der horizontalen getrennten Bereich auf einer Seite treffen! Wir definieren $Q(n)$ um: Sei $Q(n)$ die Anzahl der ℓ treffenden Regionen eines kd-Baums mit n-Blättern der mit einer vertikalen Aufspaltungsgerade beginnt.

Jeder der 4 Knoten der Tiefe 2 beschreibt eine Region mit $n/4$ Punkte (Genauer: mit höchstens $\lceil\lceil n/2\rceil/2\rceil \leq \lceil n/4\rceil$ Punkte). Wobei genau 2 der Bereiche ℓ treffen. Weiters trifft ℓ die Region der Wurzel und auch eines ihrer Kinder:

$$Q(n) = \begin{cases} O(1) & \text{für } n = 1 \\ 2 + 2Q(n/4) & \text{für } n > 1 \end{cases}$$

Die Lösung dieser Rekursionsvorschrift ist $Q(n) = O(\sqrt{n})$ (siehe Übungen). Das Gleiche gilt für horizontale Geraden. Somit ist die Gesamtanzahl der Regionen, die den

Rand eines rechteckigen Suchbereichs treffen, durch $O(\sqrt{n})$ beschränkt. □
Beachte allerdings, daß die Abschätzung grob ist, da wir nicht wirklich den Schnitt mit den Geraden sondern nur mit den eventuell viel kürzeren Seitenkanten von R betrachten müßten.

5.5 Theorem [dBvKOS97, S.111]**.**
Der Aufbau eines kd-Baums für eine n-elementige Menge $P \subseteq \mathbb{R}^2$ benötigt $O(n)$ Speicher und $O(n \log n)$ Laufzeit. Eine orthogonale rechteckige Bereichsabfrage benötigt dann $O(k + \sqrt{n})$ Zeit, wobei k die Anzahl der ausgegebenen Punkte ist. □

Höhere Dimensionen [dBvKOS97, S.111]

Es können *kd*-Bäume auch für höhere Dimensionen als 2 verwendet werden. Dabei zerlegen wir die Bereiche längs Hyperebenen in 2 Teile. Dies müssen wir nach einander für Hyperebenen mit Richtungsvektor $e_1, \ldots, e_d$ tun. Wieder benötigt der (binäre) *kd*-Baum $O(n)$ Speicher und $O(n \log n)$ Aufbauzeit (bei fixer Dimension). Man kann zeigen, daß die Suchabfrage dann in $O(k + n^{1-1/d})$ Laufzeit erfolgt.

Laufzeit versus Speicherbedarf, [dBvKOS97, S.111]

Wir wollen nun anstelle der *kd*-Bäume sogenannte Bereichsbäume besprechen, die eine Bereichssuche mit $O(k+(\log n)^2)$-Laufzeit (statt $O(k+\sqrt{n})$) erlauben, allerdings einen Speicherbedarf von $O(n \log n)$ (statt $O(n)$) haben.

Bereichsbäume, [dBvKOS97, S.112]

Sei $P(v)$ die Menge der in den Blättern des Teilbaums mit Wurzel v gespeicherten Punkte (die sogenannte kanonische Teilmenge zu v). Die Menge der 1-dimensionalen Punkte, die in einem Bereichsintervall liegen, ist die disjunkte Vereinigung von $O(\log n)$ vielen kanonischen Teilmengen von P. Für 2-dimensionale Punkte machen wir nun zuerst eine 1-dimensionale Bereichssuche bzgl. der x-Koordinate. Letztlich sind wir aber nur an jenen Punkten der kanonischen Teilmengen interessiert, deren y-Koordinate im gewünschten Bereich $[y, y']$ liegt. Um diese effektiv zu finden, benötigen wir einen binären Suchbaum für die y-Koordinate der Punkte in $P(v)$. Wir betrachten also folgende Datenstruktur eines sogenannten Bereichsbaums (engl. range tree):

- Ein balanzierter Suchbaum T für die x-Koordinaten der Punkte in P.
- Für jeden seiner inneren Knoten v einen Pointer zu einem balanzierten binären Suchbaum $T(v)$ für die y-Koordinate der Punkte in der kanonischen Teilmenge $P(v) \subseteq P$.

Weitere multilevel Datenstrukturen werden uns in Kapitel 10 und eventuell auch in 16 begegnen.

Algorithmus Build2dRangeTree [dBvKOS97, S.113]

Input: Eine n-elementige Menge $P \subseteq \mathbb{R}^2$.
Output: Die Wurzel eines 2-dimensionalen Bereichbaums.
1: Bilde einen binären Suchbaum T_{assoz} für die y-Koordinaten der Punkte in P. Speichere dazu in den Blättern nicht nur die y-Koordinate sondern auch die x-Koordinate der Punkte.

2: **if** $P = \{p\}$ **then**
3: Erzeuge ein Blatt v mit p als Wert und setze T_{assoz} für die assoziierte Struktur zu v.
4: **else**
5: Zerlege P in $P_l := \{p \in P : p_x \leq x_{\text{mid}}\}$ und $P_r := \{p \in P : p_x > x_{\text{mid}}\}$.
6: Erzeuge einen neuen Knoten v mit Wert x_{mid} und setze für seine Kinder:
7: $\text{l}(v) :=$Build2dRangeTree(P_l)
8: $\text{r}(v) :=$Build2dRangeTree(P_r)
9: Setze T_{assoz} als assozierte Struktur für v.
10: **return** v

5.6 Lemma (Speicherbedarf) [dBvKOS97, S.114]**.**
Ein Bereichsbaum für n Punkte in der Ebene hat $O(n \log n)$ Speicherbedarf.

Beweis. Die Punkte p werden nur in den Blättern der assoziierten Strukturen von Knoten auf den Suchpfad für p_x gespeichert. Also wird p für alle Knoten mit gegebener Tiefe t genau einmal gespeichert. Da 1-dimensionale Bereichsbäume linearen Speicherbedarf haben, haben alle assoziierten Strukturen zu den Knoten fixer Tiefe zusammen $O(n)$ Speicherbedarf. Die Tiefe von T ist $O(\log n)$ und somit der Gesamtspeicherbedarf $O(n \log n)$. □

Aufbauzeit, [dBvKOS97, S.114]

Build2dRangeTree hat nicht Konstruktionszeit $O(n \log n)$, denn die Konstruktion der binären Suchbäume in Schritt (1) benötigt allein $O(n \log n)$-Zeit. Wenn wir aber die Punkte nach y-Koordinate in $O(n \log n)$ Zeit vorsortieren, dann können die binären Suchbäume T_{assoz} in linearer Zeit konstruiert werden. Darum werden wir die Punkte sowohl in einer nach x-Koordinate als auch unabhängig in einer nach der y-Koordinate sortierten Liste mitschleifen. Damit ist die Laufzeit bei jedem Knoten v von T linear in der Größe von $P(v)$ und somit die Gesamtkonstruktionszeit wie der Speicherbedarf ein $O(n \log n)$.

Abfrage-Algorithmus [dBvKOS97, S.114]

Zuerst bestimmen wir mittels 1-dimensionaler Suchabfrage die $O(\log n)$ vielen kanonischen Mengen, welche die Punkte p mit x-Koordinate in $[x, x']$ haben. Für jede dieser Mengen liefern wir dann mittels 1-dimensionaler Suchabfrage über den assoziierten Baum die Punkte mit y-Koordinate in $[y, y']$. Der Algorithmus ist also wie 1dRangeQuery wobei ReportSubtree durch 1dRangeQuery zu ersetzen ist:

Algorithmus 2dRangeQuery [dBvKOS97, S.114]

Input: Ein 2-dimensionaler Bereichsbaum T und ein Bereich $R := [x, x'] \times [y, y']$.
Output: Alle Punkte P von T die in R liegen.
1: $v_{\text{split}} :=$FindSplitNode(T, x, x').
2: **if** v_{split} ist Blatt **then**
3: Überprüfe, ob der Punkt von v_{split} in R liegt und somit ausgegeben werden muß.
4: **else**

// Folge dem Suchpfad zu x und rufe 1dRangeQuery für die rechts davon liegenden Teilbäume auf.
$v := \mathrm{l}(v_{\text{split}})$
while v ist nicht Blatt **do**
  if $x \leq x_v$ **then**
    1dRangeQuery$(T_{\text{assoz}}(\mathrm{r}(v)), [y, y'])$
    $v := \mathrm{l}(v)$
  else
    $v := \mathrm{r}(v)$
Überprüfe, ob $v \in [x, x']$ und somit ausgegeben werden muß.
Verfahre analog mit dem Suchpfad von v_{split} nach x'

Tag 5.03

5.7 Lemma (Abfragezeit) [dBvKOS97, S.115]**.**
Die Abfrage in einem Bereichsbaum mit n-Ecken für ein achsenparalleles Rechteck benötigt $O(k + (\log n)^2)$ Zeit, wobei k die Anzahl der berichteten Punkte ist.

Beweis. Bei jedem Knoten $v \in T$ benötigen wir konstante Zeit um festzustellen bei welchem Kind wir weitersuchen müssen und wir rufen möglicherweise 1dRangeQuery auf. Laut Theorem 5.2 ist die Laufzeit für diesen Aufruf $O(k_v + \log n)$, wobei k_v die Anzahl der im Aufruf berichteten Punkte ist. Somit ist die Gesamtlaufzeit:

$$\sum_v O(k_v + \log n),$$

wobei die Summe über alle besichtigten Knoten v von T geht. Klarerweise ist $\sum_v k_v = k$, die Gesamtanzahl der berichteten Punkte. Die Suchpfade für x und x' haben Länge $O(\log n)$ und somit ist $\sum_v O(\log n) = O((\log n)^2)$ □

5.8 Theorem [dBvKOS97, S.115]**.**
Sei $P \subseteq \mathbb{R}^2$ eine n-elementige Menge. Ein Bereichsbaum für P benötigt $O(n \log n)$ Speicher und $O(n \log n)$ Aufbauzeit. Die Bereichsabfrage für achsenparallele rechteckige Bereiche benötigt $O(k + (\log n)^2)$ Laufzeit, wobei k die Zahl der berichteten Punkte ist. □

Höher-dimensionale Bereichsbäume [dBvKOS97, S.115]

Die Abfragezeit werden wir im Abschnitt über fraktionelles Cascading auf $O(k+\log n)$ drücken.

Dimensionen größer als 2, [dBvKOS97, S.115]

Bereichsbäume lassen sich leicht auf Dimensionen $d > 2$ verallgemeinern: Sei $P \subseteq \mathbb{R}^d$ endlich. Wir konstruieren einen balanzierten binären Suchbaum T für die 1. Koordinate der Punkte aus p. Für jeden seiner Knoten konstruieren wir rekursiv die assoziierte Struktur $T(v)$, einen $(d-1)$-dimensionalen Bereichsbaum für die Projektion der Punkte in $P(v)$, die in den Blättern des Teilbaums von T mit Wurzel v gespeichert sind, auf die letzten $d-1$-Koordinatenachsen. Der Abfragealgorithmus ist analog zum

2-dimensionalen Fall: Wir lokalisieren die $O(\log n)$ vielen Knoten v in T, deren kanonische Teilmengen $P(v)$ nur Punkte enthält deren 1. Koordinate im korrekten Bereich liegt. Für jede diese kanonischen Teilmengen (auf Level 1) werden dann seinerseits die $O(\log n)$ vielen Knoten v' in $T(v)$ lokalisiert, deren kanonische Teilmengen $P(v')$ nur Punkte enthält deren 2. Koordinate im korrekten Bereich liegt. Insgesamt haben wir somit $O((\log n)^2)$ kanonische Teilmengen auf Level 2, und so weiter.

5.9 Theorem [dBvKOS97, S.116]**.**
Sei $P \subseteq \mathbb{R}^d$ mit $d \geq 2$ eine n-elementige Menge. Ein Bereichsbaum für P benötigt $O(n\,(\log n)^{d-1})$-Speicher und $O(n\,(\log n)^{d-1})$ Aufbauzeit. Bereichsabfragen für achsenparallele rechteckige Bereiche benötigt $O(k + (\log n)^d)$ Zeit, wobei k die Anzahl der berichteten Punkte ist.

Beweis. Sei $T_d(n)$ die Aufbauzeit für Bereichsbäume von n Punkten im $\mathbb{R}^d$. Nach Theorem 5.8 ist $T_2(n) = O(n \log n)$. Für einen d-dimensionalen Bereichsbaum müssen wir einen balanzierten binären Suchbaum (in $O(n \log n)$ Zeit) aufbauen. Nun zu den assoziierten Strukturen: Für die Knoten von T von fixer Tiefe wird jeder Punkt in genau einer assoziierten Struktur gespeichert (siehe 5.6). Also ist die Gesamtkonstruktionszeit der assoziierten Strukturen dieser Knoten $O(T_{d-1}(n))$. Die Gesamtkonstruktionszeit für Bereichsbäume erfüllt somit:

$$T_d(n) = O(n \log n) + O(\log n) \cdot T_{d-1}(n).$$

Wegen $T_2(n) = O(n \log n)$ folgt $T_d(n) = O(n\,(\log n)^{d-1})$. Der Speicherbedarf ergibt sich auf die gleiche Weise (vgl. mit dem Beweis in 5.6).

Sei $Q_d(n)$ die Laufzeit für die d-dimensionale Suche unter n Punkten ohne die für die Ausgabe benötigte Zeit. Die d-dimensionale Suchabfrage benötigt eine Suche im Baum T von $O(\log n)$ Zeit und die Abfrage von $\log(n)$ vielen $(d-1)$-dimensionalen Bereichsbäumen. Also gilt

$$Q_d(n) = O(\log n) + O(\log n) \cdot Q_{d-1}(n)$$

und wegen $Q_2(n) = O((\log n)^2)$ (siehe 5.7) folgt $Q_d(n) = O((\log n)^d)$. Wenn wir die für die Ausgabe der k Punkte benötigte Zeit $O(k)$ berücksichtigen, so erhalten wir die behauptete Gesamtabfragezeit. □
Wie für das 2-dimensionale werden wir das noch durch einen logarithmischen Faktor verbessern.

Allgemeine Punkte

Behandlung degenerierter Situationen, [dBvKOS97, S.117]

Wir können die Einschränkung, daß die Punkte in P nicht die gleiche x- oder y-Koordinate haben, dadurch beseitigen, daß wir Punkte $p = (p_x, p_y)$ durch $\hat{p} := ((p_x, p_y), (p_y, p_x))$ ersetzen, wobei wir die "Koordinaten" (p_x, p_y) bzw. (p_y, p_x) lexikographisch ordnen. Da wir mit diesen nicht gerechnet haben, sondern nur eine lineare Ordnung verwendet haben, können wir kd-Bäume für $\hat{P} := \{\hat{p} : p \in P\}$ bilden und die Suchabfrage für $R := [x, x'] \times [y, y']$ durch jene für $\hat{R} := [(x, -\infty), (x', +\infty)] \times [(y, -\infty), (y', +\infty)]$ ersetzen.

5.10 Lemma, [dBvKOS97, S.117]**.**
Sei $p \in \mathbb{R}^2$ und R ein achsenparalleles Rechteck. Dann gilt:

$$p \in R \Leftrightarrow \hat{p} \in \hat{R}.$$

Beweis. Sei $R = [x, x'] \times [y, y']$ und $p = (p_x, p_y)$. Dann ist $p \in R \Leftrightarrow x \leq p_x \leq x'$ und $y \leq p_y \leq y'$, also genau dann, wenn $(x, -\infty) \leq (p_x, p_y) \leq (x', +\infty)$ und $(y, -\infty) \leq (p_y, p_x) \leq (y', +\infty)$, d.h. $\hat{p} \in \hat{R}$ ist. □

Beachte, daß wir nicht wirklich die transformierten Punkte $\hat{p}$ speichern müssen, sondern nur beim Vergleichen die entsprechende lexikographische Ordnung verwenden müssen. Dies läßt sich offensichtlich auch auf höhere Dimensionen übertragen.

Fraktionelles Cascading

Zeitreduktion bei Suche in assoziierten Strukturen, [dBvKOS97, S.118]

Bereichsbäume ermöglichten die orthogonale Bereichsabfrage in $O(k + (\log n)^2)$-Zeit. Mittels des nun zu beschreibenden fraktionellen cascading wollen wir die Zeit auf $O(k + \log n)$ drücken. Dazu versuchen wir die Zeit für die Suche in den assoziierten Strukturen $T(v)$ von $O(k_v + \log n)$, wobei k_v die Anzahl der ausgegebenen Punkte ist, auf $O(k_v + 1)$ zu verringern, was eine Gesamtsuchzeit von $O(k + \log n)$ zur Folge hätte. Beantwortung einer 1-dim. Bereichsabfrage ist im allgemeinen nicht in $O(k + 1)$-Zeit möglich. Hier allerdings können wir verwenden, daß wir viele solche Abfragen mit dem gleichen Suchbereich machen müssen.

Ein einfaches Beispiel, [dBvKOS97, S.118]

Seien A_1 und A_2 zwei sortierte Arrays. Wir wollen alle ihre Elemente ausgeben, die in einem Intervall $[y, y']$ liegen:

- Zuerst suchen wir nach y in A_1 nach den kleinsten Eintrag $\geq y$.
- Dann durchlaufen wir von dort die Elemente von A_1 und geben diese aus, bis wir auf einen Eintrag $> y'$ stoßen.
- Analog gehen wir mit A_2 vor.

Falls die Gesamtanzahl der ausgegebenen Punkte k ist, so ist die Abfragezeit $O(k)$ plus der beiden Suchzeiten in A_1 und A_2. Falls aber $A_2 \subseteq A_1$, so können wir uns die Suche in A_2 sparen:

- Für jedes $A_1[i]$ speichern wir einen Pointer auf den kleinsten Wert in A_2 welcher $\geq A_1[i]$ ist (bzw. auf nil, falls so einer nicht existiert).
- Die Suche in A_2 geht nun wie folgt: Sei i die Stelle von A_1, wo die Suche nach y endet. Dann zeigt der zugehörige Pointer auf den kleinsten Wert in A_2 welcher $\geq y$ ist, und wir können dort mit der Ausgabe beginnen.

Geschichteter Bereichsbaum [dBvKOS97, S.119]

Beachte, daß für die kanonischen Mengen $P(l(v)), P(r(v)) \subseteq P(v)$ gilt. Anstatt einen binären Suchbaum für die assoziierte Struktur zu verwenden, speichern wir $P(v)$ nun in einem Array $A(v)$ sortiert nach der y-Koordinate. Weiters für jeden Eintrag $A(v)[i]$ in $A(v)$ Pointer zu den Elementen in $A(l(v))$ und $A(r(v))$ mit kleinster y-Koordinate

größer oder gleich jener von $A(v)[i]$. Ein so modifizierter Bereichsbaum heißt layered range tree.

Suchabfrage in einem geschichteten Bereichsbaum, [dBvKOS97, S.120]

- Zuerst suchen wir nach x und x' im Hauptbaum T um die $O(\log n)$ vielen Knoten zu bestimmen, deren kanonische Mengen aus Punkten mit x-Koordinate in $[x, x']$ bestehen.
- Bei v_{split} suchen wir mittels binärer Suche nach dem Eintrag in $A(v_{\text{split}})$ mit minimaler y-Koordinate $\geq y$ in $O(\log n)$ Zeit.
- Während wir weiter nach x und x' suchen, verfolgen wir laufend den Eintrag in den assoziierten Arrays, mit minimaler y-Koordinate $\geq y$. Mittels der vorhandenen Pointer kann das in konstanter Zeit erledigt werden.
- Sei v einer der selektierten Knoten. Um die Punkte in $A(v)$ auszugeben deren y-Koordinate in $[y, y']$ liegt, müssen wir nur das Array von dort an solange ausgeben bis wir auf einen Wert $> y'$ stoßen. Um also die k_v vielen Punkte in $A(v)$ mit y-Koordinate in $[y, y']$ auszugeben benötigen wir $O(k_v + 1)$ Zeit und die Gesamtzeit wird damit $O(k + 3 \log n) = O(k + \log n)$.

Mehrdimensionales Fractional Cascading [dBvKOS97, S.121]

Auch im mehrdimensionalen verbessert fractional cascading die Suchabfrage um einen logarithmischen Faktor:

5.11 Theorem [dBvKOS97, S.121]**.**
Sei $d \geq 2$ und $P \subseteq \mathbb{R}^d$ eine n-elementige Teilmenge. Ein geschichteter Bereichsbaum für P benötigt $O(n (\log n)^{d-1})$ Platz und $O(n (\log n)^{d-1})$ Konstruktionszeit. Die Abfragezeit für orthogonale Bereichssuche ist $O(k + (\log n)^{d-1})$, wobei k die Anzahl der ausgegebenen Punkte ist. □

Nachbemerkungen

[dBvKOS97, S.121]

Orthogonale Bereichsabfrage war ein sehr wichtiges Thema in den Anfangstagen der algorithmischen Geometrie und entsprechend viele Resultate gibt es.

Eine der ersten Datenstrukturen dafür waren die quadtrees (siehe Kapitel 14). Diese haben sehr schlechtes schlimmster-Fall-Verhalten. Kd-Bäume wurden von [Ben75] eingeführt. Ein paar Jahre später die range-trees durch [Ben79], [LW80], [Lue78] und [Wil79]. Die Verbesserung der Abfragezeit mittels fractional cascading stammt von [Lue78] und [Wil78]. Die effektivste Datenstruktur für 2-dimensionale Bereichsabfrage ist eine modifizierte Version der layered range trees von [Cha86] mit Speicherplatz $O(n \log n / \log(\log n))$ mit Abfragezeit $O(k + \log n)$. In [Cha90b] und [Cha90c] wurde gezeigt, daß dies optimal ist. Für das Mehrdimensionale stammt das optimale Resultat von [Cha90b].

6. Ortsbestimmung

Tag 5.05

Ortsbestimmung, [dBvKOS97, S.127]

Gegeben sei eine Karte, also eine Unterteilung der Ebene in Regionen, sowie die Koordinaten eines Punktes. Gesucht ist jene Region, die den Punkt enthält. Anwendung z.B. in der Seefahrt, wo man wissen will, wie die Strömung an der aktuellen Position ist. Diese Information sollte kontinuierliche upgedated werden. Dazu sollten wir die Karte in so einer Datenstruktur aufbereiten, daß die Suchabfragen schnell beantwortet werden können. Diese Aufgabenstellung tritt auch auf, wenn wir in einen GIS entsprechende Informationen über den Bereich, in welchem sich der Mauszeiger befindet, erhalten wollen.

Punktpositionen und Trapezische Karten

Punktpositionsproblem in der Ebene, [dBvKOS97, S.128]

Sei S eine Unterteilung der Ebene mit n Kanten. Der Abfragealgorithmus soll für Punkte q jene Fläche (bzw. jene Kante oder Ecke) von S liefern in welcher er liegt.

Ein erster Versuch, [dBvKOS97, S.128]

Wir sortieren die x-Koordinaten. Dann können wir in $O(\log n)$ Zeit den Streifen bestimmen, der durch vertikale Geraden durch Ecken von S begrenzt ist und den gegebenen Punkt enthält. Im Inneren dieser Streifen liegt keine Ecke (und somit auch kein Schnittpunkt der Kanten), also können wir die Kanten, die ihm in Inneren treffen, nach Höhe sortieren. Wir merken uns zu jeder Kante die Fläche von S, die unmittelbar oberhalb im Streifen liegt.

Abfrage [dBvKOS97, S.129]

- Zuerst machen wir einen binäre Suche nach der x-Koordinate des gesuchten Punktes q unter den x-Koordinaten der Ecken von S.
- Danach machen wir eine binäre Suche im entsprechenden Streifen. Dazu müssen wir für Kanten e, die durch den Streifen laufen, feststellen, ob q oberhalb oder unterhalb liegt.
- Die zur unmittelbar unterhalb gelegenen Kante e assoziierte Fläche ist dann der gesuchte Bereich. Falls wir keine solche Kante finden, so liegt q im unbeschränkten Bereich.

Suchzeit [dBvKOS97, S.129]

Wir müssen zwei Binärsuchen machen. Die erste ist über höchstens $2n$ Eintragungen (Jede der n-Kanten hat 2 Endpunkte), die zweite ist über höchstens n Eintragungen. Also ist die Suchzeit $O(\log n)$.

Speicherbedarf [dBvKOS97, S.129]

Das erste Suchstruktur benötigt $O(n)$ Speicherplatz. Die Suchstruktur für jeden Streifen benötigt ebenfalls jeweils maximal $O(n)$ Speicherplatz. Zusammen für die $O(n)$ vielen Streifen also insgesamt $O(n^2)$ Speicherplatz. Beispiele mit $n/4$ Streifen, wo jeder von $n/4$ der Kanten getroffen wird zeigen, daß dies optimal ist, siehe [dBvKOS97, S.129]. Grund für den (uns zu) großen Speicherbedarf ist also, daß die Unterteilung in Streifen eine Verfeinerung S' von S mit bisweilen quadratischer Komplexität in trapezförmige Flächen darstellt. Wir wollen nun eine andere trapezoidale Verfeinerung (d.h. mit trapezförmigen Flächen) angeben, welche schnellere Punktsuche garantiert und deren Komplexität dennoch nicht viel größer als jene von S ist.

Trapezoidale Verfeinerung [dBvKOS97, S.130]

Wir nennen zwei Geradensegmente nicht-kreuzend, wenn ihr Durchschnitt höchstens aus einem gemeinsamen Endpunkt besteht. Die Kanten einer Unterteilung S sind jedenfalls nicht-kreuzend. Sei allgemeiner S irgendeine Menge nicht kreuzender Geradensegmente, wobei wir vorerst zusätzlich voraussetzen, daß S ganz in einem großen achsenparallelen Rechteck R enthalten ist und keine zwei verschiedenen Endpunkte die selbe x-Koordinate haben (also insbesonders keine vertikalen Segmente vorliegen). Wir sagen kurz S sei in allgemeiner Lage, wenn diese Annahmen erfüllt sind. Die trapezoidale Verfeinerung $T(S)$ erhalten wir, indem wir bei jedem Endpunkt zwei vertikale Segmente (die sogenannten vertikalen Erweiterung) nach oben und unten hinzufügen, deren anderer Endpunkt der Schnittpunkt mit dem jeweils ersten Segment von S ist, auf welches sie stoßen. Die Flächen von $T(S)$ sind durch Kanten von $T(S)$ berandet, und wir fassen die aneinanderhängenden kollinearen unter ihnen zu sogenannten Seiten der Flächen zusammen, siehe [dBvKOS97, S.131]

6.1 Lemma [dBvKOS97, S.131]**.**
Jede Fläche f in $T(S)$ hat eine oder zwei vertikale Seiten und genau 2 nicht-vertikale Seiten (die wir $\mathrm{top}(f)$ *und* $\mathrm{bot}(f)$ *nennen), ist also ein (möglicherweise degeneriertes) Trapez.*

Beweis. Wir zeigen zuerst, daß f konvex ist: Da die Segmente von S nicht-kreuzend sind, ist jede Ecke von f entweder ein Endpunkt einer Seite von S (oder R), oder ein Endpunkt einer vertikalen Erweiterung. Der innere Winkel bei den Endpunkten der Segmente muß wegen der davon ausgehenden vertikalen Erweiterungen $\leq \pi$ sein und Gleiches gilt auch für die Endpunkte der vertikalen Erweiterungen. In den Ecken von R ist der Winkel $\pi/2$. Somit ist f konvex. Wegen der Konvexität kann f höchstens 2 vertikale Seiten besitzen. Angenommen f habe mehr als 2 nicht-vertikale Seiten. Dann müßten unter ihnen zwei aufeinanderfolgende existieren, für welche f bei beiden oberhalb oder unterhalb liegt. Da die nicht-vertikalen Seiten in Segmenten von S enthalten sein müssen, müssen sich diese beiden in einem Endpunkt eines Segments von S treffen. Die von diesem ausgehenden vertikalen Erweiterungen widersprechen dann aber dem Aufeinanderfolgen der beiden Seiten. Da $f \subseteq R$ und somit beschränkt ist, müssen (mindestens) 2 nicht-vertikale (obere & untere) Seiten vorliegen und mindestens eine vertikale. □

[dBvKOS97, S.132]

Da S in allgemeiner Lage vorausgesetzt wurde, bestehen die vertikalen Seiten der Flächen entweder aus vertikalen Erweiterungen oder vertikalen Seiten von R. Genauer kann für die linke vertikale Seite von f genau einer der folgenden Fälle eintreten:

1. Sie ist zu einem Punkt degeneriert, d.h. top(f) und bot(f) haben den selben linken Endpunkt.
2. Sie ist die untere vertikale Erweiterung des linken Endpunkts von top(f).
3. Sie ist die obere vertikale Erweiterung des linken Endpunkts von bot(f).
4. Sie besteht aus beiden vertikalen Erweiterungen eines Segments von S.
5. Sie ist die linke Seitenkante von R (dies tritt nur bei der am linkesten gelegenen Fläche von $T(S)$ auf)

Bis auf den letzten Fall, wird die linke vertikale Seite durch einen Endpunkt eines Segments in S beschrieben, wir nennen diesen leftp(f), siehe [dBvKOS97, S.132]. Für das Ausnahmetrapez sei leftp(f) die untere linke Ecke von R. Analog definiert man rightp(f). Beachte, daß f durch leftp(f), rightp(f), top(f) und bot(f) eindeutig festgelegt ist.

6.2 Lemma. Komplexität der trapezoidalen Verfeinerung [dBvKOS97, S.133].
Die trapezoidale Verfeinerung $T(S)$ einer Menge S von n Geradensegmenten in allgemeiner Lage hat höchstens $6n + 4$ Ecken und $3n + 1$ Trapeze.

Beweis. Ecken von $T(S)$ sind entweder Ecken (4) von R, oder Endpunkte ($\leq 2n$) von Segmenten von S oder Endpunkte ($\leq 2 \cdot 2n$) einer vertikalen Erweiterung. Zusammen also $\leq 4 + 6n$.

Wir ordnen jedem Trapez f (bis auf dem aus Fall 5) das leftp(f) definierende Segment zu, und zwar im Fall 1 bot(f). Dabei kann für jedes Segment e der rechte Endpunkt gleich leftp(f) nur für ein f sein, und der linke Endpunkt gleich leftp(f) nur für höchstens zwei f (mit top(f) $\subseteq e$ oder bot(f) $\subseteq e$) sein. Also ist die Maximalanzahl von Trapezen $n + 2n + 1$. □

Datenstruktur [dBvKOS97, S.133]

Wir nennen zwei Trapeze in $T(S)$ benachbart, wenn sie eine gemeinsame vertikale Kante haben. Jedes Trapez hat höchstens 4 benachbarte. Falls ein Trapez f' benachbart zu f längs der linken vertikalen Seite von f ist, dann ist entweder top(f') = top(f) oder bot(f') = bot(f), und wir nennen f' entsprechend oberer/unterer linker Nachbar von f. Analog für rechte Nachbarn. Anstelle der doppeltverbundenen Kantenliste aus Kapitel 2 wollen wir für trapezoidale Verfeinerungen eine besser angepaßte Datenstruktur verwenden:

- Records für alle Geradensegmente von S und deren Endpunkte.
- Records für die Trapeze f von $T(S)$ beschrieben durch 4 Pointer zu top(f), bot(f), leftp(f), rightp(f), sowie Pointer zu den 4 möglichen Nachbarn.

Diese Datenstruktur enthält die Koordinaten der Ecken der Trapeze nicht explizit, aber wir können diese Information in konstanter Zeit aus dem entsprechenden Record extrahieren.

Tag 5.06

Ein randomizierter inkrementeller Algorithmus, [dBvKOS97, S.134]

Wir wollen nun einen Algorithmus beschreiben, der die trapezoidale Verfeinerung $T(S)$ konstruiert und gleichzeitig eine Datenstruktur D aufbaut, die wir für Punktpositions-Bestimmungen verwenden können. Letzteres würde von einem planesweep Algorithmus nicht geliefert, darum gehen wir anders vor. Die Suchstruktur D ist ein gerichteter azyklischer Graph mit einziger Wurzel und mit genau einem Blatt für jedes Trapez von $T(S)$. Die inneren Knoten haben 2 ausgehende Kanten. Es gibt zwei Typen von ihnen, siehe [dBvKOS97, S.135]:

- x-Knoten mit einem Endpunkt eines Segments von S als Daten.
- y-Knoten mit den Segmenten selbst als Daten.

Ein Abfrage mit Punkt q started an der Wurzel und schreitet in Richtung jenes Blattes fort, dessen Trapez q enthält. Dabei wird für das Weitergehen in x-Knoten getestet, ob q links oder rechts von der vertikalen Gerade durch den dort gespeicherten Endpunkt liegt, und in den y-Knoten wir getestet, ob q ober- oder unterhalb des dort gespeicherten Segments liegt. Dazu müssen wir sicher stellen, daß wenn wir einen y-Knoten erreichen die vertikale Gerade durch q vom entsprechenden Segment getroffen wird. Um die Tests eindeutig zu machen, setzen vorerst voraus, daß q bei x-Knoten nicht auf der vertikalen Gerade und bei y-Knoten nicht auf dem Segment liegt. Im nächsten Abschnitt werden wir all diese Zusatzvoraussetzungen loswerden.

Verlinkung von T und D, [dBvKOS97, S.135]

Die Suchstruktur D und die Datenstruktur von $T(S)$ werden miteinander verbunden: Jedes Trapez in $T(S)$ hat einen Pointer zu dem entsprechenden Blatt in D und vice cersa.

Inkrementeller Aufbau, [dBvKOS97, S.135]

Der Aufbau von T und D geschieht inkrementell, d.h. Segmente werden nacheinander hinzugefügt. Da die Reihenfolge die Datenstruktur beeinflußt, verwenden wir wieder eine zufällige Anordnung:

Algorithmus TrapezoidalMap [dBvKOS97, S.135]

Input: Eine Menge S von n nicht-kreuzenden Geradensegmenten in $\mathbb{R}^2$.
Output: Die trapezoidale Verfeinerung T von S und eine Suchstruktur D für T in einem umfassenden Rechteck R.

1: Finde ein umfassendes Rechteck R und initializiere T und D damit.
2: Sei $s_1, \ldots, s_n$ eine Zufallspermutation von S.
3: **for** $i := 1$ to n **do**
4: Finde die Trapeze von T, die durch s_i zerteilt werden.
5: Ersetze diese Trapeze in T durch die entstehenden Teile und ersetze die entsprechenden Blätter in D durch Blätter für die neu entstandenen Trapeze. Verlinke diese Blätter mit den inneren Knoten indem neue innere Knoten (wie noch zu besprechen) hinzugefügt werden.

Wir beschreiben dies nun im Detail, wobei $S_i := \{s_1, \ldots, s_i\}$ und T immer die trapezoidale Verfeinerung von S_i und D die zugehörige Suchstruktur bezeichnet. Dabei besteht T für S_0 aus einem einzigen Trapez, nämlich R. Und ebenso besteht die Suchstruktur D aus einem einziges Blatt.

[dBvKOS97, S.136]

Beim Übergang von S_{i-1} zu S_i müssen wir nur jene Trapeze ersetzen, die von s_i zerteilt werden. Wir ordnen diese nach ihrem Schnitt mit s_i von links nach rechts als $f_0, \ldots, f_k$. Da sich die Segmente nicht kreuzen, muß f_j einer der rechten Nachbarn von f_{j-1} sein. Und zwar genau dann der untere, falls rightp(f_{j-1}) oberhalb s_i liegt. Nachdem wir f_0 haben, bekommen wir die übrigen f_j also aus den Daten in T. Wir müssen somit jenes $f_0 \in T$ finden, welches den linken Endpunkt p von s_i enthält: Falls p nicht Endpunkt eines Segments aus S_{i-1} ist, so muß er im Inneren von f_0 liegen. Also können wir f_0 durch eine Punktsuche in $T(S_{i-1})$ mittels D bestimmen. Andernfalls kommt p während der Suche in $T(S_{i-1})$ bei einem x-Knoten auf der vertikalen Geraden durch den Punkt zu liegen, was wir ausgeschlossen haben. Um dies also zu verhindern, betrachten wir p als einen etwas weiter rechts liegenden Punkt, d.h. wenn er auf einer Geraden zu liegen kommt, betrachten wir ihm als rechts von dieser liegend. Analog, wenn er bei einem y-Knoten auf dem entsprechenden Segment s liegt (dann muß p der gemeinsame linke Endpunkt von s_i und s sein) so vergleichen wir die Anstiege von s_i und s und betrachten p genau dann als oberhalb s liegend, wenn ersterer Anstieg der größere ist.

Algorithmus FollowSegment[dBvKOS97, S.137]

Input: Ein trapezoidale Unterteilung T, eine Suchstruktur D für T und ein weiteres Segment s.
Output: Die Trapeze $f_0, \ldots, f_k$ von T, welche durch s zerteilt werden.

1: Sei p und q der linke und rechte Endpunkt von s.
2: Sei $j := 0$ und f_j das Ergebnis der Suche für p in D.
3: **while** q liegt rechts von rightp(f_j) **do**
4: **if** rightp(f_j) liegt oberhalb s **then**
5: f_{j+1} ist untere rechte Nachbar von f_j
6: **else**
7: f_{j+1} ist obere rechte Nachbar von f_j
8: $j := j + 1$
9: **return** $f_0, f_1, \ldots$.

Updaten von T und D, [dBvKOS97, S.137]

Betrachten wir vorerst den Fall $s \subseteq f$ für ein Trapez $f \in T$. Dann müssen wir f in T durch 4 Trapeze ersetzen. Die hierfür nötigen Daten gewinnen wir in konstanter Zeit aus dem Segment s und der Information von f in T. Um D upzudaten müssen wir das Blatt für f durch einen kleinen Baum mit 4 Blättern ersetzen. Dieser hat zwei x-Knoten zum Testen beim linken und rechten Endpunkts von s sowie einen y-Knoten zum Testen bei s. Mit dieser Information können wir jenes der 4 Trapeze bestimmen, welches einen gegebenen Punkt in f enthält. Beachte die degenerierten Fälle, wo ein(beide) Endpunkt(e) von s gleich leftp(f) oder rightp(f) sind. Dann haben wir nur 2 oder 3 neue Trapeze und brauchen entsprechend weniger Knoten

zum Testen. Jedenfalls ist die Anzahl der neuen Knoten um 1 kleiner als jene der neuen Trapeze.

[dBvKOS97, S.138]

Nun der Fall, wo s mehr als ein Trapez trifft, siehe [dBvKOS97, S.138]. Seien $f_0, \ldots, f_k$ die getroffenen Trapeze. Um T upzudaten, müssen wir die vertikalen Erweiterungen in den Endpunkten von s bilden. Diese zerlegen f_0 und f_k in jeweils 3 neue Trapeze (falls die Endpunkte nicht schon vorhanden waren). Dann kürzen wir jene vertikalen Erweiterungen, welche auf s treffen, und müssen somit f_i durch niedrigere Trapeze ersetzen und den Teil auf der anderen Seite von s mit seinem entsprechenden linken Nachbarn verbinden. Für all dies benötigen wir lineare Zeit in der Anzahl k. Um D upzudaten müssen wir die Blätter der $f_0, \ldots, f_k$ entfernen und Blätter für die neuen Trapeze hinzufügen sowie zusätzliche innere Knoten: Falls f_0 den linken Endpunkt von s im Inneren hat müssen wir das Blatt für f_0 durch einen x-Knoten für den linken Endpunkt von s und einen y-Knoten für s ersetzen. Analog, falls f_k den rechten Endpunkt von s enthält. Die Blätter für $f_1, \ldots, f_{k-1}$ werden durch y-Knoten für das Segment s ersetzt. Die hinausgehenden Kanten der neuen inneren Knoten sollen auf die korrekten Blätter zeigen. Beachte, daß durch das Verschmelzen von Trapezen von T mehrere hereinkommende Kanten bei den Blättern der neuen Trapeze sein können. Wieder ist die Anzahl der neuen Knoten um 1 kleiner als jene der neuen Trapeze.

6.3 Theorem [dBvKOS97, S.139]**.**
Der Algorithmus TrapezoidalMap *generiert in erwarteter* $O(n \log n)$ *Zeit die trapezoidale Verfeinerung* $T(S)$ *einer Menge* S *von* n *Geradensegmenten in allgemeiner Lage sowie eine Suchstruktur* D *hierfür. Der erwartete Speicherbedarf für die Suchstruktur ist* $O(n)$ *und die erwartetet Suchzeit* $O(\log n)$*.*

Beweis. Korrektheit: Diese folgt rekursiv aus der oben genannten Invarianz, nämlich, daß T jeweils die trapezoidale Verfeinerung für S_i und D ein gültige Suchstruktur für T ist.

Tag 5.10

Nun zur Laufzeit und Speicherbedarf: Dabei ist die Ordnung wichtig, denn in schlimmsten Fall werden wir sehen, daß der Speicherbedarf quadratisch und die Suchzeit linear sein. **Suchzeit:** Sei q ein fixer Punkt für die Abfrage. Die Suchzeit für q ist linear in der Länge des Suchpfads für q in D. Durch Fallunterscheidungen sieht man leicht, daß die maximale Pfadlänge höchstens um 3 bei jeder der n Iterationen des Algorithmuses zunimmt. Also ist $O(3n)$ eine obere Schranke für die maximale Suchzeit.

Unter der erwarteten Suchzeit verstehen wir den Mittelwert der Suchzeiten in den $n!$ vielen Datenstrukturen, die sich auf Grund der Permutationen von S ergeben. Sei X_i die Zufallsvariable für die Anzahl der Knoten, die im i. Iterationsschritt eingefügt wurden. Die erwartete Pfandlänge ist somit (wegen der Linearität von E)

$$E\Big(\sum_{i=1}^{n} X_i\Big) = \sum_{i=1}^{n} E(X_i).$$

Wegen $X_i \leq 3$ ist $E(X_i) \leq 3P_i$, wobei P_i die Wahrscheinlichkeit dafür ist, daß am Suchpfad liegende Knoten beim i. Iterationsschritt eingefügt wurden, d.h. das Trapez $f_q(S_{i-1}) \in T(S_{i-1})$, welches q enthält, verschieden von $f_q(S_i)$ ist. Beachte dabei, daß alle Trapeze f die im i. Schritt eingefügt wurden an s_i angrenzen: Entweder ist $\text{top}(f)$ oder $\text{bot}(f)$ gleich s_i, oder es ist $\text{leftp}(f)$ oder $\text{rightp}(f)$ ein Endpunkt von s_i. Es ist $T(S_i)$ und damit $f_q(S_i)$ eindeutig durch S_i festgelegt und hängt nicht von der Ordnung der Segmente von S_i ab. Wir verwenden wieder Rückwärtsanalyse und bestimmen die Wahrscheinlichkeit dafür, daß $f_q(S_i)$ von T verschwindet, wenn wir das Segment s_i entfernen. Dies ist genau dann der Fall, wenn einer von $\text{top}(f_q(S_i))$, $\text{bot}(f_q(S_i))$, $\text{leftp}(f_q(S_i))$ oder $\text{rightp}(f_q(S_i))$ beim Entfernen von s_i verschwindet. Die Wahrscheinlichkeit für $s_i = \text{top}(f_q(S_i))$ ist $\leq 1/i$ (sie ist 0, wenn $\text{top}(f_q(S_i))$ die obere Kante von R ist). Analoges gilt für $\text{bot}(f_q(S_i))$. Hingegen kann die Wahrscheinlichkeit dafür, daß Segmente den Punkt $\text{leftp}(f_q(S_i))$ als Endpunkt haben groß sein. Aber $\text{leftp}(f_q(S_i))$ verschwindet nur dann, wenn s_i das einzige Segment in S_i mit $\text{leftp}(f_q(S_i))$ als Endpunkt ist. Also ist die Wahrscheinlichkeit dafür ebenfalls $\leq 1/i$ und Gleiches gilt für $\text{rightp}(f_q(S_i))$. Somit ist

$$P_i = P\Big(f_q(S_i) \neq f_q(S_{i-1})\Big) = P\Big(f_q(S_i) \notin T(S_{i-1})\Big) \leq 4 \cdot \frac{1}{i}$$

und folglich ist

$$E\Big(\sum_{i=1}^{n} X_i\Big) = \sum_{i=1}^{n} E(X_i) \leq \sum_{i=1}^{n} 3P_i \leq \sum_{i=1}^{n} \frac{12}{i} = 12H_n,$$

wobei $H_n := \sum_{i=1}^{n} \frac{1}{i}$ die n. Partialsumme der harmonischen Reihe ist. Durch Integralabschätzung erhält man $\log(n+1) = \int_1^{n+1} \frac{1}{x}\,dx < H_n < 1 + \int_1^n \frac{1}{x}\,dx = 1 + \log(n)$. Also ist die erwartete Abfragezeit $O(\log n)$.

Nun zum **Speicherbedarf** für D. Dazu genügt es die Anzahl der Knoten abzuschätzen. Die Anzahl der Blätter ist jene der Trapeze in T, also $O(n)$ nach Lemma 6.2. Somit ist die Gesamtanzahl der Knoten:

$$O(n) + \sum_{i=1}^{n} (\text{Anzahl der im i. Schritt eingefügten Knoten})$$

Sei k_i die Zufallsvariable für die Anzahl der Trapeze (also der Blätter von D) die im i. Schritt durch Einfügen des Segments s_i erzeugt werden. Damit ist die Anzahl der im i. Schritt eingefügten Knoten $k_i - 1$. Selbst im schlimmsten Fall kann k_i nicht größer als die Gesamtanzahl der Trapeze in $T(S_i)$ also $O(i)$ sein. D.h. im schlimmsten Fall ist der Speicherbedarf

$$O(n) + \sum_{i=1}^{n} O(i) = O(n^2).$$

Für den erwarteten Speicherbedarf erhalten wir

$$O(n) + E\Big(\sum_{i=1}^{n} (k_i - 1)\Big) = O(n) + \sum_{i=1}^{n} E(k_i).$$

Für ein Trapez $f \in T(S_i)$ und ein Segment $s \in S_i$ sei

$$\delta(f, s) := \begin{cases} 1 & \text{falls } f \text{ aus } T \text{ verschwindet, wenn } s \text{ aus } S_i \text{ entfernt wird} \\ 0 & \text{sonst.} \end{cases}$$

Wir habe oben gesehen, daß höchstens 4 Segmente das Verschwinden von f verursachen können. Folglich ist

$$\sum_{s\in S_i}\sum_{f\in T(S_i)}\delta(f,s)=\sum_{f\in T(S_i)}\sum_{s\in S_i}\delta(f,s)\le\sum_{f\in T(S_i)}4=4|T(S_i)|=O(i).$$

Somit ist der Erwartungswert für $k_i=\sum_{f\in T(S_i)}\delta(f,s)$ folgendes Mittel über die zufällig gewählten Elemente $s\in S_i$

$$E(k_i)=\frac{1}{i}\sum_{s\in S_i}\sum_{f\in T(S_i)}\delta(f,s)\le\frac{O(i)}{i}=O(1)$$

Also ist die erwartete Anzahl von im Schritt i neu erzeugten Trapezen $O(1)$ und folglich $O(n)$ der erwartetet Speicherbedarf.

Erwartete Aufbauzeit:

Die Zeit für das Einfügen von s_i ist die Summe aus $O(k_i)$ und der Zeit um den linken Endpunkt von s_i in $T(S_{i-1})$ zu suchen. Somit ist die erwartete Laufzeit

$$O(1)+\sum_{i=1}^{n}\Big(O(\log i)+O(E(k_i))\Big)=O(n\log n).\quad\square$$

Erwartungswerte bzgl. der Suchpunkte, [dBvKOS97, S.142]

Beachte, daß für **jeden** Input S der Größe n die erwartete Laufzeit $O(n\log n)$ ist. Theorem 6.3 besagt allerdings nichts über die erwartete maximale Abfragezeit bzgl. aller möglichen Abfragepunkte. Im Buch wird fälschlicher Weise behauptet, daß im übernächsten Abschnitt gezeigt wird, daß diese ebenfalls $O(\log n)$ ist. Wir werden zeigen, daß wir eine Datenstruktur der Größe $O(n)$finden können die maximale Suchzeit $O(\log n)$ für beliebige Punkte ermöglicht, siehe 6.8.

Punktpositionen für Unterteilungen der Ebene, [dBvKOS97, S.143]

Sei nun S eine Unterteilung der Ebene. Wir setzen voraus, daß S als doppeltverbundene Kantenliste mit n-Kanten gegeben ist. Mittels TrapezoidalMap erhalten wir eine Suchstruktur D für die trapezoidale Verfeinerung $T(S)$ von S. Um diese für die Suche in S zu verwenden, müssen wir jedem Blatt in D einen Pointer auf jene Seite f von S, welche das entsprechende Trapez enthält, zuordnen. Nach Kapitel 2 enthält die doppelt-verbundene Liste von S für jede orientierte Kante einen Pointer auf die Inzidenz-Fläche. Wir müssen also nur für jedes Trapez f von $T(S)$ die unterhalb liegende Inzidenzfläche von top(f) nehmen. Falls top(f) die obere Kante von R ist, so ist f in der unbeschränkten Fläche von S.

Im nächsten Abschnitt werden wir zeigen, daß die Voraussetzung über die allgemeine Lage fallen gelassen werden kann. Die entsprechende Version von Theorem 6.3 liefert dann:

6.4 Folgerung, [dBvKOS97, S.143]**.**
Sei S eine Zerlegung der Ebene durch n Geradensegmente. Man kann in erwarteter $O(n\log n)$ Zeit eine Datenstruktur mit erwarteten $O(n)$ Speicherbedarf so konstruieren, daß die erwartete Laufzeit für Punktabfragen $O(\log n)$ ist.

Tag 5.12

Behandlung der degenerierten Fälle

Endpunkte in allgemeiner Lage, [dBvKOS97, S.143]

Diese Voraussetzung, daß keine zwei Endpunkte die gleiche x-Koordinate haben, kann durch eine kleine Drehung erreicht werden. Das macht allerdings numerische Schwierigkeiten: Wenn z.B. die Koordinaten ganzzahlig sind, dann können wir das bei der Drehung nicht beibehalten. Darum verwenden wir eine symbolische Variante: In Kapitel 5 haben wir die Bildung von zusammengesetzten Zahlen als eine symbolische Transformation kennengelernt. Hier verwenden wir eine Scherung längs der x-Achse $\varphi : (x, y) \mapsto (x+\varepsilon y, y)$ mit $\varepsilon > 0$. Dies transformiert Punkte mit gleicher x-Koordinate in solche mit verschiedenen. Bei kleinen ε wird die Ordnung der x-Koordinaten nicht zerstört (wähle $\varepsilon \cdot (\max y - \min y) < \min\{x' - x : x' > x\}$, dann ist $x + \varepsilon y < x' + \varepsilon y'$). Wir wollen also TrapezoidalMap auf $\varphi(S) := \{\varphi(s) : s \in S\}$ anwenden. Wegen der numerischen Probleme speichern wir dazu aber den Punkt $\varphi(x, y) := (x + \varepsilon y, y)$ als (x, y). Glücklicherweise werden in dem Algorithmus keine Rechnungen mit Punkten durchgeführt (insbesonders haben wir nicht die Endpunkte der vertikalen Erweiterungen berechnet) sondern nur folgende Operationen durchgeführt:

Übersetzen der Abfragen, [dBvKOS97, S.145]

1. Feststellen, auf welcher Seite der vertikalen Geraden durch einen Punkt p ein anderer Punkt q liegt.
2. Feststellen, auf welcher Seite eines durch seine zwei Endpunkte p_1 und p_2 gegebenen Segments ein anderer Punkt q liegt. Dies war nur für solche Punkte durchzuführen, für welche die vertikale Gerade durch sie das Segment trifft.

Beim Aufbau der Datenstrukturen sind sowohl p, p_1 und p_2 Endpunkte von Segmenten in S, als auch q! Wie übersetzen sich diese Operationen?

1. Falls $p_x \neq q_x$, so ist $\varphi(p)_x < \varphi(q)_x \Leftrightarrow p_x < q_x$, wegen der Wahl von ε. Andernfalls ist $\varphi(p)_x < \varphi(q)_x \Leftrightarrow p_y < q_y$. Somit liegt $\varphi(q)$ in beiden Fällen nicht auf der vertikalen Gerade durch $\varphi(p)$.
2. Sei $p_i = (x_i, y_i)$ und $q = (x, y)$. Da die vertikale Gerade durch $\varphi(q)$ das Segment $\varphi(s)$ treffen soll, ist $x_1 + \varepsilon y_1 \leq x + \varepsilon y \leq x_2 + \varepsilon y_2$. Also ist $x_1 \leq x \leq x_2$ und wenn $x_1 = x$ ist, so muß $y_1 < y$ sein und analog für $x = x_2$. Falls $x_1 = x_2$ gilt, so ist das untransformierte Segment s vertikal, also $x_1 = x = x_2$ und somit $y_1 \leq y \leq y_2$, d.h. $q \in s$ und somit auch $\varphi(q) \in \varphi(s)$. Andernfalls ist $x_1 < x_2$ und damit offensichtlich $\varphi(p)$ oberhalb $\varphi(s)$ genau dann, wenn p oberhalb s liegt.

Symbolische Scherung, [dBvKOS97, S.145]

Wenn wir also den Algorithmus für $\varphi(S)$ anstelle von S durchführen, so ist die einzige Änderung, daß wir Punkte lexikographisch anstatt nach der x-Koordinate allein vergleichen müssen. Damit erhalten wir die trapezoidale Verfeinerung von $\varphi(S)$ und eine Suchstruktur hierfür. Der für ε gewählte Wert hat dabei keine Rolle gespielt.

Abfragepunkt in allgemeiner Lage, [dBvKOS97, S.145]

Die Lösung dafür, daß der Abfragepunkt auf keiner vertikalen Geraden durch einen Endpunkt und auf keinen Segment liegen soll, ist die selbe wie zuvor: Da die Suchstruktur für die trapezoidale Verfeinerung von $\varphi(S)$ konstruiert wurde, müssen wir auch die Suchabfrage für den transformierten Punkt $\varphi(q)$ durchführen: Bei x-Knoten müssen wir nun lexikographisch vergleichen. Die einzige Möglichkeit, daß der Suchpunkt auf der vertikalen Geraden bei einem x-Knoten liegt, ist, daß er mit den im Knoten gespeicherten Endpunkt übereinstimmt, und dann ist unsere Suche beendet. Bei y-Knoten wird die Lage des transformierten Punktes $\varphi(q)$ bzgl. $\varphi(s)$ getestet. Mit den möglichen Ergebnissen "oberhalb", "unterhalb" und "darauf". In den ersten beiden Fällen ist festgelegt, wie wir weiter gehen. Im letzten Fall liegt der Punkt auf dem entsprechenden Segment und unsere Suche ist wieder beendet.

Zusammenfassend gilt somit:

6.5 Theorem [dBvKOS97, S.146]**.**
Der Algorithmus TrapezoidalMap *(via lexikographischer Ordnung) liefert die trapezoidale Verfeinerung einer Menge von n nicht-kreuzenden Geradensegmenten und eine Suchstruktur hierfür in erwarteter $O(n \log n)$ Zeit. Der erwartete Speicherbedarf ist $O(n)$ und die erwartete Abfragezeit für Punkte ist $O(\log n)$.* □

Wahrscheinlichkeit für die maximale Abfragezeit

6.6 Lemma, [dBvKOS97, S.146]**.**
Sei S eine Menge von n nicht-kreuzenden Geradensegmenten, q ein Abfragepunkt und $\lambda > 0$. Die Wahrscheinlichkeit, daß der Suchpfad für q in der durch TrapezoidalMap(S) *berechneten Suchstruktur mindestens $3\lambda \log(n+1)$ viele Knoten durchläuft ist höchstens $1/(n+1)^{\lambda \log(5/4)-1}$.*

Beweis. Wir würden gerne für $1 \leq i \leq n$ die Zufallsvariable X_i verwenden, welche 1 ist, wenn mindestens ein Knoten am Suchpfad von q in der i. Iteration erzeugt wurde, und 0 sonst. Wir benötigen für diesen Beweis die Unabhängigkeit der Variablen, was aber für X_i nicht gegeben ist. Darum definieren wir einen gerichteten azyklischen Graphen G mit einer einzigen Quelle und einer einzigen Senke. Die Pfade von der Quelle zur Senke entsprechen den Permutationen von S in folgender Weise: Die Knoten sind die Teilmengen von S. Für jede Teilmenge $S' \subset S$ und $s \in S \setminus S'$ definieren wir eine orientierte Kante $S' \to S' \cup \{s\}$. Jedes $S' \subseteq S$ hat somit genau $|S'|$ eingehende und $n - |S'|$ ausgehende Kanten, siehe [dBvKOS97, S.147]. Pfade von $\emptyset$ to S beschreiben Reihenfolgen der Elemente in S, also Permutationen. Wir markieren jene Kanten $s : S' \to S' \cup \{s\}$, die eine Änderung des q-enthaltenden Trapezes bei Einfügen von s in $T(S')$ bewirken. Wir sind an der Anzahl der markierten Kanten interessiert. Dazu verwenden wir wieder Rückwärts-Analysis wie im Beweis von Theorem 6.3. Es gibt höchstens 4 Segmente s, die das Trapez bei Übergang von $S' \setminus \{s\} \to S'$ ändern. Also sind höchstens 4 der hereinkommenden Kanten bei jedem Knoten markiert. Falls es weniger als 4 sind, so markieren wir entsprechend viele zusätzlich. Bei Knoten der Kardinalität kleiner als 4 geht das nicht und wir markieren dort alle eingehenden. Sei nun die Zufallsvariable X_i abhängig von den Pfaden maximaler Länge (also den Permutationen) wie folgt definiert:

$$X_i := \begin{cases} 1 & \text{falls die } i\text{. Kante des Pfades markiert ist} \\ 0 & \text{sonst} \end{cases}$$

Da jeder Knoten S' mit Kardinalität i genau i hereinkommende Kanten besitzt und davon genau 4 (falls $i \geq 4$) markiert sind, ist

$$P(X_i = 1) = \begin{cases} 4/i & \text{für } i \geq 4 \\ 1 \leq 4/i & \text{sonst} \end{cases}$$

Beachte, daß die so definierten X_i nun unabhängig sind, denn, da die Anzahl der eingehenden markierten Kanten bei jedem Knoten der Tiefe i gleich ist, ist $P(X_i = 1)$ unabhängig davon, ob die j-te Kante (mit $j > i$) von σ markiert ist. Sei $Y := \sum_{i=1}^{n} X_i$. Die Anzahl der Knoten am Suchpfad ist höchstens $3Y$.

Wir verwenden die Markow-Ungleichung: Für jedes $\alpha > 0$ und Zufallsvariable $Z \geq 0$ ist $P(Z \geq \alpha) \leq E(Z)/\alpha$, denn sei χ_B die charakteristische Funktion von B, dann ist

$$\alpha\, P(Z \geq \alpha) = \alpha\, E(\chi_{Z^{-1}[\alpha,+\infty)}) = E(\alpha \cdot \chi_{Z^{-1}[\alpha,+\infty)}) \leq E(Z). \quad \square$$

Somit gilt für alle $t > 0$:

$$P\Big(Y \geq \lambda \log(n+1)\Big) = P\Big(e^{tY} \geq e^{t\,\lambda\,\log(n+1)}\Big) \leq e^{-t\,\lambda\,\log(n+1)}\, E(e^{tY}).$$

Für unabhängige Zufallsvariablen Z_i (d.h. $P(\bigcap_i A_i) = \prod_i P(A_i)$, bzw. $P(Z_1 = a_1, Z_2 = a_2) = P(Z_1 = a_1) \cdot P(Z_2 = a_2)$) ist $E(\prod_i Z_i) = \prod_i E(Z_i)$ und somit

$$E(e^{tY}) = E\Big(e^{\sum_i tX_i}\Big) = E\Big(\prod_i e^{tX_i}\Big) = \prod_i E(e^{tX_i})$$

Für $t := \log(5/4)$ erhalten wir:

$$\begin{aligned} E(e^{tX_i}) &= e^{t\cdot 1} P(X_i = 1) + e^{t\cdot 0} P(X_i = 0) \\ &= \frac{5}{4}\, P(X_i = 1) + 1 - P(X_i = 1) = \frac{1}{4}\, P(X_i = 1) + 1 \leq \frac{i+1}{i} \end{aligned}$$

und somit

$$E(e^{tY}) = \prod_i E(e^{tX_i}) \leq \prod_{i=1}^{n} \frac{i+1}{i} = n+1$$

und schließlich

$$\begin{aligned} P\Big(Y \geq \lambda \log(n+1)\Big) &\leq e^{-\lambda\, t\, \log(n+1)} E(e^{tY}) \\ &\leq e^{-\lambda\, t\, \log(n+1)}(n+1) = \frac{n+1}{(n+1)^{\lambda t}} = \frac{1}{(n+1)^{\lambda t - 1}}. \quad \square \end{aligned}$$

Tag 5.17

6.7 Lemma, [dBvKOS97, S.148]**.**
Sei S eine Menge von n nicht-kreuzenden Geradensegmenten und $\lambda > 0$. Die Wahrscheinlichkeit, daß mindestens ein Suchpfad in der durch TrapezoidalMap(S) *berechneten Suchstruktur mindestens $3\lambda \log(n+1)$ viele Knoten durchläuft ist höchstens $2/(n+1)^{\lambda \log(5/4) - 3}$.*

Beweis. Wir nennen zwei Abfragepunkte äquivalent, wenn sie den gleichen Suchpfad in der Suchstruktur D haben. Wenn wir die Ebene in vertikale Streifen durch Gerade in den Endpunkten der Segmente zerlegen und diese seinerseits durch die Segmente in Trapeze zerlegen, so erhalten wir eine Partition der Ebene in höchstens $2(n+1)^2$ viele Trapeze. Punkte, die im selben Trapez liegen, sind dann in jeder Suchstruktur äquivalent, denn die Abfragen sind ja nur nach der Lage bzgl. vertikaler Geraden durch Endpunkte und bzgl. der Segmente. Um eine obere Schranke für die maximale Länge der Suchpfade zu erhalten, genügt es also in jedem dieser Trapeze einen Punkt zu betrachten. Nach Lemma 6.6 ist die Wahrscheinlichkeit dafür, daß die Länge des Suchpfads für einen fixen Punkt $3\lambda\log(n+1)$ überschreitet, höchstens $1/(n+1)^{\lambda\log(5/4)-1}$. Somit ist die Wahrscheinlichkeit, daß die Länge des Suchpfads mindestens eines der $2(n+1)^2$ Testpunkte den Wert $3\lambda\log(n+1)$ überschreitet, höchstens $2(n+1)^2/(n+1)^{\lambda\log(5/4)-1} = 2/(n+1)^{\lambda\log(5/4)-3}$. □

Wahrscheinlichkeit für guten Algorithmus, [dBvKOS97, S.149]

Laut Buch impliziert Lemma 6.7, daß die erwartete maximale Abfragezeit $O(\log n)$ ist, beachte jedoch folgendes: Lemma 6.6 besagt, daß die Wahrscheinlichkeit dafür, daß die Länge des Suchpfades für ein $q \in \mathbb{R}^2$ in der Datenstruktur D_S mindestens $3\lambda\log(n+1)$ ist, höchstens $1/(n+1)^{\lambda\log(5/4)-1}$ ist, wobei $n := |S|$. Lemma 6.7 besagt, daß die Wahrscheinlichkeit dafür, daß der Suchpfad mindestens eines Punktes $q \in \mathbb{R}^2$ in der Datenstruktur D_S mindestens $3\lambda\log(n+1)$ ist, höchstens $1/(n+1)^{\lambda\log(5/4)-3}$ ist. Sei also M_S die Zufallsvariable, die zu jeder Permutation σ von S die maximale Suchpfadlänge in $D_S(\sigma)$ beschreibt, dann besagt 6.7: $P(M_S \geq 3\lambda\log(n+1)) \leq 1/(n+1)^{\lambda\log(5/4)-3}$ für jedes $\lambda > 0$, also $P(M_S \geq \ell) \leq (n+1)^3/\sqrt[3]{5/4}^{\ell}$ für jedes $\ell \in \mathbb{N}$ (wobei wir $\lambda := \ell/(3\log(n+1))$ gesetzt haben). Der Erwartungswert für die maximale Suchpfadlänge $M_S : \sigma \mapsto M_S(\sigma)$ läßt sich also wie folgt abschätzen

$$E(M_S) = \sum_{\ell=1}^{\infty} P(M_S \geq \ell) \leq \sum_{\ell=1}^{\infty} \frac{(n+1)^3}{(\sqrt[3]{5/4})^{\ell}} = (n+1)^3 \frac{1}{\sqrt[3]{5/4}-1}$$

und somit **folgt nicht** $E(M_S) = O(\log n)$.

Konstruktion einer guten Suchstruktur, [dBvKOS97, S.149]

Wählt man allerdings z.B. $\lambda := 20$, dann ist die Wahrscheinlichkeit, daß die maximale Länge eines Suchpfades mehr als $3\cdot 20\cdot\log(n+1)$ ist höchstens $2/(n+1)^{20\log(5/4)-3} < 2/(n+1)^{1.4628} < \frac{1}{4}$ für $n > 3$ (für $\lambda \to \infty$ geht die entsprechende Wahrscheinlichkeit sogar gegen 0). Also ist die Wahrscheinlichkeit für eine gute ($= O(\log n)$) Abfragezeit mindestens 3/4. Nach 6.3 ist der Erwartungswert $E(|D|)$ für die Größe der Suchstruktur ein $O(n)$, also $E(|D|) \leq c\,n$, und somit nach der Ungleichung von Markow auch $P(|D| \geq 4\,c\,n) \leq E(|D|)/(4\,c\,n) \leq 1/4$. Wie im Beweis von 6.3 gezeigt, ist die Aufbauzeit $O(n\log n)$, falls die maximale Länge eines Suchpfades $O(\log n)$ und der Speicherbedarf $O(n)$ ist. Also ist die Wahrscheinlichkeit, daß die Abfragezeit, die Größe und die Aufbauzeit gut sind mindestens $1-(1/4+1/4) = 1/2$. Um nun eine gute Suchstruktur zu erhalten, rufen wir TrapezoidalMap(S) auf und verfolgen die Größe und die maximale Länge der Suchpfade der Suchstruktur. Sobald der Speicherbedarf $c_1\,n$ oder die Suchtiefe $c_2\,\log n$ überschreitet (mit geeignet fixierten Konstanten c_1 und c_2), starten wir den Algorithmus erneut mit einer Zufallspermutation von S. Da die Wahrscheinlichkeit, daß eine Permutation eine Datenstruktur der gewünschten

Größe und Suchtiefe liefert mindestens 1/2 (im Buch 1/4) ist, erwarten wir das Ziel in 2 Versuchen zu erreichen:

$$
\begin{aligned}
E(\text{Anz. der nötigen Permutat.}) &= \sum_{k=1}^{\infty} k \left(1-\frac{1}{2}\right)^{k-1} \frac{1}{2} = \sum_{k=1}^{\infty} k/2^k = 2 \\
\text{oder allgemeiner} \quad &= \sum_{k=1}^{\infty} k\,(1-\mu)^{k-1}\,\mu = \frac{\mu}{\Big(1-(1-\mu)\Big)^2} = \frac{1}{\mu} \\
\text{für } 0<\mu<1, \text{ wegen} \quad S &:= \sum_{k=1}^{\infty} k\,q^{k-1} = \sum_{k=0}^{\infty} k\,q^{k-1} \text{ mit } q := 1-\mu \\
q\,S &= \sum_{k=0}^{\infty} k\,q^k = \sum_{j=1}^{\infty} (j-1)\,q^{j-1} \\
(1-q)\,S = S - q\,S &= \sum_{k=1}^{\infty} \Big(k-(k-1)\Big)\,q^{k-1} = \sum_{j=0}^{\infty} q^j = \frac{1}{1-q}
\end{aligned}
$$

Da wir spätestens nach der Hälfte der möglichen Permutationen eine gute Suchstruktur erhalten, ergibt sich:

6.8 Theorem, [dBvKOS97, S.149]**.**
Sei S eine Zerlegung der Ebene mittels n Kanten. Dann existiert eine Suchstruktur für Punktabfragen, die maximal $O(n)$ Speicherbedarf und maximal $O(\log n)$ Suchzeit hat.

Vorbereitungszeit, [dBvKOS97, S.149]

Das Theorem sagt nichts über die Vorbereitungszeit. Allein um die maximale Länge eines Suchpfades zu bestimmen, mußten wir $2(n+1)^2$ viele Testpunkte in 6.7 betrachten. Laut Buch kann man diese Anzahl auf $O(n \log n)$ drücken und damit eine erwartete Vorbereitungszeit von $O(n(\log n)^2)$ erreichen.

Nachbemerkungen

[dBvKOS97, S.149]

In [PS85] findet sich eine Übersicht, über die Anfänge des Punktlokalisierungsproblems. Es wurde 4 grundsätzlich verschiedene Methoden, die alle die optimale Suchzeit $O(\log n)$ und optimalen Speicherbedarf $O(n)$ liefern, gefunden: Ketten-Methode von [EGS86] (basierend auf fractional cascading), Triangulierungs-Verfeinerung von [Kir83], Persistence (Ausdauer) von [ST86] und [Col86] und randomizierte inkrementelle Methode von [Mul90]. Wir haben letztere wie in [Sei91a] behandelt.

Im mehr als 2-dimensionalen ist das Punktabfrageproblem im wesentlichen offen, und nur für spezielle Zerlegungen bekannt.

Für Punktlokalisation in konvexen Polyeder im $\mathbb{R}^d$ existiert eine $O(n)$ große Datenstruktur die Suchabfragen in $O(n^{1-1/\lfloor d/2 \rfloor}(\log n)^{O(1)})$ Zeit erlaubt, siehe [Mat92].

7. Voronoi Diagramme (Postamtproblem)

Aufgabenstellung [dBvKOS97, S.153]

Stelle den Einzugsbereich für vorgegebene Standorte von Geschäften oder Dienstleistungen, z.B. Postämtern. Wir machen folgende vereinfachende nicht sehr realistische Annahmen:

- Der Preis der Güter bzw. Dienstleistungen ist an allen Standorten gleich.
- Die Kosten für deren Erwerb ist gleich dem Preis plus den Transportkosten.
- Die Transportkosten sind proportional zur euklidischen Distanz.
- Die Konsumenten versuchen die Kosten zu minimieren.

Unter diesen Annahmen ergibt sich eine Zerlegung der Ebene in die Verkaufsgebiete, wo all jene Konsumenten wohnen, die den gleichen Standort bevorzugen. Diese Zerlegung heißt Voronoi Diagramm. Ein Beispiel für die Information, die wir aus diesen Diagramm entnehmen können ist, daß Standorte deren Einzugsgebiete einen gemeinsamen Rand haben, für die Leute die bei diesen Wohnen in direkter Konkurrenz stehen. Die Anwendungen der Voronoi Diagramme liegen keinesfalls nur in der Wirtschaftsgeographie sondern auch in der Physik, Astronomie, Robotik, und vielen anderen Gebieten. Diese Diagramme stehen in enger Beziehung zu den sogenannten Delaunay Triangulierungen, siehe Kapitel 9.

5.1 Mathematisches Formulierung, [dBvKOS97, S.148]

Es bezeichne $d(p,q) := \sqrt{\sum_i (p_i - q_i)^2}$ die euklidische Distanz zwischen p und q. Sei $P := \{p_1, \ldots, p_n\}$ die Menge der paarweise verschiedenen Standorte in $\mathbb{R}^2$. Das Voronoi-Diagramm $\mathrm{Vor}(P)$ von P ist die Unterteilung der Ebene in Bereiche (sogenannte Voronoi-Zellen) $V(p)$ für jeden Standort $p \in P$, welche gegeben sind durch

$$V(p) := \{x \in \mathbb{R}^2 : d(x,p) < d(x,q) \forall q \in P \setminus \{p\}\}.$$

Wir nennen das Voronoi-Diagramm zusammenhängend, wenn das Komplement aller Voronoi-Zellen (die Kanten und Ecken des Diagramms) zusammenhängend ist. Für Punkte $p \neq q$ bezeichnen wir die offene Halbebene, die durch die Streckensymmetrale von p und q begrenzt wird und p enthält, mit $h(p,q) := \{x \in \mathbb{R}^2 : d(x,p) < d(x,q)\}$. Damit ist $V(p) := \bigcap_{q \in P \setminus \{p\}} h(p,q)$, also ein konvexes (möglicherweise unbeschränktes) Gebiet, welches durch höchstens $n-1$ viele Kanten und Ecken begrenzt wird.

Tag 5.19

7.2 Theorem [dBvKOS97, S.155]**.**
Sei P eine n-elementige Teilmenge von $\mathbb{R}^2$. Falls P in einer Geraden enthalten ist, dann besteht $\mathrm{Vor}(P)$ *aus $n-1$ parallelen Kanten. Andernfalls ist* $\mathrm{Vor}(P)$ *zusammenhängend und die Kanten sind Geradensegmente oder Halbgeraden.*

Beweis. Wegen $V(p) = \bigcap_{q \neq p} h(p,q)$ ist der Rand von $V(p)$ enthalten in der Vereinigung der entsprechenden Streckensymmetralen $\partial h(p,q)$ (wegen $\partial \circ \cap \subseteq \cup \circ \partial$). Falls $P \subseteq \ell$ für eine Gerade ℓ, so stehen die Streckensymmetralen normal auf ℓ und das Komplement von $\bigcup_p V(p)$ ist die Vereinigung einiger dieser Symmetralen. Andernfalls

besteht das Komplement jedenfalls aus möglicherweise unbeschränkten Segmenten der Streckensymmetralen. Angenommen eine ganze Streckensymmetrale $e = \partial h(p_1, p_2)$ von $p_1, p_2 \in P$ wäre Randkante der Zelle $V(p_1)$. Sei p ein Punkt aus P der nicht auf der Geraden durch p_1 und p_2 liegt. Dann ist die Streckensymmetrale $\partial h(p, p_1)$ nicht parallel zu e und schneidet somit diese. Damit ist jener Halbstrahl von e, der in $h(p, p_1)$ liegt, nicht im Rand von $V(p_1)$. (Achtung: Ecken P eines Rechtecks!) Angenommen $\mathrm{Vor}(P)$ wäre nicht zusammenhängend. Wähle zwei Punkte in verschiedenen Komponenten mit minimalen Abstand. Wegen der Konvexität der Zellen müßten die Kanten durch diese Punkte parallele Geraden sein, die eine Zelle begrenzen, was wir eben als unmöglich erkannt haben. □

Komplexität des Voronoi-Diagramms [dBvKOS97, S.156]

Da jede der n Voronoi-Zellen höchstens $n-1$ Ecken und Kanten hat ist die Komplexität von $\mathrm{Vor}(P)$ höchstens quadratisch in n. Einzelne Voronoi-Zellen können wirklich lineare Komplexität haben, wie das Beispiel [dBvKOS97, S.156] zeigt. Viele Zellen können das aber nicht sein:

7.3 Theorem [dBvKOS97, S.156]**.**
Für Voronoi-Diagramme von $P \subseteq \mathbb{R}^2$ mit $n := |P| \geq 3$ gilt: Die Anzahl der Ecken ist höchstens $2n - 5$ und jene der Kanten höchstens $3n - 6$.

Beweis. Falls P in einer Geraden enthalten ist, so ist dies nach 5.2 klar. Andernfalls wollen wir die Eulersche Formel anwenden. Dazu fügen wir einen Punkt p_∞ hinzu und verbinden alle Halbstrahlen von $\mathrm{Vor}(P)$ mit p_∞ und erhalten einen ebenen Graphen. Sei n_v die Anzahl der Ecken von $\mathrm{Vor}(P)$ und n_e jene der Kanten. Dann ist $(n_v + 1) - n_e + n = 2$ nach der Eulerschen Formel. Die Summe aller Indizes der Ecken ist das Doppelte der Anzahl der Kanten, denn jede Kante endet in genau zwei Ecken. Jeder Index ist mindestens 3 (auch bei p_∞), denn die Ecken von $V(P)$ sind Schnittpunkte mindestens zweier Streckensymmetralen $\partial h(p_0, p_i)$ mit $i \in \{1, 2\}$ also auch von $\partial h(p_1, p_2)$. Somit ist $2n_e \geq 3(n_v + 1) = 3(2 - n + n_e)$, also $3(n - 2) \geq n_e$ und $3\,n_v \leq 2n_e - 3 \leq 3(2(n - 2) - 1)$. □

Kreis $C(q)$ zum nächsten Standort, [dBvKOS97, S.156]

Die Anzahl der Streckensymmetralen ist quadratisch in n, jene der Ecken und der Kanten von $\mathrm{Vor}(P)$ aber nur linear, und somit enthalten nicht alle Symmetralen Kanten und nicht alle Schnitte von Paaren von Symmetralen sind Ecken. Um jene zu charakterisieren, die dies dennoch tuen, bezeichnen wir mit $C(q)$ den Kreis mit Mittelpunkt q und Radius $d(q, P) := \min\{d(q, p) : p \in P\}$.

7.4 Theorem [dBvKOS97, S.157]**.**
Sei $P \subseteq \mathbb{R}^2$ endlich, $n := |P|$ und $q \in \mathbb{R}^2$. Dann gilt:

1. *Falls $C(q) \cap P = \{p_1\}$, so ist $q \in V(p_1)$.*
2. *Falls $C(q) \cap P = \{p_1, p_2\}$ mit $p_1 \neq p_2$, so liegt q auf einer Kante $e \subseteq \partial h(p_1, p_2)$ von $\mathrm{Vor}(P)$.*
3. *Falls $|C(q) \cap P| \geq 3$, so ist q eine Ecke von $\mathrm{Vor}(P)$.*

Beweis. 1 Sei $C(q) \cap P = \{p_1\}$, d.h. $d(q,p_1) < d(q,p)$ für alle $p \in P \setminus \{p_1\}$. Damit ist $q \in V(p_1)$.

2 Sei $C(q) \cap P = \{p_1, p_2\}$ mit $p_1 \neq p_2$. Dann ist $d(q,p_i) = d(q,P) \leq d(q,p)$ für $i \leq 2$ und alle $p \in P$. Somit ist $q \in \partial h(p_1,p_2) \setminus \bigcup_{p\in P} V(p)$. Also liegt q in $\mathrm{Vor}(P)$. Angenommen q läge nicht auf einer Kante e von $\mathrm{Vor}(P)$ und wäre somit eine Ecke und damit ein Randpunkt zweier aufeinanderfolgender Kanten einer Zelle $V(p)$, läge also auf $\partial h(p,p_\pm)$ für zwei Punkte $p_\pm \in P$. Also wäre q von p, p_+ und p_- gleich weit entfernt und von keinen anderen p' weniger weit entfernt, denn andernfalls wäre q kein Randpunkt der $V(p)$. Somit wäre $\{p_-, p, p_+\} \subseteq C(q)$, ein Widerspruch.

3 Seien p_1, p_2, p_3 aufeinanderfolgende Punkte von $C(q) \cap P$. Dann ist $d(q,p_i) = d(q,P) \leq d(q,p)$ für $i \leq 3$ und $p \in P$, d.h. $q \in \partial V(p_i)$ für $i \in \{1,2,3\}$ und somit eine Ecke von $V(p_2)$. □

Berechnen des Voronoi-Diagramms [dBvKOS97, S.157]

Naive Methode [dBvKOS97, S.157]

Mittels des Algorithmuses IntersectHalfPlanes können wir für $p \in P$ den Durchschnitt der Halbebenen $h(p,p')$ mit $p' \neq p \in P$ nach 4.4 in $O(n \log n)$ Zeit bestimmen. Wir benötigen also auf diese Weise $O(n^2 \log n)$ Zeit zur Bestimmung von $\mathrm{Vor}(P)$. Wir wollen dies nun mittels eine Plane-Sweep Verfahren (den nach Fortune benannten Algorithmus) in $O(n \log n)$ Zeit erreichen. Da Sortieren von n Zahlen durch Bestimmen eines zugehörigen Voronoi-Diagramms erreicht werden kann (siehe Aufgabe 7.3 in [dBvKOS97, S.168]), ist (trotz linearer Komplexität von $\mathrm{Vor}(P)$) die Laufzeit für die Bestimmung von $\mathrm{Vor}(P)$ nicht auf $O(n)$ drückbar.

Punkte, wo nächster Standort bekannt, [dBvKOS97, S.158]

Das Problem beim Plane-Sweep ist, daß wenn die Sweep-Line die oberste Ecke einer Zelle $V(p)$ erreicht, so haben wir p noch nicht besucht und können somit diese Ecke nicht bestimmen. Anstelle im Status den Durchschnitt von $\mathrm{Vor}(P)$ mit der Sweepline zu protokollieren, speichern wir Information über jene Zellen $V(p)$ mit $p \in P$ oberhalb der Sweepline $\ell = \{(x,t) : x \in \mathbb{R}\}$, welche sich durch Standorte unterhalb nicht ändern können. Jene Punkte q in der Halbebene ℓ^+ oberhalb ℓ, von denen wir die nächstgelegenen Standort p bereits feststellen können, sind jene die oberhalb einer gleichseitigen Parabel mit Brennpunkt $p \in P \cap \ell^+$ und Leitlinie ℓ liegen. In Formeln ist diese gegeben durch $|(x,y) - (p_x,p_y)| = |y - t|$, also $(x-p_x)^2 = 2(p_y - t) \cdot \left(y - \frac{p_y+t}{2}\right)$. Parabeln mit gleichen Brennpunkt und tieferer Leitlinie liegen unterhalb. Zwei dieser Parabeln mit gleicher Leitlinie schneiden sich in genau zwei Punkten, außer Brennpunkte liegen auf der Leitlinie (dann degeneriert die Parabel zu einen vertikalen Halbstrahl) oder die Brennpunkte liegen auf gleicher Höhe oberhalb der Leitlinie, dann schneiden sich die Parabeln in genau einen Punkt. Der Schnittpunkt, wo von links nach rechts zuerst die Parabel mit Brennpunkt p und danach jene mit Brennpunkt p' die tiefergelegene ist, berechnet sich aus den Schnitt der Streckensymmetrale $\frac{p+p'}{2} + \lambda\,(p'-p)^\perp$ mit einer der beiden Parabeln und zwar jener Schnittpunkt mit dem größeren Parameter λ. Insbesonders können sich nur idente Parabeln berühren.

Tag 5.20

Strandlinie als Status, [dBvKOS97, S.158]

Den aus Parabelbögen bestehenden Rand diese Vereinigung nennen wir Strandlinie. Beachte, daß die (Knick-)Punkte der Strandlinie, d.h. wo zwei ihrer Parabelbögen zusammenkommen, auf abgeschlossenen Kanten des Voronoi-Diagramms liegen (und, wärend die Sweepline von oben nach unten wandert, diese überstreichen): Sei q ein Knickpunkt der Parabelbögen zu p_1 und p_2. Dann ist $C(q)$ der Kreis durch p_1 und p_2, denn diese (und die Sweepline) sind gleich weit entfernt und jedes weitere $p \in P$ ist mindestens soweit entfernt, denn andernfalls läge q oberhalb der Parabel zu p. Nach 7.4 ist somit q entweder eine Ecke oder liegt im Inneren einer Kante von $\mathrm{Vor}(P)$. Der Status sollte also aus einer Beschreibung der Strandline bestehen.

Standort-Ereignisse, [dBvKOS97, S.159]

Ein neuer Parabelbogen taucht auf der Strandlinie auf, wenn die Sweepline eine neuen Standort p erreicht. Die Parabel entsteht aus dem vertikalen Halbstrahl durch den Standort, der die Strandlinie in mindestens einem Parabelbogen, sagen wir zu $p' \in P$, trifft. Dabei entstehende Knickpunkte liegen auf der gemeinsamen Kante von $V(p)$ und $V(p')$.

7.6 Lemma. Auftauchen neuer Bögen, [dBvKOS97, S.159]**.**
Die einzige Möglichkeit, daß ein neuer Parabelbogen auf der Strandlinie auftaucht, ist auf Grund des Erreichens eines Standorts, wie eben beschrieben.

Beweis. Angenommen die Parabel eines von der Sweepline bereits besuchten Standorts bricht durch die Strandlinie. Falls der Durchstoßpunkt im Inneren eines Parabelbogens liegt, so sind die beiden Parabeln dort tangential und damit ihre Brennpunkte ident, ein Widerspruch. Andernfalls trifft der Parabelbogen zu p in einem Knickpunkt q zweier Parabelbögen mit Brennpunkten $p_\pm$ auf die Strandlinie. Somit liegen $p_\pm$ und p (in der Reihenfolge p_+, p, p_-) auf dem Kreis durch q der die sweepline berührt. Bewegen wir die sweepline weiter, so würde ein Schnittpunkt $q_\pm$ der Parabel zu $p_\pm$ mit jener zu p auf einem Kreis, mit Mittelpunkt auf der Streckensymmetrale weiter unterhalb und somit größeren Radius und $p_\mp$ im Inneren enthaltend, liegen. Also läge $q_\pm$ oberhalb der Parabel zu $p_\mp$ und somit nicht auf der Strandlinie, siehe [dBvKOS97, S.160]. □

Komplexität der Strandlinie, [dBvKOS97, S.160]

Aus 7.6 folgt, daß die Strandlinie aus höchstens $2n - 1$ Parabelbögen bestehen kann, denn jeder weitere Standort liefert einen neuen Bogen, und schlimmstenfalls die Aufspaltung eines vorhandenen in 2 Teile (zwei dieser Parabeln schneiden sich in zwei Punkten, außer der Brennpunkt ist auf gleicher Höhe), also wird die Anzahl um maximal 2 erhöht.

Kreis-Ereignisse, [dBvKOS97, S.161]

Falls ein Parabelbogen α zwischen zwei Parabelbögen α_- und α_+ verschwindet, dann können α_- und α_+ nicht auf der gleichen Parabel liegen (sonst wäre alle 3 Parabeln ident), haben also verschiedene Standorte $p_\pm$ und p als Brennpunkte. Im Moment,

wo α verschwindet, treffen sich die 3 Parabeln in einem gemeinsamen Punkt q. Nach dem oben gesagten geht $C(q)$ durch alle 3 Brennpunkte, also ist q eine Ecke von $\mathrm{Vor}(P)$ nach 7.4.1. Somit treffen sich zwei Kanten (auf denen die Knickpunkte laufen) in q. Wir nennen den tiefsten Punkt auf einen Kreis durch 3 Standorte, welche aufeinanderfolgende Parabelbögen der Strandlinie beschreiben, ein Kreisereignis.

7.7 Lemma. Verschwinden von Bögen, [dBvKOS97, S.161]**.**
Die einzige Möglichkeit, daß ein Parabelbogen von der Strandlinie verschwindet, ist auf Grund eines Kreis-Ereignisses, wie eben beschrieben. □

Datenstruktur, [dBvKOS97, S.161]

- Das Voronoi-Diagramm wird als doppelt-verbundene Kantenliste gespeichert. Dabei machen die unbeschränkten Segmente Probleme. Deshalb bestimmen wir ein großes Rechteck, welches P ganz im Inneren enthält. Als Endstadium der Zerlegung geben wir dieses Rechteck zusammen mit jenem Teil des Voronoi-Diagramms, welches im Inneren enthalten ist, aus.
- Status: Die Strandlinie speichern wir als balanzierten binären Suchbaum T. Dessen Blätter beschreiben ihre von links nach rechts sortierten Parabelbögen mit Brennpunkten $p \in P$. Zusätzlich speichern wir in jedem Blatt einen Pointer auf den Knoten der Event-Queue, der jenes Kreis-Event beschreibt, daß das Verschwinden dieses Bogens bewirkt. Dieser Pointer ist nil, falls so ein Kreis-Ereignis (noch) nicht gefunden wurde. Die internen Knoten speichern die Knickpunkte als Paare (p, p') benachbarter Bögen sowie Pointer auf jene Kante der doppelt-verbundenen Kantenliste, auf welcher der Knickpunkt des Knotens läuft. Damit benötigen wir $O(\log n)$ Zeit, um Bögen der Strandlinie zu finden, die oberhalb neuer Standorte liegen.
- Die Eventqueue Q enthält die nach y-Koordinate sortierten Events. Für neu erreichte Standorte speichern wir diese. Für Kreis-Events speichern wir den tiefsten Punkt des Kreises zusammen mit einem Pointer auf das, den verschwindenden Bogen repräsentierende, Blatt von T ($\Rightarrow$ Mittelpunkt).

Bestimmung der Kreisereignisse, [dBvKOS97, S.162]

Wir müssen sicherstellen, daß für je 3 aufeinanderfolgende Parabelbögen (die Anlaß für ein Kreisereignis) sein könnten, dieses potentielle Ereignis in Q gespeichert wird. Es kann allerdings passieren, daß die beiden Knickpunkte "divergieren", falls sie sich nämlich auf verschiedenen Symmetralen von deren Schnittpunkt entfernen (dies tritt genau dann ein, wenn die Brennpunkte am Kreis gegen der Uhrzeigersinn angeordnet sind). Selbst wenn sie "konvergieren", so kann trotzdem das zugehörige Kreisereignis ausbleiben, wenn z.B. zuvor ein unterhalb liegender Standort erreicht wird, der dann den Bogen zerteilt. Wir nennen diese Ereignisse einen 'falscher Alarm'. Und wir müssen diese bei Erreichen des unterhalb liegenden Standorts von der Queue entfernen. In jedem Eventpunkt testen wir alle neuen aufeinanderfolgenden Tripel von Parabelbögen. Falls die beiden Knickpunkte sich nähern, so fügen wir das Ereignis zu Q hinzu. Bei Standortevents brauchen wir nur die Fälle, wo der neue Bogen der linke oder der rechte ist untersuchen, falls er nämlich der mittlerer ist, so führt dies zu keinem Kreisereignis, da die beiden Knickpunkte sich entfernen.

Tag 5.26

7.8 Lemma. Identifikation der Ecken, [dBvKOS97, S.163]**.**
Jede Ecke des Voronoi-Diagramms wird durch ein Kreis-Ereignis mit konvergenten Knickpunkten bestimmt.

Beweis. Sei q eine Ecke des Voronoi-Diagramms und $C(q)$ der Kreis aus 7.4. Wir beweisen nur den generischen Fall, daß der tiefste Punkt von $C(q)$ nicht zu P gehört. Sei die Reihenfolge von $P \cap C(q)$ vom tiefsten Punkt aus im Uhrzeigersinn $p_1, p_2, p_3, \ldots$. Wir müssen zeigen, daß (kurz) bevor die Sweepline diesen tiefsten Punkt erreicht drei aufeinanderfolgende Parabelbögen $\alpha_1, \alpha_2, \alpha_3$ zu p_1, p_2, p_3 auf der Strandlinie liegen. Da $C(q)$ keine Standorte im Inneren hat, umfassen etwas kleinere Kreise durch p_1 und p_2 keine anderen Standorte und deren Mittelpunkte sind Knickpunkte. Damit existieren auf der Strandlinie benachbarte Bögen α_1 und α_2 zu p_1 und p_2. Gleiches geht für p_2 und p_3, wobei der Bogen für p_2 mit α_2 übereinstimmt. Somit ist das entsprechende Kreis-Ereignis rechtzeitig in Q und die Ecke q wird damit entdeckt. □

Wenn alle Ereignisse behandelt worden sind und somit Q leer ist, dann existiert die Strandlinie noch immer. Ihre Knickpunkte gehören zu den unbeschränkten (auseinanderlaufenden) Kanten von $\mathrm{Vor}(P)$ und wir müssen ein umfassendes Rechteck verwenden um diese Kanten in einer doppelt-verbundenen Kantenliste zu speichern.

Algorithmus VoronoiDiagramm [dBvKOS97, S.164]

Input: Eine endliche Menge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$.
Output: Das Voronoi-Diagramm $\mathrm{Vor}(P)$ innerhalb eines umfassenden Rechtecks als doppelt-verbundenen Kantenliste

1: Initializiere Q mit den Standort-Ereignissen. $T := \emptyset$. $D := \emptyset$.
2: **while** $Q \neq \emptyset$ **do**
3: Entferne Ereignis q mit größter y-Koordinate aus Q.
4: **if** q ist ein Standortereignis **then**
5: HandleSiteEvent(q)
6: **else** // q ist ein Kreisereignis
7: HandleCircleEvent(γ), wobei γ das den verschwindenden Bogen repräsentierende Blatt in T ist.
8: Die inneren Knoten von T beschreiben die unbeschränkten Kanten von $\mathrm{Vor}(P)$. Bestimme ein Rechteck, welches alle Ecken von $\mathrm{Vor}(P)$ umfaßt. Füge die unbeschränkten Kanten zu D hinzu.
9: Durchlaufe die orientierten Kanten von D und füge Einträge für die Zellen und entsprechende Pointer hinzu.

Subroutine HandleSiteEvent, [dBvKOS97, S.164]

1: **if** $T = \emptyset$ **then**
2: Füge p als einziges Blatt in T ein.
3: **return**
4: Suche in T nach einem Blatt mit Bogen α oberhalb p. Falls α einen Pointer auf ein (notwendigerweise unerledigtes) Kreis-Ereignis (also unterhalb p) in Q besitzt, so ist dieses ein **falscher Alarm** und muß aus Q entfernt werden.

5: Ersetze das **Blatt** für α in T durch einen Teilbaum mit 3 Blättern. Das mittlere Blatt speichert p. Die beiden anderen den Standort p', der ursprünglich in α gespeichert war. Speichere die **Knickpunkte** als (p', p) und (p, p') in zwei neuen inneren Knoten. Rebalanziere T falls notwendig.
6: Erzeuge neue Kanten-Records in D für die **Kante** $e \subseteq \partial h(p, p')$ auf welcher die neuen Knickpunkte laufen.
7: Überprüfe für das Tripel aufeinanderfolgender Bögen, wobei der neue Bogen für p der linke Bogen ist, ob die Knickpunkte zueinander konvergieren. Falls ja, füge ein **neues Kreis-Ereignis** in Q ein sowie Pointer zwischen den entsprechenden Knoten in T und in Q. Verfahre ebenso mit dem neuen Bogen als rechten Bogen.

Subroutine HandleCircleEvent [dBvKOS97, S.164]

1: **Lösche** das **Blatt** γ von T. Aktualisiere die **Knickpunkte** in den inneren Knoten. Rebalanziere T falls notwendig. **Entferne** alle **Kreis-Ereignisse** aus Q die γ involvieren. Dazu muß man nur die Nachbarn von γ in T aufsuchen, denn das Kreis-Ereignis mit γ als mittlere Bogen wird gerade behandelt und wurde somit schon aus Q entfernt.
2: Speichere den Mittelpunkt des Kreises des Ereignisses als **Ecke** in D. Erzeuge Kanten-Records für die **Kante**, die den neuen Knickpunkt auf der Strandlinie enthält, und setzte die zugehörigen Pointer entsprechend.
3: Überprüfe für das Tripel aufeinanderfolgender Bögen, mit dem ursprünglich linken Nachbar von γ als mittleren, ob die Knickpunkte zueinander konvergieren. Falls ja, füge ein **neues Kreis-Ereignis** in Q ein sowie Pointer zwischen den entsprechenden Knoten in T und in Q. Verfahre ebenso mit den ursprünglich rechten Nachbarn als mittleren.

Tag 5.27

7.9 Lemma. Laufzeit und Speicherbedarf, [dBvKOS97, S.165]**.**
Der Algorithmus VoronoiDiagramm *benötigt* $O(n \log n)$ *Zeit und* $O(n)$ *Speicherplatz.*

Beweis. Die Basisoperationen am Baum T und der Eventqueue Q benötigen jeweils $O(\log n)$ Zeit nach Kapitel 2 und jene der doppelt-verbundenen Kantenliste $O(1)$ Zeit nach Kapitel 2 . Bei jedem Event führen wir eine konstante Anzahl solcher Operationen durch. Die Anzahl der Standortereignisse ist n. Jedes echte Kreisereignis beschreibt eine Ecke von $\mathrm{Vor}(P)$. Die falschen Alarme werden werden erzeugt und gelöscht wärend ein echtes Ereignis behandelt wird uns aus Q entfernt bevor sie dran kämen. Die für sie benötigte Zeit wird somit bei diese echten Ereignissen mitberücksichtigt. Folglich ist die Anzahl der Kreisereignisse höchstens $2n - 5$ nach 7.3.

Die Zeit- und die Speicherschranken folgen hieraus, denn auch falsche Alarme kann es höchstens so viele wie Standorte geben. □

Degeneriertheit [dBvKOS97, S.165]

Zwei **Standorte mit gleicher y-Koordinate**: Es ist egal in welcher Reihenfolge wir diese behandeln außer am Beginn, wo wir diesen Fall dann extra kodieren müssen, da

noch kein Bogen oberhalb des zweiten Punktes vorhanden ist. Falls **Ereignispunkte ident** sind (z.B. wenn mehr als 3 Standorte auf einem Kreis liegen), dann ist das Zentrum dieses Kreises ein Ecke von $V(P)$ mit Index mindestens 4. Wir benötigen keinen neuen Code, sondern können diese in beliebiger Reihenfolge abarbeiten und der Algorithmus wird uns zwei Ecken vom Index 3 am gleichen Ort und eine Kante zwischen ihnen liefern. Diese können wir zum Schluß verschmelzen. Falls ein **Standort** p genau **unterhalb des Knickpunktes** zweier Bögen der Strandlinie auftaucht, so spaltet der Algorithmus einen Bogen und füge Bögen ein. Einer dieser Bögen ist der mittlere Bogen eines Tripels, welches ein Kreisereignis beschreibt. Der tiefste Punkt dieses Kreises ist p. Dieses Kreisereignis kommt in die Eventqueue und wenn es behandelt wird, wird eine Ecke von $\mathrm{Vor}(P)$ korrekt erzeugt und wir können den Bogen der Länge 0 später entfernen. Falls die Brennpunkte dreier aufeinanderfolgende Bögen der Strandlinie **auf einer Geraden liegen**, dann definieren diese keinen Kreis und auch kein Kreisereignis. Es werden also auch alle degenerierten Fälle korrekt behandelt und wir erhalten:

7.10 Theorem. Korrektheit, [dBvKOS97, S.166]**.**
Der Algorithmus VoronoiDiagramm *berechnet das Voronoi-Diagramm von n-elementigen Punktmengen in $O(n \log n)$ Zeit und mit $O(n)$ Speicherbedarf.*

Beweis. Die Anzahl der echten Kreisevents können wir nicht mit der Anzahl der Ecken des Voronoi-Diagramms abschätzen, denn falls der Index k einer Ecke größer als 3 ist, erzeugt sie $k-2$ Kreisereignisse. Die Indizes aller Ecken ist das Doppelte der Kantenanzahl des Voronoi-Diagramms, also höchstens $2(3n-6)$ und somit dennoch ein $O(n)$. □

Voronoi-Diagramme für Geradensegmente

Verallgemeinerte Voronoi-Diagramme, [dBnvKO08, S.167]

Anstelle die Einzugsbereiche für eine endliche Menge $P \subseteq \mathbb{R}^2$ von Punkten zu betrachten kann man Voronoi-Diagramme auch für Mengen P anderer Objekte $p \subseteq \mathbb{R}^2$ behandeln. Dann ist $d(x,p) := \inf\{y \in q : d(x,y)\}$, die Voronoi-Zelle $V(p) := \{x \in \mathbb{R}^2 : d(x,p) < d(x,p')\, \forall p' \in P \setminus \{p\}\}$ und das Voronoi-Diagramm $\mathrm{Vor}(P) := \mathbb{R}^2 \setminus \bigcup_{p \in P} V(p)$.

Wenn z.B. s und s' zwei disjunkte Geradensegmente sind, dann besteht $\ell := \{x \in \mathbb{R}^2 : d(x,s) = d(x,s')\}$ aus mehreren Parabelbögen und Geradensegmenten: Seien nämlich $x \in \ell$ und $p \in s$ und $p' \in s'$ die eindeutig bestimmten Punkte mit $d(x,s) = d(x,p)$ und $d(x,s') = d(x,p')$. Es liegt p genau dann im Inneren der Strecke s, wenn x im Inneren des Streifens der durch die auf das Segment normalstehenden Geraden durch dessen Endpunkte begrenzt wird. Andererseits ist p genau dann ein Endpunkt, wenn x auf jener Seite der normalstehenden Geraden durch den Endpunkt liegt, welche den anderen Endpunkt nicht enthält. Der Teil von ℓ, für den p und p' ein Endpunkt ist, ist durch die Streckensymmetrale von p und p' gegeben. Jener Teil, für den genau einer der Punkte ein Endpunkt ist, ist durch die Parabel mit diesem Brennpunkt und der anderen Strecke als Leitlinie gegeben. Der Teil, wo beide Punkte innere Punkte sind, ist durch die Winkelsymmetrale (oder im degenerierten Fall durch die Mittellinie) der Geraden, die s und s' enthalten, gegeben. Die Ränder der Vornonoi-Zellen sind

somit endliche Durchschnitte solcher Begrenzungskurven bestehen also seinerseits aus Parabelbögen und Geradensegmenten.

Falls die Segmente einen gemeinsamen Randpunkt besitzen, ist die Situation komplizierter, denn dann sind alle Punkte im Sektor zwischen den Normalen gleich weit von den Segmenten entfernt. Darum setzen wir die (abgeschlossenen) Segmente im Folgenden als disjunkt voraus. Dies kann in vielen Anwendungen durch geringfügiges Verkürzen erreicht werden. Das Voronoi-Diagramm besteht dann aus der Vereinigung der Ränder der Voronoi-Zellen, hat also al Kanten Kurven die aus Parabelbögen und Geradensegmenten bestehen.

Sei also $S = \{s_1, \ldots, s_n\}$ eine endliche Menge disjunkter abgeschlossener Geradensegmente. Wir nennen die s_i wieder Standorte. Die Strandlinie kann entsprechend definiert werden und besteht nun aus Parabelbögen (dort wo der nächste Punkt in $\bigcup S = \bigcup_i s_i$ ein Endpunkt ist) und Geradenstücken auf Symmetralen (andernfalls).

Knickpunkte, [dBnvKO08, S.167]

Die Knickpunkte q können nun verschiedener Art sein:

1. Falls die zu q nächstgelegenen Punkte auf zwei Segmenten Endpunkte sind (und diese gleichweit wie die Sweepline entfernt sind), so läuft q auf einem Geradensegment (der Streckensymmetrale) von $\mathrm{Vor}(S)$.
2. Falls die zu q nächstgelegenen Punkte auf zwei Segmenten innere Punkte sind (und diese gleichweit wie die Sweepline entfernt sind), so läuft q ebenfalls auf einem Geradensegment (der Winkelsymmetrale) von $\mathrm{Vor}(S)$.
3. Falls genau einer der beiden nächstgelegenen Punkte ein Endpunkt ist, so läuft q auf einem Parabelsegment von $\mathrm{Vor}(S)$.
4. Falls der nächstgelegene Punkt Endpunkt eines Segments ist, der Abstandsvektor normal auf dieses steht und der Abstand zur Sweepline gleich groß ist, so läuft dieser Knickpunkt auf einer Geraden (dieser Normalen).
5. Falls das Innere eines Segments die Sweepline trifft, so ist der Schnittpunkt ein Knickpunkt, der auf diesem Segment läuft.

In den Fälle (4) und (5) läuft der Punkt nicht im Voronoi-Diagramm (da nur eine Segment involviert ist). Diese sind aber für die Strandlinie des Algorithmus relevant.

[dBnvKO08, S.168]

Wieder haben wir Standortevents und Kreisevents. Bei den Standortevents müssen wir nun zwischen oberen und unteren Endpunkt unterscheiden. Bei oberen wird ein Bogen in zwei Teil zerlegt und dazwischen tauchen 4 neue Bögen auf mit Knickpunkten vom Typ (4) und (5) zwischen diesen. Bei unteren wird der Knickpunkt, welcher der Schnittpunkt des Segments mit der Sweepline ist, durch zwei Knickpunkte von Typ (4) ersetzt mit einem parabolischen Bogen dazwischen. Auch für Kreisereignisse gibt es nun verschiedene Typen, die alle dem Verschwinden eines Bogens auf der Strandlinie entsprechen und die auftreten, wenn der unterste Punkt eines durch drei Standorte definierten Kreis erreicht wird. In den Mittelpunkten dieser Kreise treffen sich benachbarte Knickpunkte. Das entstehende Voronoi-Diagramm ist eine Zerlegung der Ebene durch gerade Kanten und Parabelsegmente. Wir können das wieder in

einer doppelt-verbundenen Kantenliste speichern. Für jede Fläche speichern wir den entsprechenden Standort. Somit können wir für jede orientierte Kante e die beiden gleich weit entfernten Standorte durch IncidenceFace(e) und IncidenceFace(Twin(e)) bestimmen. Zusammen mit den Ecken Origin(e) und Origin(Twin(e)) können wir die Form der Kante in konstanter Zeit bestimmen.

[dBnvKO08, S.168]

Der Sweepline Algorithmus VoronoiDiagramm verallgemeinert sich entsprechend, es müssen nur mehr Fälle unterschieden und behandelt werden. Es gibt aber wieder nur $O(n)$ viele Ereignisse und damit erhält man:

7.11 Theorem, [dBnvKO08, S.169]**.**
Das Voronoi-Diagramm einer Menge von n disjunkten Geradensegmenten kann in $O(n \log n)$ Zeit und mit $O(n)$ Speicherbedarf berechnet werden.

Tag 5.31

Bewegungsplanung, [dBnvKO08, S.168]

Dies läßt sich anwenden auf das Problem einen (kreisförmigen) Roboter unter Vermeidung von Hindernissen in Form von Geradensegmenten zu manövrieren. Wenn wir bei der Bewegung an maximaler Bewegungsfreiheit interessiert sind, so sollten wir uns entlang der Kanten des Voronoi-Diagramms bewegen. Die Bestimmung eines entsprechenden Wegs zwischen gegebenen Anfangs- und Endpunkten liefert folgender Algorithmus:

[dBnvKO08, S.170]

Input: Ein Menge $S = \{s_1, \ldots, s_n\}$ von disjunkten Geradensegmenten. Zwei Positionen q_{start} und q_{end} und ein Radius $r > 0$ so, daß die Kreisscheiben um diese Positionen mit Radius r die Menge $\bigcup S$ nicht treffen.

Output: Ein Pfad zwischen den Positionen so, daß keine Kreisscheibe mit Mittelpunkt auf ihm und Radius r die Menge $\bigcup S$ trifft, oder die Aussage der nicht-Existenz so eines Pfades.

1: Berechne das Voronoi-Diagramm Vor(S) innerhalb eines hinreichend großen Rechtecks.
2: Bestimme die Zellen die q_{start} bzw. q_{end} enthalten.
3: Finde den Schnittpunkt p_{start} der Normalen auf die zu q_{start} nächste Seite der Zelle mit Vor(S). Genauso für p_{end}. Füge diese Punkte in Vor(P) ein, indem die entsprechenden Bögen zerteilt werden.
4: Sei G der Graph des Voronoi-Diagramms. Entferne alle Kanten von G für welche die kleinste Distanz $\bigcup S$ höchstens r ist.
5: Stelle mittels einer Tiefensuche fest, ob ein Pfad von p_{start} nach p_{end} in G existiert. Falls ja, so gib diesen zusammen mit den Geradensegmenten von q_{start} zu p_{start} und analog für "end" aus. Andernfalls melde die Nichtexistenz.

7.12 Theorem, [dBnvKO08, S.170]**.**
Für n disjunkte Geradensegmente und zwei Positionen eines kreisförmigen Roboters kann die Existenz eines kollisionsfreien Wegs in $O(n \log n)$ Zeit und bei $O(n)$ Speicherbedarf bestimmt werden. □

Voronoi-Diagramme für entfernteste Punkte [dBnvKO08, S.170]

Abweichung der erzeugten Objekte von der Kreisform durch Koordinaten-Bestimmung mehrere Randpunkte $p \in P$ und Suche des Kreisrings (Bereich zwischen zwei konzentrischen Kreisen) mit minimaler Dicke (Differenz der Radien), welcher diese Punkte umfaßt. Klarerweise müssen die beiden Randkreise jeweils mindestens einen Punkt aus P enthalten, andernfalls ließe sich der Kreisring verkleinern. Es können folgende Fälle eintreten:

- Einer der Randkreise enthält mindestens 3 Punkte aus P (und legt damit den Mittelpunkt fest) und der andere mindestens einen Punkt.
- Beide Kreise enthalten jeweils mindestens 2 Punkte und die Streckensymmetralen sind nicht parallel.

Obwohl dies ähnlich zum Problem des kleinsten P umfassenden Kreises scheint, können wir nicht die Methode von damals verwenden, denn ein weiterer Punkt, welcher nicht im bisherigen optimalen Kreisring liegt, muß nicht am Rand des neuen optimalen Kreisrings liegen, siehe Aufgabe 7.15 in [dBvKOS97, S.178].

Im Voronoi-Diagramm ist der nächste Standort jener in dessen Zelle der Punkt liegt. Das Voronoi-Diagramm für den entferntesten Punkt ist jenes, welches die Ebene in Zellen zerlegt, die all jene Punkte enthalten, welche einen gemeinsamen Standort p als weitest entfernten haben. Die Zellen sind ebenfalls Durchschnitt von Halbebenen, nämlich $\bigcap_{p' \neq p} h(p', p)$. Dieser Durchschnitt ist konvex, kann aber auch leer sein: Für jeden Punkt $p \in P$ im Inneren der konvexen Hülle von P ist seine Zelle leer, denn für jede Kreischeibe, die ihm am Rand hat liegt dann ein Punkt $p' \in P$ außerhalb. Die Voronoi-Zelle eines Punktes ist genau dann nicht leer, wenn er eine Ecke der konvexen Hülle von P ist. Sei $p \in P$ so eine Ecke und q in der zugehörigen Voronoi-Zelle. Dann gehört der Halbstrahl auf der Geraden durch p und q der von q in die entgegengesetzte Richtung geht zu dieser Zelle. Die Zellen sind also unbeschränkt und somit bilden die Kanten dieses Voronoi-Diagramms eine im Sinne der Graphentheorie baumartige Struktur (d.h. zusammenhängend und ohne Zyklen (Ränder beschränkter Zellen)). In [dBnvKO08, Aufgabe 4.17] wird gezeigt, daß diese Voronoi-Diagramme ebenfalls $O(n)$ Ecken, Kanten und Flächen haben.

Der Mittelpunkt der kleinsten umfassenden Kreisscheibe aus Kapitel 4 ist entweder eine Ecke dieses Voronoi-Diagramms oder der Mittelpunkt zweier Standorte, deren Zellen eine gemeinsame Kante haben. Im ersten Fall gibt es 3 weitest entfernte Punkte im anderen Fall zwei.

Um dieses Voronoi-Diagramm in einer doppelt-verbundenen Kantenliste zu speichern fügen wir virtuelle Ecken-Records (mit der Richtung statt der Koordinaten) für die

Startpunkte der von ∞ kommenden orientierten Kanten hinzu. Die entsprechenden Next bzw. Prev Pointer der unbeschränkten sind dabei nil.

Algorithmus [dBnvKO08, S.172]

Bestimme zuerst die Ecken $p_1, \ldots, p_h$ der konvexe Hülle von P in zufälliger Reihenfolge. Entferne p_i sukzessive von $i := h$ bis $i := 4$ und speichere dabei jeweils in $\text{cw}(p_i)$ und $\text{ccw}(p_i)$ den aktuellen Nachbar im Uhrzeiger- und gegen dem Uhrzeigersinn. Dann bilden wir das Voronoi-Diagramm für entfernteste Punkte p_1, p_2, p_3. Und fügen sukzessive die Punkte $p_4, \ldots, p_h$ hinzu. Um dabei das Voronoi-Diagramm für $\{p_1, \ldots, p_i\}$ effektiv aus jenem von $\{p_1, \ldots, p_{i-1}\}$ zu konstruieren speichern wir Pointer für jeden Punkt p_j mit $j < i$ auf jene nach unendlich gehende orientierte Kante des Randes der Voronoi-Zelle von p_j.

Die Zelle von p_i wird zwischen der Zelle von $\text{cw}(p_i)$ und $\text{ccw}(p_i)$ liegen. Unmittelbar bevor p_i hinzugefügt wird, sind diese beiden Zellen benachbart und somit durch einen unendlichen Strahl der Streckensymmetrale getrennt. Der Punkt $\text{ccw}(p_i)$ hat einen Pointer auf diese Kante. Die Streckensymmetrale von p_i und $\text{ccw}(p_i)$ liefert eine neue unendliche Kante, die in der Zelle von $\text{ccw}(p_i)$ liegt und Teil des Randes der Zelle von p_i ist. Wir durchlaufen im Uhrzeigersinn die Kanten der Zelle von $\text{ccw}(p_i)$ um die, die Symmetralen schneidende, Kante zu finden. Auf der anderen Seite dieser Kante ist die Zelle eines Punktes p_j mit $j < i$ und die Symmetralen von p_j und p_i liefert ebenfalls eine Kante der Zelle von p_i. Wir fahren nun mit p_j anstelle $\text{ccw}(p_i)$ induktiv fort, bis wir schließlich bei $\text{cw}(p_i)$ landen und die entsprechende Symmtrale liefert die andere unbeschränkte Kante der Zelle von p_i. Wir fügen all diese neuen Kanten dem Voronoi-Diagramm hinzu und entfernen danach alle Kanten, die in der Zelle von p_i liegen.

7.14 Theorem [dBnvKO08, S.173]**.**
Es P eine Menge von n Punkten in der Ebene. Das Voronoi-Diagramm für die entferntesten Punkte kann in erwarteter $O(n \log n)$ Zeit und mit $O(n)$ Speicherbedarf berechnet werden.

Beweis. Für die Bestimmung der konvexe Hülle benötigen wir $O(n \log n)$ Zeit nach 1.1. Sei $h \leq n$ die Anzahl deren Ecken. Danach benötigen wir nur noch erwartete $O(h)$ Zeit zur Bestimmung dieses Voronoi-Diagramms: (Rückwärts Analyse) Falls der Rand der Zelle zu p_i aus k Kanten besteht, dann mußten wir k Zellen sowie deren Kanten im Diagramm zu $\{p_1, \ldots, p_{i-1}\}$ bearbeiten. Das Diagramm zu $\{p_1, \ldots, p_{i-1}\}$ hat nach Aufgabe 33 (siehe auch [dBnvKO08, Aufgabe 7.14]) höchstens $2i-5$ Kanten. Da jeder Punkt aus $\{p_1, \ldots, p_i\}$ mit gleicher Wahrscheinlichket der letzte hinzugefügte ist, ist die erwartete Anzahl der Kanten einer Zelle kleiner als 4, denn dieser Erwartungswert $\frac{2}{i}$ mal der Anzahl aller Kanten, also $\frac{2}{i}(2i-5) < 4$. Somit ist die erwartete Laufzeit für jeden Schritt $O(1)$ und die erwartetet (nicht die maximale) Gesamtlaufzeit für die Bestimmung des Voronoi-Diagramms aus der konvexen Hülle $O(h)$. Zusammen mit der maximalen Lauftzeit für die konvexe Hülle ergibt sich für die erwartete Laufzeit $O(n \log n) + O(h) = O(n \log n)$. □

[dBnvKO08, S.174]

Zurück zum Problem des dünnsten Kreisrings. Falls der innere Randkreis mindestens 3 Punkte enthält, dann liegt sein Mittelpunkt in einer Ecke des gewöhnlichen Voronoi-Diagramms. Falls der äußere Randkreis mindestens 3 Punkte enthält, dann liegt sein Mittelpunkt in einer Ecke des Voronoi-Diagramms für entfernteste Punkte. Falls beide Randkreise nur 2 Punkte enthalten, dann liegt der Mittelpunkt auf einer Kante des gewöhnlichen und einer des Voronoi-Diagramms für entfernteste Punkte. In allen Fällen erhalten wir eine vernünftige kleine Menge möglicher Mittelpunkte. Dazu bestimmen wir das Overlay (die gemeinsame Verfeinerung) der beiden Voronoi-Diagramme. Deren Ecken sind genau die Kandidaten für Mittelpunkte. Wenn wir die 4 Punkte der Randkreise kennen, so können wir in $O(1)$ Zeit den Mittelpunkt bestimmen.

Algorithmus: Bestimme die beiden Voronoi-Diagramme. Für jede Ecke des zweiten bestimme den nächsten Punkt von P und für jede Ecke des ersten den weitesten Punkt in P. Damit erhalten wir $O(n)$ Mengen von jeweils 4 Punkten. Für jede zwei Kanten aus je einen Diagramm die sich schneiden haben wir weiter 4 Punkte. Finde unter all diesen Quadrupel jenes welches den dünnsten Kreisring beschreibt.

7.15 Theorem [dBnvKO08, S.174]**.**
Der dünnste Kreisring, der eine gegeben Menge von n-Punkte umfaßt, kann in $O(n^2)$-Zeit bei $O(n)$ Speicherbedarf berechnet werden. □

Tag 6.02

Nachbemerkungen [dBvKOS97, S.166]

Voronoi Diagramme (nach [Vor07] und [Vor08]) laufen wegen [Dir50] auch unter den Namen Dirichlet-Mosaike. Aber sich kommen auch bereits in Descartes Principia Philosophia aus 1644 vor. Siehe [OBS92] für eine Übersicht.

Verbindet man die Standorte aneinandergrenzender Vornonoi-Zellen, so erhält man die sogenannte Delaunay Triangulierung, siehe Kapitel 9.

Zu einer Menge $P \subseteq \mathbb{R}^2$ sei $H(P)$ die Menge der Tangentialebenen an $z = x^2 + y^2$ zu Punkten $(p, |p|^2)$ für $p \in P$. Die vertikale Projektion des Durchschnitts der darüber liegenden Halbräume auf die xy-Ebene ist nach [ES86] genau das Voronoi-Diagramm von P, siehe Kapitel 11.

Der erste Algorithmus mit optimaler $O(n \log n)$ Laufzeit zur Bestimmung des Voronoi-Diagramms stammt von [SH75], der hier behandelte von [For87] in der Version von [GS88].

Es gibt Verallgemeinerungen auf $P \subseteq \mathbb{R}^d$ und auch andere Metriken.

8. Anordnungen und Dualität (Supersampling)

Ray-tracing [dBnvKO08, S.179]

Photorealistische Bilder werden am Computer durch raytracing erzeugt, dabei wird von der angenommenen Augenposition ein virtueller Strahl durch jeden Pixel des Bildes auf die darzustellenden Objekte gerichtet. Die Objekte auf die er trifft, sind

jene deren reflektiertes oder ausgesendetes Licht zur Farbe der Bildpunkts beitragen können (Falls die Objekte nicht durchscheinend sind, so ist nur das erste getroffene relevant). Um das reflektierte Licht zu bestimmen müssen Oberflächen Eigenschaften (wie Farbe, Reflexivität, etc.) sowie die Menge und Art des Lichts, welches von Lichtquellen oder durch Reflexion oder Streuung an anderen Objekten auf dieses trifft, berücksichtigt werden.

Problem dabei sind die durch die Rasterung des Bildes in Punkte (Quadrate) positiver Größe und entstehenden Treppeneffekte: Es kann sein das Bild einer Kante eines Objekts auf einen Pixel trifft und somit nur ein Teil des Pixels vom Bild des Objekts ausgefüllt wird. Es wäre somit besser nur einen entsprechenden Prozentanteil des vom Objekt kommenden Lichts zu berücksichtigen. Die Methode dafür ist das supersampling, wo nicht nur ein einziger Strahl durch (den Mittelpunkt) des Pixels betrachtet wird, sonder mehrere gut verteilte Strahlen. Es ist aber nicht günstig diese in einen regelmäßigen Muster zu verteilen, da das menschliche Auge sensibel auf reguläre Muster ist und entsprechende Artefakte generiert, sondern wir sollten Strahlen in einen zufälligen gute verteilten Muster verwenden.

Diskrepanz [dBnvKO08, S.180]

Für ein gegebenes Objekt ist die Diskrepanz einer Strahlenverteilung die Differenz der Prozentsätze der Treffer und des sichtbaren Bereichs. Wir müssen die maximale Diskrepanz für alle Objekte gewissen Typs, sagen wir für Polyeder, bestimmen und, da es nur auf die Projektion ankommt, also für alle Polygone. Im generischen Fall wird jeder Pixel nur von einer Kante des Polygons getroffen, also können wir uns auf Halbebenen anstelle von Polygonen beschränken. Sei das Quadrat $I := [0,1] \times [0,1]$ das Modell für einen Pixel. Für Halbebenen h bezeichnen wir mit $\mu(h) \leq 1$ die Fläche $h \cap I$. Für eine n-elementige Stichprobe $S \subseteq I$ sei $\mu_S(h) := |h \cap S|/|S| \leq 1$. Es heißt μ das kontinuierliche Maß und μ_S das diskrete Maß. Die Diskrepanz von h bzgl. S ist dann

$$\Delta_S(h) := |\mu(h) - \mu_S(h)| \leq 1$$

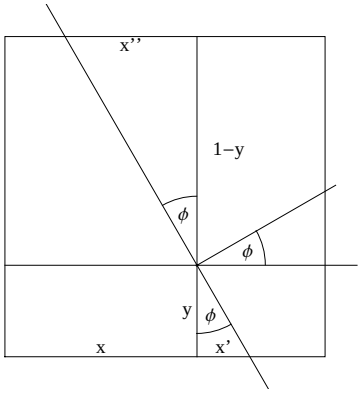
und wenn H die Menge aller Halbebenen bezeichnet, so ist die Halbebenen-Diskrepanz von S:

$$\Delta_H(S) := \sup_{h \in H} \Delta_S(h).$$

Wir wollen nun untersuchen, wie sich das kontinuierliche Maß μ unter Bewegungen ändert. Bei Translationen ist dies offensichtlich. Betrachen wir also nun Drehungen $R_{\varphi,p}$ mit Winkel φ um einen Punkt $p \in I$. Sei $h \in H$ mit $p \in \partial h$. Betrachten wir die beiden Schnittpunkte von ∂h mit ∂I. Falls diese auf gegenüberliegenden Seiten von I liegen, sagen wir der vom Inneren von h aus betrachtete rechte Schnittpunkt auf der x-Achse liegt, dann ist

$$f(k) := \mu(R_{\varphi,p}(h)) = x + \frac{1}{2}\left(k\,y^2 - k\,(1-y)^2\right) = x + k\,(y-1),$$

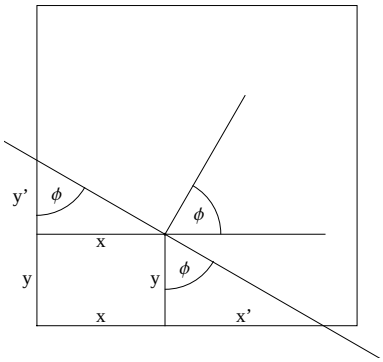
wobei $p = (x, y)$ und k der Anstieg der nach außen weisenden Normale ist. Also ist f streng monoton fallend.



Seien andererseits die Schnittpunkte auf angrenzenden Seiten von I, also o.B.d.A. auf der x-Achse und der y-Achse. Sei y der Abstand von p nur nächstgelegenden Seite und $x \geq y$ zur zweitnächsten. O.B.d.A. gehöre die Ecke zwischen diesen beiden Seiten zu h (andernfalls ersetze h durch die komplementäre Ebene $\bar{h}$). Dann ist

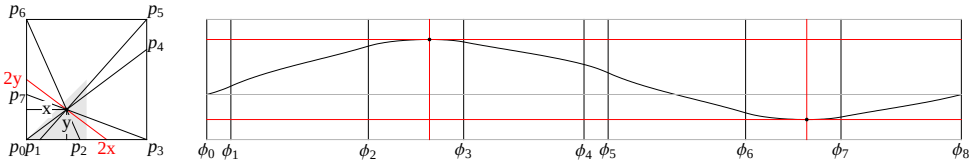
$$f(k) := \mu(R_{\varphi,p}(h)) = xy + \frac{1}{2}\left(k\,y^2 + \frac{x^2}{k}\right),$$
$$2f'(k) = y^2 - \frac{x^2}{k^2} \text{ und } 2f''(k) = -\frac{2x^2}{k^3} \leq 0,$$

als f konkav und das einzig mögliche Extremum ein Minimum von $f(k) = 2xy$ bei $k := \frac{x}{y}$. Wegen $\frac{x}{1-y} \leq k \leq \frac{1-x}{y}$ liegt dieses genau dann im zulässigen Bereich, wenn $x, y \leq \frac{1}{2}$.



Nummerieren wir nun die rechten Schnittpunkte von Geraden ∂h durch p, die eine Ecke von I treffen, beginnend bei der nächstgelegenen Ecke mit $p_0, p_1, \ldots, p_7, p_8 = p_0$, die zugehörigen Winkel mit $\varphi_0 < \cdots < \varphi_7$, $\varphi_8 = \varphi_0$ und die zugehörigen Anstiege mit $k_0 < k_1 < k_2 < k_3 < k_0 < \cdots < k_3 < k_0$ so hat $\mu(R_{\varphi,p}(h))$ Extrema bei $\tan\varphi = k = \frac{x}{y}$ und ist dazwischen monoton.

i	0	1	2	3	4	5	6	7
zw. p_i und p_{i+1}	↓	↓	Min.	↑	↑	↑	Max.	↓



Bestimmung der Halbebenen-Diskrepanz [dBnvKO08, S.182]

Da die Menge H unendlich ist, versuchen wir eine endliche Menge $H' \subseteq H$ mit $\sup_{h\in H} \Delta_S(h) = \sup_{h\in H'} \Delta_S(h) = \max_{h\in H'} \Delta_S(h)$ zu finden. Sei H' die Menge aller Halbebenen h, für welche $|\partial h \cap S| \geq 2$ oder $\partial h \cap S = \{p\}$ und $\varphi \mapsto \mu(R_{\varphi,p}(h))$ hat ein lokales Extremum bei $\varphi = 0$. Um $\sup_{h\in H} \Delta_S(h) = \sup_{h'\in H'} \Delta_S(h')$ nachzuweisen, genügt es zu jedem $h \in H$ ein $h' \in H'$ mit $\Delta_S(h') \geq \Delta_S(h)$ zu finden. Da $h \mapsto \Delta_S(h)$ nicht konstant ist (μ_S ist stückweise (solange $\partial h \cap S = \emptyset$) konstant, μ aber nicht), existieren $h \in H$ mit $\Delta_S(h) > 0$ und wir müssen nur für solche ein entsprechendes h' finden.

Sei nun $\partial h \cap S = \{p\}$ und 0 kein lokales Extremum von $\varphi \mapsto \mu(R_{\varphi,p}(h))$. Wie wir gezeigt haben, ist diese Funktion zwischen den beiden (lokalen) Extrema monoton. Angenommen $\mu_S(h) > \mu(h)$. O.B.d.A. ist $\mu(R_{\varphi,p}(h))$ monoton fallend, andernfalls ersetze φ durch $-\varphi$. Nun betrachte den minimalen Drehwinkel $\varphi > 0$, für welchen die gedrehte Ebene $h' := R_{\varphi,p}(h)$ in H' liegt, also ein lokales Minimum vorliegt oder $|\partial h' \cap S| \geq 2$ ist. Tritt letzeres ein, so ist $\mu_S(h') \geq \mu_S(h) > \mu(h) > \mu(h')$ und andernfalls ist $\mu_S(h') = \mu_S(h) > \mu(h) \geq \mu(h')$. In beiden Fällen folgt $\Delta_S(h') = \mu_S(h') - \mu(h') > \mu_S(h) - \mu(h) = \Delta_S(h)$. Ist andererseits $\mu_S(h) < \mu(h)$, so gilt für die gespiegelte Ebene $\bar{h}$ wegen $h \cap \bar{h} \cap S = \{p\}$: $\mu_S(\bar{h}) = 1 - \mu_S(h) + \frac{1}{|S|} > 1 - \mu(h) = \mu(\bar{h})$ und somit $\Delta_S(\bar{h}) = \mu_S(\bar{h}) - \mu(\bar{h}) > \mu(h) - \mu_S(h) = \Delta_S(h)$, also können wir für $\bar{h}$ anstelle von h wie zuvor argumentieren.

Falls $\partial h \cap S = \emptyset$ und $\mu(h) < \mu_S(h)$, so verschieben wir h in Richtung des nach innen weisenden Normalvektors so lange bis $\partial h \cap S \neq \emptyset$ ($\mu_S(h) > 0 \Rightarrow h \cap S \neq \emptyset$) um eine neue Ebene h' zu erhalten. Dann ist $h' \subset h$ also $\mu(h') < \mu(h)$ und $S \cap h' = S \cap h$ also $\mu_S(h') = \mu_S(h)$. Folglich ist $\Delta_S(h') = \mu_S(h') - \mu(h') > \mu_S(h) - \mu(h) = \Delta_S(h)$. Falls $h' \notin H'$, so ist $\partial h' \cap S = \{p\}$ für ein p und 0 kein lokales Extremum von $\varphi \mapsto \mu(R_{\varphi,p}(h))$, also können wir wie zuvor fortfahren. Falls $\partial h \cap S = \emptyset$ und $\mu(h) > \mu_S(h)$, so betrachten wir die komplementäre Ebene $\bar{h}$. Wegen $h \cap \bar{h} \cap S = \emptyset$ ist $\mu_S(\bar{h}) = 1 - \mu_S(h) > 1 - \mu(h) = \mu(\bar{h})$ und $\Delta_S(\bar{h}) = \mu_S(\bar{h}) - \mu(\bar{h}) = \mu(h) - \mu_S(h) = \Delta(h)$, also können wir für $\bar{h}$ anstelle von h wie zuvor argumentieren.

8.1 Lemma, [dBnvKO08, S.182]**.**
Sei $S \subseteq I$ eine n-elementige Teilmenge. Halbebenen h, welche die Diskrepanz bzgl. S maximieren, müssen eine der folgenden Typen sein:

- *∂h enthält genau einen Punkt aus $p \in S$ und $\varphi \mapsto \mu(R_{\varphi,p}(h))$ hat ein lokales Extremum bei $\varphi = 0$.*
- *∂h enthält mindestens zwei Punkte aus S.*

Die Anzahl der Kandidaten vom Typ (1) ist $O(n)$ und diese können in $O(n)$ Zeit bestimmt werden. Das kontinuierliches Maß jedes solchen Kandidaten kann in $O(1)$ bestimmt werden und das diskrete in $O(n)$ Zeit. Die Anzahl jener vom Typ (2) ist $n(n-1)/2$. □

Wir werden zeigen, daß auch für die Kandidaten vom Typ (2) das diskrete Maß insgesamt in $O(n^2)$ Zeit bestimmt werden kann. Daraus folgt dann:

8.2 Theorem, [dBnvKO08, S.183] **.**
Die Halbebenen-Diskrepanz eine n-elementigen Menge $S \subseteq I$ kann in $O(n^2)$-Zeit bestimmt werden.

Beweis. Dazu müssen wir nur das Maximum der Diskrepanzen für die Kandidaten vom Typ (1) und Typ (2) bestimmen. □

Tag 6.07

Dualität, [dBnvKO08, S.183]

Dualitätstransformation [dBnvKO08, S.183]

Wir wollen nun Dualitätstransformationen studieren die Punkte auf Geraden und umgekehrt abbilden. Eine solche ist gegeben durch $p = (p_x, p_y) \mapsto p^* := \{(x,y) : y = p_x\, x - p_y\}$ und $\ell := \{(x,y) : y = k\, x + d\} \mapsto \ell^* := (k, -d)$. Achtung: Diese ist für den Punkt $(0,0)$ und vertikale Geraden nicht definiert, aber wir können die Ebene etwas drehen und verschieben, sodaß die Dualität für endlich viele gegebene Punkte und Geraden funktioniert.

8.3 Bemerkung, [dBnvKO08, S.184]

Diese Dualitätstransformation hat folgende Eigenschaften:

- Inzidenzerhaltend: $p \in \ell \Leftrightarrow \ell^* \in p^*$.
- Monotonie: p liegt oberhalb $\ell \Leftrightarrow \ell^*$ liegt oberhalb p^*.

Beweis.

$$p_y \overset{>}{<} k\, p_x + d \Leftrightarrow -d \overset{>}{<} p_x\, k - p_y. \quad \square$$

Dual eines Geradensegments, [dBnvKO08, S.184]

Wenn wir das duale Objekt s^* eines Geradensegments $s = [p,q]$ als Vereinigung $s^* := \bigcup_{r \in s} r^*$ definieren (Achtung $\{\ell^*\} = \bigcap_{r \in \ell} r^* \neq \bigcup_{r \in \ell} r^*$), dann ist dies der Bereich zwischen den beiden kreuzenden Geraden p^* und q^*, denn für jeden Punkt $r \in s \subseteq \ell$ ist, $\ell^* \in r^* \subseteq s^*$ nach 8.3. Eine Gerade ℓ schneidet s genau dann, wenn $\ell^* \in s^*$, denn für $r \in \ell \cap s$ ist $\ell^* \in r^* \subseteq s^*$ und aus $\ell^* \in s^* = \bigcup_{r \in s} r^*$ folgt $\exists r \in s$: $\ell^* \in r^*$, also $r \in \ell$.

Tag 6.09

Geometrische Interpretation der Dualität, [dBnvKO08, S.184]

Betrachten wir die gleichseitige Parabel $\gamma : y = x^2/2$. Sei $p = (p_x, p_y) \in \gamma$. Dann hat die Tangente $\ell : y = kx + d$ an γ durch p den Anstieg der Parabel, also p_x, und wegen $p_y = k\,p_x + d$ ist $d = p_y - p_x^2 = -p_y$, d.h. $\ell = p^*$. Falls q beliebig ist, so betrachten wir den Schnittpunkt p der Parabel mit der Vertikalen durch q. Dann ist p^* und q^* parallel, also q^* parallel zur Tangente durch p. Da der vertikale Abstand von p^* und q^* gerade das Negative jenes von p und q ist, liegt q^* auf der anderen Seite der Tangente wie q und hat den gleichen Abstand.

Anwendbarkeit dieser Dualität. [dBnvKO08, S.185]

Sei S eine endlichen Punktmenge. Wir wollen das diskrete Maß jeder Halbebene, die zwei Punkte aus S am Rand hat, bzgl. S bestimmen. Dazu betrachten wir die Menge $\{p^* : p \in S\}$ von Geraden. Der Geraden $\ell(q_1, q_2)$ durch zwei Punkte $q_1, q_2 \in S$ entspricht dann $\ell(q_1, q_2)^* = q_1^* \cap q_2^*$. Betrachten wir nun die offene Halbebene unterhalb $\ell(q_1, q_2)$. Das diskrete Maß dieser Halbebene ist die gemittelte Anzahl der Punkte $p \in S$ unterhalb $\ell(q_1, q_2)$, also aus jene der Geraden p^* oberhalb des Punktes $\ell(p_1, p_2)^*$. Wir werden im nächsten Abschnitt einen effektiven Algorithmus beschreiben, der die Anzahl vorgegebener Geraden, die oberhalb eines Punktes liegen, zählt.

Degeneriertheit, [dBnvKO08, S.185]

Punkte in S mit gleicher x-Koordinate dualisieren zu Geraden mit gleichem Anstieg, also entspricht der Geraden durch zwei solche Punkte kein Schnittpunkt (im Endlichen) der beiden dazu dualen Geraden. In unserer Situation müssen wir die Diskrepanz für die höchstens n vielen vertikalen Geraden durch je zwei Punkte aus S also gesondert bestimmen.

Anordnungen von Geraden, [dBnvKO08, S.185]

[dBnvKO08, S.186]

Sei L eine Menge von n Geraden in der Ebene. Diese bilden eine Unterteilung der Ebene in Kanten, Ecken und Flächen. Manche Kanten und Flächen sind unbeschränkt. Wir nennen so etwas eine Anordnung von Geraden. Die Komplexität so einer Anordnung ist die Anzahl der Ecken, Kanten und Flächen.

8.4 Theorem, [dBnvKO08, S.186]**.**
Sei L eine Menge von n Geraden in der Ebene und $A(L)$ die zugehörige Anordnung.

- *Die Anzahl der Ecken von $A(L)$ ist höchstens $n(n-1)/2$.*
- *Die Anzahl der Kanten von $A(L)$ ist höchstens n^2.*
- *Die Anzahl der Flächen von $A(L)$ ist höchstens $1 + n(n+1)/2$.*

Gleichheit gilt genau dann, wenn $A(L)$ einfach ist, d.h. keine 3 Geraden durch einen Punkt gehen und keine zwei Geraden parallel sind.

Beweis. Die Ecken sind die Schnittpunkte der Geraden, also höchstens $n(n-1)/2$ und Gleichheit gilt genau dann, wenn je zwei Geraden sich in genau einen Punkt

schneiden und diese Schnittpunkte alle verschieden sind. Die Anzahl der Kanten auf einer Gerade ist um 1 größer als die ihrer Schnittpunkte, also höchstens $(n-1)+1$. Insgesamt also höchstens $n \cdot n$. Gleichheit gilt genau dann, wenn jede Kante von jeder anderen in jeweils verschiedenen Punkten getroffen wird. Um die Anzahl der Flächen abzuschätzen, fügen wir induktiv Geraden hinzu. Sei $L = \{\ell_1, \ldots, \ell_n\}$ und $L_i := \{\ell_j : j \leq i\}$. Wenn wir ℓ_i zu L_{i-1} hinzufügen, dann teilt jede Kante auf ℓ_i eine Fläche von $A(L_{i-1})$. Also nimmt die Anzahl der Flächen um die Anzahl der Kanten auf ℓ_i, d.h. höchstens i, zu. Somit ist die Gesamtanzahl der Flächen höchstens $1 + \sum_{i=1}^{n} i = 1 + n(n+1)/2$. Gleichheit gilt genau dann, wenn jede Kante ℓ_i alle Vorgänger in verschiedenen Punkten trifft. □

Die Komplexität von $A(L)$ ist somit höchstens quadratisch. Um die Anordnung in einer doppelt-verbundenen Kantenliste zu speichern, beschränken wir diese wieder durch ein hinreichend großes die Ecken umfassendes Rechteck. Wir können wieder den Plansweep Algorithmus FindIntersections aus 2.2 adaptieren um diese doppelt-verbundene Kantenstruktur zu bestimmen. Da die Anzahl der Schnittpunkte quadratisch ist, würden wir nach 2.3 dafür $O(n^2 \log n)$ Zeit benötigen. Dies versuchen wir durch einen inkrementellen Algorithmus zu unterbieten.

Der inkrementelle Algorithmus fügt die Geraden $\ell_1, \ldots, \ell_n$ nacheinander hinzu. Sei A_i jene Zerlegung, die durch das umfassende Rechteck B und $A(\{\ell_1, \ldots, \ell_i\}) \cap B$ gegeben ist. Um A_i aus A_{i-1} zu berechnen, müssen wir die Flächen von A_{i-1} teilen, die von ℓ_i getroffen werden. Dazu wandern wir längs ℓ_i von links nach rechts, siehe [dBnvKO08, S.187]. Wenn wir auf eine Fläche in deren Randkante e treffen, so durchlaufen wir den Rand der Fläche bis wir auf jene Kante treffen, wo ℓ_i die Fläche verläßt. Die dafür benötigte Zeit ist $O(k)$ wobei k die Komplexität der Fläche ist. Mittels Twin erhalten wir die nächste Fläche in die ℓ_i eintritt, u.s.f.. Falls ℓ_i eine Fläche in einer Ecke verläßt, dann durchlaufen wir die von dort ausgehenden Kanten bis wir die nächste von ℓ_i getroffene Fläche gefunden haben. Dafür benötigen wir $O(k)$ Zeit, wobei k der Index der Ecke ist. Die erste Fläche finden wir, indem wir die Kanten von A_{i-1} die am Rand des Rechtecks liegen auf Schnitt überprüfen. Die Anzahl dieser Kanten ist linear in i, also ist es auch die dafür benötigte Zeit. Das Zerlegen der Flächen f geschieht wie folgt: Sei dazu vorerst die Kante des Eintritts bereits zerlegt. Dann erzeugen wir zwei Flächen-Records für den Teil von f ober- und unterhalb von ℓ Falls ℓ die Fläche in einer Kante e' verläßt, so zerteilen wir diese und erzeugen eine neue Ecke $\ell \cap e'$. Weiters erzeugen wir Kanten-records für das Segment $\ell \cap f$. Nun müssen wir alle Pointer entsprechend anpassen (Vgl. mit der Overlay-Konstruktion aus Abschnitt 2.3) sowie den Records für f und e' entfernen. Die dafür benötigte Gesamtzeit ist linear in der Komplexität von f. Zusammengefaßt erhalten wir:

Algorithmus ConstructArrangement [dBnvKO08, S.188]

Input: Eine n-elementige Menge $L = \{\ell_1, \ldots, \ell_n\}$ von Geraden.
Output: Eine doppelt-verbundene Kantenliste für die Zerlegung durch B und $B \cap A(L)$, wobei B ein alle Ecken von $A(L)$ umfassendes Rechteck ist.

1: Bestimme ein umfassendes Rechteck B.
2: Erzeuge eine doppelt-verbundene Kantenliste A für B.
3: **for** $i = 1$ to n **do**
4: Suche die Kante e von A die den am weitesten links gelegenen Punkt Schnittpunkt von ∂B mit ℓ_i enthält.

5: Sei f die angrenzende Fläche.
6: **while** f ist nicht unbeschränkt **do**
7: Teile f wie oben beschrieben. Und ersetze f durch die nächste von ℓ_i getroffene Seite.

[dBnvKO08, S.187]

Das umfassende Rechteck B kann einfach in quadratischer Zeit bestimmt werden indem man unter allen Schnittpunkte den am weitesten links, rechts, oben, bzw. unten liegenden bestimmt.

Laufzeit [dBnvKO08, S.189]

Falls $A(L)$ einfach ist, so ist die Zeit um f zu zerlegen und die nächste Fläche zu finden linear in der Komplexität von f, also ist die Gesamtzeit um ℓ_i hinzuzufügen linear in der Summe der Komplexitäten der Flächen von A_{i-1} die ℓ_i treffen. Falls $A(L)$ nicht einfach ist, dann kann der Austrittpunkt aus einer Fläche eine Ecke sein und wir müssen um diese herumwandern um die richtige Kante zu finden. Die dabei angetroffenen Kanten sind Randkanten von Flächen deren Abschluß ℓ_i trifft. Dies führt zu folgenden Begriff

Zonen [dBnvKO08, S.189]

Die Zone einer Gerade ℓ der Anordnung $A(L)$ ist die Menge aller Flächen von $A(L)$ deren Abschluß von ℓ getroffen wird. Die Komplexität einer Zone ist die Summe der Komplexitäten all ihrer Flächen, d.h. die Summe der Anzahlen der Kanten und Ecken all dieser Flächen. Dabei werden manche Ecken bis zu vierfach gezählt. Die Zeit um ℓ hinzuzufügen ist linear in der Komplexität der Zone von ℓ in $A(\{\ell_1, \ldots, \ell_{i-1}\})$.

8.5 Zonen Theorem, [dBnvKO08, S.190]**.**
Die Komplexität der Zone einer Gerade in einer Anordnung von m Geraden der Ebene ist $O(m)$.

Beweis. Sei L eine m-elementige Menge von Geraden und $\ell \notin L$ eine weitere Gerade. O.B.d.A. sei ℓ die x-Achse. Wir setzen vorerst voraus, daß keine Gerade in L parallel zu ℓ ist. Jede e Kante von $A(L)$ begrenzt zwei Flächen f. Wir sagen e ist eine links/rechts begrenzende Kante von f, wenn f rechts/links von ihr liegt. Wir behaupten, daß die Anzahl der links begrenzenden Kanten in der Zone von ℓ höchstens $5m$ ist. Aus Symmetrie folgt dann gleiches für die rechts begrenzenden Kanten und damit das Theorem. Induktion nach m: $(m = 1)$ ist trivial. $(m > 1)$ Sei ℓ_1 jene Gerade aus L, die den am weitesten rechts liegenden Schnittpunkt mit ℓ hat. Wir setzen vorerst voraus, daß diese eindeutig ist. Nach Induktionsannahme hat die Zone von ℓ in $A(L \setminus \{\ell_1\})$ höchstens $5(m-1)$ viele links begrenzende Kanten. Wenn wir ℓ_1 hinzufügen kann die Anzahl auf zwei Arten zunehmen:

- Durch neue links begrenzende Kanten auf ℓ_1.
- Durch alte links begrenzende Kanten durch ℓ_1 zerteilt werden.

Sei v der erste Schnittpunkt oberhalb ℓ von ℓ_1 mit einer anderen Gerade von L und w der erste Schnittpunkt unterhalb ℓ von ℓ_1 mit einer anderen Gerade von L. Die Kante

die v mit w verbindet ist eine neue links begrenzende Kante auf ℓ_1. Weiters zerschneidet ℓ_1 alte links begrenzende Kanten in v und in w (so diese Punkte existieren). Wir behaupten, das dies die einzigen Vermehrungen darstellen: Sei ℓ_2 die Gerade die ℓ_1 in v trifft. Der Bereich oberhalb v zwischen ℓ_1 und ℓ_2 gehört nicht zur Zone von ℓ. Da ℓ_2 von links nach rechts durch ℓ_1 schneidet, liegt diese Bereich rechts von ℓ_1, also kann der Teil von ℓ_1 oberhalb v nichts zu den links begrenzenden Kanten der Zone beitragen. Falls eine alte links begrenzende Kante e der Zone durch ℓ_1 oberhalb von v zerschnitten wird, dann gehört der rechts von ℓ_1 liegende Teil von e nicht länger zur Zone, also verursachen solche Durchschnitte keine Zuname der Anzahl der links begrenzenden Kanten. Gleiches gilt für den unterhalb von w liegenden Teil von ℓ_1. Damit ist der Gesamtzuwachs längs ℓ_1 wie behauptet höchstens 3. Die Gesamtzahl der links begrenzenden Kanten ist in diesen Fall somit höchstens $5(m-1)+3 < 5m$.

Falls nun die Gerade ℓ_1 nicht eindeutig bestimmt ist, so verwenden wir irgendeine dieser Kanten. Dann ist nach ähnlichen wie obigen Argumenten der Zuwachs höchstens 4 (In der 3. Auflage des Buches wird dies (grundlos) abgeschwächt zur Aussage, daß diese Anzahl maximal 5 ist, wenn genau zwei Kanten durch den Schnittpunkt gehen). Somit ist die Gesamtanzahl von links-begrenzenden Kanten höchstens $5(m-1)+5 = 5m$.

Tag 6.10

Falls eine von ℓ verschiedene Gerade in L horizontal ist, so bewirkt eine kleine Rotation von ℓ nur eine Zuname der Komplexität der Zone von ℓ. Da wir eine obere Schranke der Komplexität der Zone beweisen, können wir o.B.d.A. annehmen, daß so eine Gerade nicht existiert. Falls $\ell \in L$, dann zeigt obiger Beweis, daß die Zone von ℓ in $A(L \setminus \{\ell\})$ höchstens $2 \cdot 5(m-1)$ Kanten hat und Hinzufügen von ℓ erhöht diese Anzahl höchstens um $4m-2$ (Jeweils höchstens m Kanten auf ℓ für die Flächen oberhalb bzw. unterhalb von ℓ und höchstens $m-1$ Kanten werden geteilt und diese zählen als links begrenzende und als rechts begrenzende Kanten). Dies vervollständigt den Beweis auch in den degenerierten Fällen. □

[dBnvKO08, S.191]

Da die Zeit um ℓ_i einzufügen linear in der Komplexität der Zone von ℓ_i ist und diese nach 8.5 seinerseits linear in i ist, ist die Gesamtzeit alle n Geraden einzufügen $\sum_{i=1}^{n} O(i) = O(n^2)$. Zusammen mit der nötigen $O(n^2)$ Zeit für Schritt 1 und 2 erhalten wir $O(n^2)$ als Gesamtlaufzeit des Algorithmuses ConstructArrangement. Da die Komplexität für einfaches $A(L)$ nicht geringer ist, ist dieser Algorithmus optimal.

8.6 Theorem.
Eine doppelt-verbundene Kantenliste für eine Anordnung von n Geraden in der Ebene kann in $O(n^2)$ Laufzeit berechnet werden. □

Niveau und Diskrepanz, [dBnvKO08, S.191]

Zurück zum Diskrepanz-Problem: Wir müssen für jede Gerade $\ell(p_1, p_2)$ mit $p_i \in S$ die Anzahl der Punkte in $p \in S$ zählen, die oberhalb, unterhalb oder auf dieser liegen. Dazu haben wir die Menge S dualisiert zur Menge $S^* = \{p^* : p \in S\}$ von Geraden und müssen nun für jede Ecke $p_1^* \cap p_2^* = \ell(p_1, p_2)^*$ von $A(S^*)$ feststellen, wieviele Geraden p^* für $p \in S$ unterhalb-, oberhalb oder durch diese gehen. Nach Konstruktion von $A(S^*)$ kennen wir letzeres. Da die Summe dieser 3 Anzahlen n ist, genügt es eine

weitere zu bestimmen. Wir definieren das Niveau (engl.: level) für $q \in \mathbb{R}^2$ als die Anzahl der echt oberhalb liegenden Geraden p^* für $p \in S$. Das Niveau jeder Ecke in $A(S^*)$ (auf einer Geraden $\ell \in S^*$) bestimmt man wie folgt:

- Bestimme zuerst das Niveau der am weitesten links auf ℓ liegenden Ecke $\ell \cap \ell_1$ in $O(n)$ Zeit, indem die Lage aller übrigen Geraden in S^* getestet wird.
- Danach durchlaufe die Ecken auf ℓ von links nach rechts. Dabei ändert sich das Niveau nur in Ecken und in diesen kann es unter Zuhilfenahme der dort ausgehenden Kanten bestimmt werden. Da S^* durch Dualisieren gewonnen wurde, enthält es keine vertikalen (also problematischen) Geraden.

Um die Niveaus aller Punkte auf ℓ zu bestimmen benötigen wir $O(n)$ Zeit und somit $O(n^2)$ Zeit für die Niveaus aller Ecken von $A(L)$. Mit diesen Zahlen können wir das diskrete Maß der Halbebene durch je zwei Punkte aus S bestimmen. Also erhalten wir dieses in $O(n^2)$ Zeit und damit ist Theorem 8.2 vollständig bewiesen.

Nachbemerkungen [dBnvKO08, S.192]

Die behandelte Dualität kann auch ohne Abstände beschrieben werden. Für Punkte p unterhalb der Geraden ist die duale Gerade jene, die durch die Berührpunkte der beiden Tangenten von p an die Parabel geht, denn q ist der Schnittpunkt der beiden Tangenten $t_\pm$ also q^* die Gerade, die durch die Berührpunkte $t_\pm^*$ geht. Auch für Punkte oberhalb der Parabel ist ein entsprechende Konstruktion möglich, für einen Hinweis dazu siehe [dBnvKO08, S.192].

Für andere Dualitätstransformationen, siehe [Ede87b]. Analoge Dualitäten funktionieren auch im höher-dimensionalen zwischen Punkten und Hyperebenen.

Eine andere wichtige Transformation in der algorithmischen Geometrie ist die Inversion, welche die Relation "Punkt innerhalb Kreis" auf "Punkt unterhalb Ebene" übersetzt. Dabei werden Punkte $p = (p_x, p_y)$ vertikal auf das Paraboloid $z = x^2 + y^2$ projeziert und Kreise auf die Ebene durch die vertikale Projektion des Kreises auf das Paraboloid: $(x-a)^2 + (y-b)^2 = r^2$ und $z = x^2 + y^2 \Rightarrow z = a(2x-a) + b(2y-b) + r^2$.

Auch Anordnungen von Geraden haben entsprechende höherdimensionale Analoga für Hyperebenen, siehe [Ede87b] für die Ergebnisse bis 1987 und [Hal04] für Neueres. In [EOS86] wurde der erste optimale Algorithmus dafür entwickelt. Dieser basiert auf einer d-dimensionalen Version des Zone-Theorems (siehe [ESS93]), welches die Komplexität einer Zone mit $O(n^{d-1})$ abschätzt.

Niveaus verallgemeinern sich auch aufs höher-dimensionale, siehe [Ede87b]. Verwandt damit ist das duale Problem, für n-punktige Mengen P die Anzahl der k-elementigen Teilmengen zu bestimmen, die durch eine Hyperebene von ihren Komplement in P getrennt werden können. Diese werden auch k-Mengen genannt.

Scharfe obere Schranken für die Komplexität der k-Niveaus oder auch der k-Mengen sind selbst in der Ebene unbekannt. In [ELSS73] wurde $O(n\sqrt{k})$ als obere Schranke bewiesen. In [PSS92] wurde das etwas verbessert und in [Dey97] und [Dey98] auf $O(nk^{1/3})$ gedrückt.

Hingegen ist die optimale Schranke für die Anzahl der höchstens k-elementigen Teilmengen, die von ihrem Komplement getrennt werden können, $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ nach [CS89]. Dieselbe Schranke gilt auch für die Anzahl der Punkte mit Niveau höchstens k.

Die ursprüngliche Motivation Anordnungen zu studieren kommt aus dem Motionplanning, siehe [GSS89, HS95a, HS95b, KLPS86, SS89, SS90]. Die hier behandelte Motivation Anordnungen zu studieren ist die Computer-Grafik: In [Shi91] wurde Diskrepanz in die Computer-Grafik eingeführt und durch [DM93], [DE93], [Cha93] und [dB96] algorithmisch entwickelt.

Tag 6.14

9. Delaunay Triangulierungen (Höheninterpolation)

Reliefs [dBnvKO08, S.197]

Bis jetzt haben wir Teile der Erdoberfläche als ebene Karten modelliert. Wir wollen nun die variierende Höhe mitberücksichtigen und diese durch ein Relief oder Gelände (engl.: terrain), also als Graph einer Funktion $f : \mathbb{R}^2 \supseteq G \to \mathbb{R}$ darstellen. Davon können wir dann 2-dimensionale perspektivische Darstellungen oder auch durch solche durch Höhenlinien anfertigen. Allerdings kennen wir die Höhe nicht für jeden Punkt der Erde, sondern nun für eine endliche Menge $P \subseteq G$ von gewissen Abtastpunkten (engl: sample point). Die naive Methode daraus ein Relief zu machen ist, für jeden Punkt die Höhe des nächstgelegenen Abtastpunkts zu verwenden. Das sieht unrealistisch aus, besser ist eine Triangulierung für P zu bestimmen, d.h. eine Zerlegung der Ebene in Dreiecke mit den Punkten aus P als Ecken, in den Ecken die bekannte Höhe aufzutragen und diese Punkte durch die entsprechenden räumlichen Dreiecke zu verbinden. Dieses polyederförmige Approximation des Geländes ist also Graph einer stetigen stückweise affinen Funktion. Für gegebenes P gibt es allerdings mehrere Triangulierungen, die nicht gleich-realistische Approximationen liefern, siehe Bild 9.3 in [dBnvKO08, S.198]. Es scheint also, daß wir Dreiecke mit sehr spitzen Winkeln vermeiden sollten. Eine optimale Triangulierung von P sollte jene (der endlich vielen) sein, die den kleinsten auftretenden Winkel maximiert.

Triangulierungen ebener Punkte-Mengen

[dBnvKO08, S.199]

Sei $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$ eine n-elementige Menge von Punkten. Unter einer Triangulierung von P verstehen wir eine maximale Menge von sich nicht schneidenden Verbindungsstrecken der Punkte aus P. Offensichtlich muß eine solche unter den endlich vielen Teilmengen nicht schneidender Verbindungsstrecken existieren. Die beschränkten Flächen jeder Triangulierung sind Dreiecke, denn diese sind offensichtlich Polygone und jedes solche kann nach 3.1 weiter trianguliert werden. Jede Kante des Randes der konvexen Hülle von P muß offensichtlich eine Kante jeder Triangulierung T von P sein. Somit ist die Vereinigung der beschränkten (abgeschlossenen) Flächen von T die konvexe Hülle von P und das Komplement der konvexen Hülle die einzige unbeschränkte Fläche von T. Wenn der Bereich G ein Rechteck ist, dann sollten

also jedenfalls dessen Ecken zu P gehören, damit die Dreiecke der Triangulierung G überdecken.

9.1 Theorem [dBnvKO08, S.199]**.**
Sei $P \subseteq \mathbb{R}^2$ eine n-elementige Menge von Punkten, die nicht alle auf einer Geraden liegen und sei k die Anzahl der Punkte die am Rand der konvexen Hülle von P liegen (das sind bisweilen mehr als die Ecken der konvexen Hülle). Dann besteht jede Triangulierung von P genau aus $2n-k-2$ Dreiecken und $3n-k-3$ Kanten.

Beweis. Sei m die Anzahl der Dreiecke einer Triangulierung von P, weiters $n_f = m+1$ die Anzahl der Flächen, n_e jene der Kanten und $n_v = n$ jene der Ecken. Jedes Dreieck hat 3 Kanten und die unbeschränkte Fläche hat k viele. Jede Kante gehört zu zwei Flächen, also ist $2n_e = 3m + k$. Wegen Eulers Formel ist $2 = n_v - n_e + n_f = n - \frac{3m+k}{2} + (m+1)$, also $m = 2n-k-2$ und $n_e = 3n-k-3$. □

Die Winkel-optimale Triangulierung [dBnvKO08, S.200]

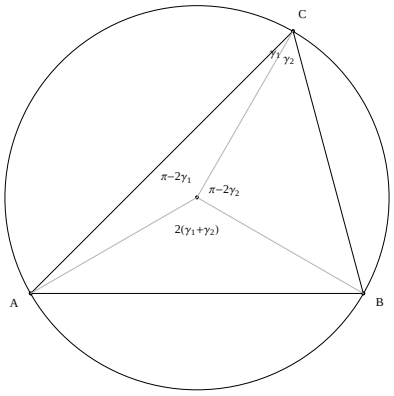
Sei T eine Triangulierung von P durch m Dreiecke. Seien die 3m Winkel der Dreiecke aufsteigend geordnet $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_{3m}$, dann nennen wir $A(T) := (\alpha_1, \ldots, \alpha_{3m})$ den Winkel-Vektor von T. Für verschiedene Triangulierungen vergleichen wir diese lexikographisch. Die Winkel-optimale Triangulierung ist jene mit lexikographisch größten Winkel-Vektor. Um diese zu erkennen benötigen wir:

9.2 Peripheriewinkelsatz von Thales [dBnvKO08, S.200]

Sei C ein Kreis, ℓ eine Gerade mit $\ell \cap C = \{a, b\}$. Seien p, q, r, s Punkte auf der gleichen Seite von ℓ mit $p, q \in C$, weiter r im Inneren und s im Äußeren von C. Dann gilt für die eingeschlossenen Winkel:

$$\sphericalangle asb < \sphericalangle aqb = \sphericalangle apb < \sphericalangle arb.$$

Beweis.



□

9.3 Flips illegaler Kanten, [dBnvKO08, S.200]

Sei $e = \overline{pp'}$ eine Kante einer Triangulierung T, die nicht im Rand der konvexen Hülle liegt, also gemeinsame Kante zweier Dreiecke $pp'q$ und $pp'q'$ ist. Falls das Polygon

$pqp'q'$ konvex ist, können wir die Kante $\overline{pp'}$ durch $\overline{qq'}$ ersetzen (das soll Kantenflip heißen) und somit eine andere Triangulierung T' erhalten, siehe Bild 9.4 in [dBnvKO08, S.200]. Wir nennen e illegale Kante, falls der kleinste Innenwinkel der ursprünglichen beiden Dreiecke kleiner als jener der neuen ist, also $A(T') > A(T)$ ist.

9.4 Lemma, [dBnvKO08, S.201]**.**
Es liege $e = \overline{pp'}$ am gemeinsamen Rand der beiden Dreiecke $pp'q$ und $pp'q'$. Dann ist e genau dann illegal, wenn q' im Inneren des Umkreises von $pp'q$ liegt. Wenn das 4-Eck $pqp'q'$ konvex ist und die Ecken nicht auf einem gemeinsamen Kreis liegen, dann ist genau eine der beiden Kanten $\overline{pp'}$ und $\overline{qq'}$ illegal.

Beweis. ($\Leftarrow$) Jeder der neuen Teilwinkel bei q und q' ist wegen 9.2 größer als der im anderen Dreieck liegende ursprüngliche Teilwinkel bei p oder p'. Also ist die Kante illegal.

($\Rightarrow$) Falls q' nicht im Inneren des Umkreises von p, p', q liegt, so sind beide Teilwinkel bei q' kleiner oder gleich den entsprechenden ursprünglichen Teilwinkel des Dreiecks $pp'q$ und aus Symmetriegründen gilt analoges für q'. Also ist der kleinste Winkel nicht größer als im ursprünglichen, d.h. die Kante ist nicht illegal.

Falls die 4 Punkte nicht auf einem Kreis liegen, so liegt einer der Punkte im Inneren des kleinsten Kreises der alle umfaßt, also existiert die entsprechende Kante. □

Legale Triangulierungen [dBnvKO08, S.201]

Wie nennen eine Triangulierung legal, falls sie keine illegalen Kanten besitzt. Jede Winkel-optimale Triangulierung ist somit legal. Legale Triangulierungen können wir aus Anfangstriangulierungen dadurch gewinnen, daß wir solange illegale Kanten flippen bis keine mehr vorhanden sind. Dies führt zum Ziel, da in jedem Schritt der Winkel-Vektor lexikographisch vergrößert wird, es aber nur endlich viele Triangulierungen gibt.

Algorithmus LegalTriangulation, [dBnvKO08, S.201]

Input: Eine Triangulierung T einer endlichen Punktmenge P.
Output: Eine legale Triangulierung von P.
1: **while** T enthält eine illegale Kante $\overline{pp'}$ **do**
2: Seien $pp'q$ und $pp'q'$ die angrenzenden Dreiecke. Ersetze $\overline{pp'}$ durch $\overline{qq'}$.
3: **return** T

Allerdings ist dieser Algorithmus zu langsam um interessant zu sein.

Delaunay Triangulierungen

[dBnvKO08, S.202]

Wir betrachten den dualen Graphen G des Voronoi-Diagramms einer endlichen Menge $P \subseteq \mathbb{R}^2$. Dieser hat für jede Voronoi-Zelle $V(p)$ (also jedem $p \in P$) eine Ecke und eine Kante zwischen zwei Ecken genau dann, wenn die entsprechenden Vornoi-Zellen eine gemeinsame Randkante besitzen. Die beschränkten Flächen von G entsprechen

dabei genau den Ecken von $\mathrm{Vor}(P)$, siehe Bild 9.5 in [dBnvKO08, S.202]. Wir betrachten die folgende Darstellung von G als Graphen mit den Ecken $p \in P$ und mit verbindenden Geradensegmenten als Kanten. Dieser heißt Delaunay-Graph $DG(P)$ (nach Boris Nikolaevich (engl.) Delone bzw. (franz.) Delaunay) von P.

Tag 6.16

9.5 Theorem [dBnvKO08, S.203]**.**
Der Delaunay Graph einer endlichen Menge $P \subseteq \mathbb{R}^2$ ist ein ebener Graph.

Beweis. Ein Segment $\overline{pp'}$ gehört genau dann zum Delaunay Graphen, wenn eine abgeschlossene Kreisscheibe C existiert die p und p' am Rand hat und keinen weiteren Punkt aus P enthält, denn nach 7.4.2 sind Mittelpunkte m dieser Kreise die inneren Punkte der Kante im Voronoi-Diagramm. Betrachten wir nun so ein Dreieck $\triangle pp'm$. Es ist die offene Kante $\overline{pm} \subseteq V(p)$ und ebenso $\overline{p'm} \subseteq V(p')$. Sei nun $\overline{qq'}$ ein weiteres Segment des Delaunay Graphens. Angenommen diese Segmente schneiden sich (damit ist $\{p, p'\} \cap \{q, q'\} = \emptyset$) und somit $q, q' \notin C \supseteq \triangle pp'm$, also muß $\overline{qq'}$ auch eine der beiden Kanten $\overline{pm}$ bzw. $\overline{p'm}$ schneiden. Analoges gilt mit vertauschten Rollen von p, p' und q, q'. Folglich muß eine der Kanten zum Mittelpunkt m eine der Kanten zum Mittelpunkt m' des Kreises für qq' schneiden, ein Widerspruch dazu, daß diese in verschiedenen Voronoi-Zellen liegen. □

Konvexität der Flächen des Delaunay Graphen, [dBnvKO08, S.203]

Die Ecken dieser Flächen sind Punkte in P und die Kanten entsprechen den benachbarten Voronoi-Zellen dieser Ecken. Ist q die gemeinsame Ecke der (konvexen) Voronoi-Zellen von $p_1, \ldots, p_k$, dann sind $p_1, \ldots, p_k$ die Ecken der q entsprechenden Fläche des Delaunay Graphens. Nach 7.4.3 liegen diese dann auf einen Kreis um q und bilden somit insbesonders ein konvexes Polygon.

Im generischen Fall liegen keine 4 Punkte aus P auf einem Kreis (wir sagen hier dafür: P sei in allgemeiner Lage). Dann sind die beschränkten Flächen des Delaunay Graphen Dreiecke. Im allgemeinen verstehen wir unter einer Delaunay-Triangulierung eine Triangulierung die aus dem Delaunay Graphen durch Hinzufügen von Kanten entsteht. Dies kann wegen der Konvexität der Flächen leicht erreicht werden.

Theorem 7.4 lautet nun:

9.6 Theorem, [dBnvKO08, S.204]**.**
Sei $P \subseteq \mathbb{R}^2$ endlich, $n := |P|$ und $q \in \mathbb{R}^2$. Dann gilt:

2. *Zwei Punkte von P definieren genau dann eine Kante des Delaunay Graphens von P, wenn ein abgeschlossene Kreisscheibe existiert, die diese beiden Punkte am Rand hat und keine weiteren Punkte aus P enthält.*
3. *Drei Punkte von P sind genau dann Ecken einer Fläche des Delaunay Graphens von P, wenn der Kreis durch sie keinen Punkt aus P im Inneren enthält.*

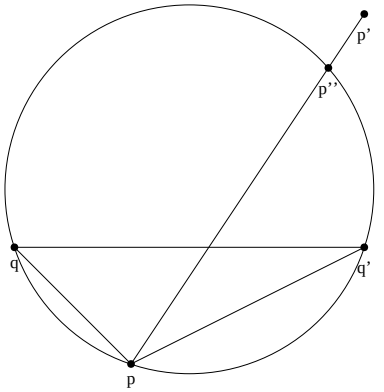
9.7 Folgerung, [dBnvKO08, S.204]**.**
Sei $P \subseteq \mathbb{R}^2$ endlich und T eine Triangulierung von P. Dann ist T genau dann eine

Delaunay-Triangulierung von P, wenn der Umkreis jedes Dreiecks von T keinen Punkt von P im Inneren enthält.

Beweis. Die Triangulierung T ist nach Definition genau dann eine Delaunay-Triangulierung von P, wenn jede Kante des Delaunay Graphens zu ihr gehört.

($\Rightarrow$) Sei T eine Delaunay Triangulierung. Dann sind die Ecken jedes Dreiecks auch Ecken einer beschränkten Fläche des Delaunay Graphens, also existiert eine Ecke q des Voronoi-Diagramms, die gemeinsame Ecke der Zellen $V(p)$ für alle Ecken des Dreiecks ist, also enthält der Kreis um q durch diese Ecken keine Punkte von P im Inneren.

($\Leftarrow$) Sei $\overline{pp'}$ eine Kante des Delaunay Graphens, die nicht zu T gehört. Dann schneidet diese in ein Dreieck $\triangle(pqq')$ von T, also liegt p' nach Voraussetzung nicht im Umkreis dieses Dreiecks. Nach 9.6.2 existiert aber eine abgeschlossene Kreisscheibe, welche die Endpunkte p und p' am Rand hat und die keine weiteren Punkte aus P enthält, also insbesonders nicht q und q'. Dies ist ein Widerspruch, denn selbst für die Schnittpunkte p und p'' von $\overline{pp'}$ mit dem Umkreis existiert keine solche (kleinere) Kreisscheibe.

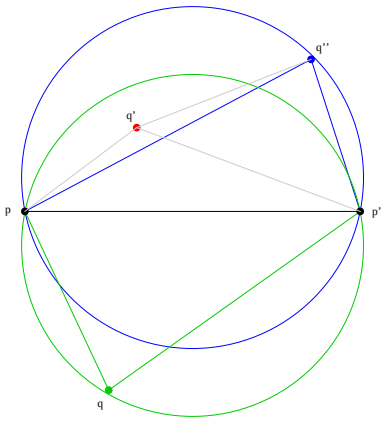


□

9.8 Theorem, [dBnvKO08, S.204]**.**
Sei $P \subseteq \mathbb{R}^2$ endlich und T eine Triangulierung von P. Dann ist T genau dann legal, wenn T eine Delaunay-Triangulierung von P ist.

Beweis. ($\Leftarrow$) Angenommen T hätte eine illegale Kante $\overline{pp'}$ mit angrenzenden Dreiecken $\triangle(pp'q)$ und $\triangle(pp'q')$. Nach 9.4 läge q' im Inneren des Kreises durch p, p', q, also wäre das Dreieck $\triangle(pp'q)$ nach 9.7 keines von T, ein Widerspruch.

($\Rightarrow$) Angenommen T wäre legal aber keine Delaunay-Triangulierung. Nach 9.7 gäbe es somit ein Dreieck $\triangle(pp'q)$ von T dessen Umkreis einen Punkt $q' \in P$ im Inneren enthält, der aber nicht im abgeschlossenen Dreieck liegt. O.B.d.A. sei $\overline{pp'}$ jene Kante des Dreiecks, die q und q' auf verschiedenen Seiten hat. Suche nun unter allen auf diese Weise konstruierbaren Quadrupeln jenes, bei welchem der Winkel $\sphericalangle pq'p'$ maximal ist. Nun betrachte das zweite an $\overline{pp'}$ angrenzende Dreieck $\triangle pp'q''$. Nach Voraussetzung ist $\overline{pp'}$ legal, also liegt q'' nach 9.4 nicht im Inneren des Umkreises von $\triangle(pp'q)$. Somit liegt der durch p und p' begrenzte Bogen des Kreises durch p, p' und q der q gegenüberliegt im Inneren des Umkreises von $\triangle(pp'q'')$, also auch q'. O.B.d.A. sei $\overline{q''p}$ jene Kante des Dreiecks $\triangle pp'q''$ die p' und q' auf verschiedenen Seiten hat.



Nach 9.2 ist $\sphericalangle pq'q'' > \sphericalangle pq'p'$, im Widerspruch zur Konstruktion des Quadrupels. □

9.9 Theorem, [dBnvKO08, S.205]**.**
Sei $P \subseteq \mathbb{R}^2$ endlich.

- *Jede Winkel-optimale Triangulierung von P ist eine Delaunay-Triangulierung von P.*
- *Jede Delaunay-Triangulierung von P maximiert unter allen Triangulierungen von P den minimalen Winkel, ist aber nicht notwendig Winkel-optimal.*

Beweis. Da jede Winkel-optimale Triangulierung offensichtlich legal ist, folgt aus 9.8, daß sie eine Delaunay Triangulierung ist.

Falls P in allgemeiner Lage ist, so ist der Delaunay Graph (eine und somit) die einzige Delaunay Triangulierung, also nach 9.8 auch die einzige legale Triangulierung und somit auch die einzige Winkel-optimale.

Falls P nicht in allgemeiner Lage ist, dann ist dennoch nach 9.8 jede Delaunay Triangulierung legal, aber nicht alle müssen Winkel-optimal sein. Wie in 9.4 zeigt man mittels des Satzes 9.2 von Thales, daß Kanten-Flips für Vierecke mit Umkreis den kleinsten Winkel ungeändert lassen und in Aufgabe 9.3 in [dBnvKO08, S.221] wird gezeigt, daß sich je zwei Triangulierungen durch Kanten-Flips ineinander überführen lassen. Also hat jede Triangulierung von fixen Punkten auf einem Kreis den selben minimalen Winkel und somit hat auch jede Delaunay Triangulierung von P den gleichen minimalen Winkel hat. □

Tag 6.17

Berechnen der Delaunay Triangulierung

[dBnvKO08, S.205]

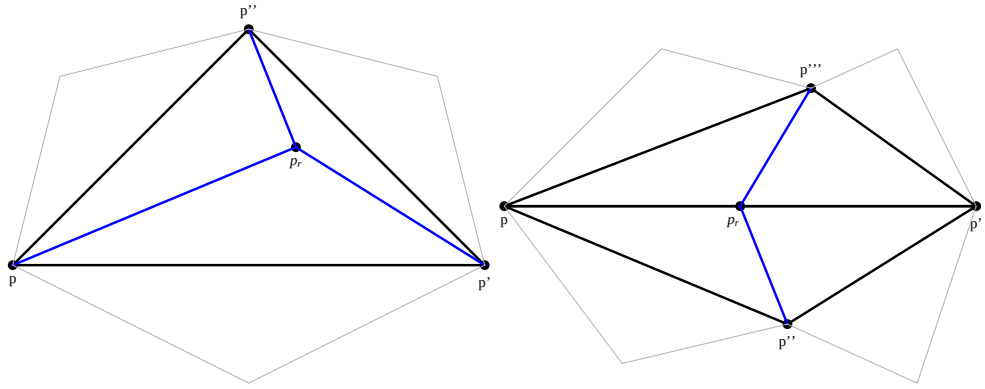
Wir wissen bereits aus Kapitel 7, wie wir das Voronoi-Diagramm $\mathrm{Vor}(P)$ aus P konstruieren können. Daraus erhalten wir leicht den Delaunay Graphen $DG(P)$ und folglich auch eine Delaunay Triangulierung, indem wir dessen Flächen, die mehr als 3 Ecken haben, triangulieren. Wir werden eine solche nun direkt berechnen, indem wir radomiziert inkrementell vorgehen (siehe Kapitel 4 und Kapitel 6).

Analog zu Kapitel 6 wählen wir dazu ein großer Dreieck (statt Rechteck), welches die Ecken von $\mathrm{Vor}(P)$ im Innere enthält. Dieses wird gebildet durch die höchstliegende Ecke $p_0 \in P$ und zwei weitere Punkte p_{-1} und p_{-2}. Dazu müssen wir diese weit genug entfernt so wählen, daß sie in keinem Kreis durch 3 Punkte aus P liegen (Details später). Wir werden dann eine Delaunay-Triangulierung von $P \cup \{p_{-1}, p_{-2}\}$ berechnen. Danach können wir p_{-1} und p_{-2} zusammen mit all ihren Kanten entfernen.

Induktionsschritt

Sei p_r der nächste hinzuzufügende Punkt. Wir suchen das Dreieck der aktuellen Triangulierung, welches p_r enthält, und verbinden dessen Ecken durch Kanten mit p_r. Falls p_r auf einer Kante e der Triangulierung liegt, so verbinden wir p_r mit den nicht auf der Kante liegenden Ecken der beiden angrenzenden Dreiecke. Damit erhalten wir

eine Triangulierung, die allerdings nicht Delaunay zu sein braucht, da nun vorhandene Kanten illegal geworden sein können. Wir wenden eine Routine LegalizeEdge an, die durch Kanten-Flips illegale Kanten in legale transformiert.



Algorithmus DelaunayTriangulation[dBnvKO08, S.206]

Input: Eine $n+1$-elementige Menge $P \subseteq \mathbb{R}^2$.
Output: Eine Delaunay-Triangulierung T von P.

1: Sei p_0 der lexikographisch höchste Punkt in P.
2: Sei p_{-1} und p_{-2} so gewählt, daß die Ecken von $\mathrm{Vor}(P)$ im Dreieck $\triangle(p_0 p_{-1} p_{-2})$ enthalten sind und initialisiere T mit diesem Dreieck.
3: Sei $p_1, \ldots, p_n$ eine Zufallspermutation von $P \setminus \{p_0\}$.
4: **for** $r = 1$ to n **do**
5: Finde das Dreieck $\triangle(pp'p'')$ von T welches p_r enthält.
6: **if** p_r liegt im Inneren des Dreiecks **then**
7: Füge Kanten von p_r zu p, p' und p'' zu T hinzu.
8: LegalizeEdge$(p_r, \overline{pp'})$, LegalizeEdge$(p_r, \overline{p'p''})$, LegalizeEdge$(p_r, \overline{p''p})$.
9: **else**
10: Es liege p_r auf der Kante $\overline{pp'}$.
11: Seien $\triangle(pp'p'')$ und $\triangle(pp'p''')$ die daran angrenzenden Dreiecke.
12: Füge Kanten von p_r zu p'' und p'''' zu T hinzu.
13: LegalizeEdge$(p_r, \overline{pp''})$, LegalizeEdge$(p_r, \overline{p''p'})$, LegalizeEdge$(p_r, \overline{p'p'''})$, LegalizeEdge$(p_r, \overline{p'''p})$.
14: Entferne p_{-1} und p_{-2} und dessen Kanten von T.
15: **return** T

[dBnvKO08, S.207]

Wir müssen erkennen, welche Kanten möglicherweise illegal geworden sind. Das kann nur passieren, wenn ein angrenzendes Dreieck geändert wurde, also müssen wir nur die Kanten der neuen Dreiecke überprüfen. Durch das Flippen illegaler Kanten können allerdings andere Kanten illegal werden, darum ruft sich LegalizeEdge rekursiv auf.

Routine LegalizeEdge [dBnvKO08, S.207]

Input: Ein Punkt q und eine Kante $e = \overline{pp'}$ eines Dreiecks der Triangulierung T.
Output: Das modifizierte T, wobei gegebenenfalls e ersetzt wurde.

1: **if** e ist illegal **then**
2: Sei $\triangle(pp'q')$ das in e an $\triangle(pp'q)$ angrenzende Dreieck.
3: Ersetze e durch $\overline{qq'}$ in T.
4: LegalizeEdge$(q, \overline{pq'})$ and LegalizeEdge$(q, \overline{p'q'})$.

Der Test in Zeile 1 kann üblicherweise durch Anwenden von 9.4 gemacht werden, allerdings muß man wegen p_{-1} und p_{-2} etwas aufpassen (siehe später).

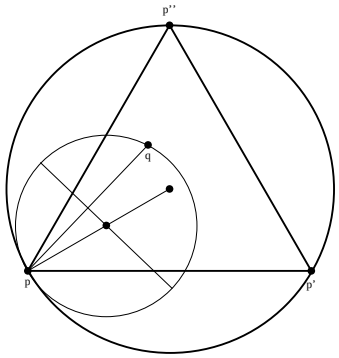
Korrektheit von DelaunayTriangulation, [dBnvKO08, S.207]

Wir müssen zeigen, daß keine illegalen Kanten nach Aufruf von LegalizeEdge übrig bleiben. Jede neu in LegalizeEdge(q, e) erzeugte Kante hat q als einen Endpunkt. Wir werden in 9.10 gleich zeigen, daß diese alle legal sind. Somit testet LegalizeEdge alle Kanten die möglicherweise illegal geworden sind. Er kann auch nicht in eine unendliche Schleife geraten, denn jeder Flip macht den Winkel-Vektor der Triangulierung größer.

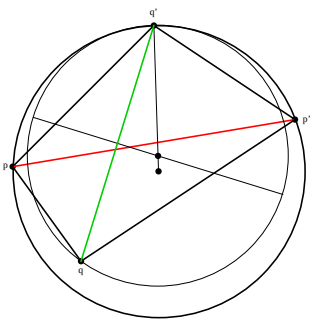
9.10 Lemma, [dBnvKO08, S.208]**.**
Jede beim Einfügen von p_r in DelaunayTriangulation *oder in* LegalizeEdge *neu erzeugte Kante ist eine Kante des Delaunay Graphens von $\{p_{-2}, p_{-1}, p_0, \ldots, p_r\}$.*

Beweis.

Betrachte die Kanten $\overline{qp}$, $\overline{qp'}$, $\overline{qp''}$ (und so vorhanden $\overline{qp'''}$). Da $\triangle(pp'p'')$ zu einer Delaunay-Triangulierung vor Einfügen von $q := p_r$ gehört, enthält sein Umkreis nach 9.6.3 keinen Punkt p_i mit $i < r$ im Inneren C. Durch Zusammenziehen erhalten wir eine Kreisscheibe $C' \subseteq C$ durch p und q, als gehört $\overline{qp}$ zum Delaunay-Graphen von $\{\ldots, p_{r-1}, p_r\}$ wegen 9.6.2 und genauso für die anderen Kanten.



Betrachte nun einen Kanten-Flip in LegalizeEdge: Dabei wird eine illegale Kante $\overline{pp'}$ durch $\overline{qq'}$ ersetzt, wobei q' die Ecke des an $\overline{pp'}$ angrenzenden Dreiecks auf der anderen Seite von q ist. Da $\triangle pp'q'$ ein Delaunay-Dreieck vor Einfügen von $q = p_r$ war und weil sein Umkreis q enthält (wegen 9.4, da $\overline{pp'}$ illegal ist), können wir diesen Umkreis zu einen Kreis C' verkleinern, der nur q' und q am Rand enthält und im Inneren keine Punkt aus $\{\ldots, p_r\}$. Wegen 9.6.2 ist somit $\overline{qq'}$ eine Kante des Delaunay-Graphens nach dem Einfügen von q. □



Das einen gegebenen Punkt enthaltende Dreieck, [dBnvKO08, S.208]

Wie in Kapitel 6 erzeugen wir einen gerichteten zyklischen Graphen D. Dessen Blätter entsprechen den Dreiecken der aktuellen Triangulierung mit entsprechenden Pointer dorthin. Die internen Knoten von D entsprechend Dreiecken in früheren Versionen der Triangulierung, die nun nicht mehr existieren. In Schritt (2) von DelaunayTriangulation initialisieren wir D mit dem einzigen Blatt $\triangle(p_0 p_{-1} p_{-2})$. Wenn wir ein Dreieck in 3 oder zwei Teile zerlegen, so fügen wir entsprechende Blätter und Pointer des ursprünglichen Dreiecks zu den neuen ein. Mittels D können wir nach jenem, den nächsten Punkt enthaltenden, Dreieck suchen (siehe [dBnvKO08, S.209]). Da jeder Knoten höchstens 3 ausgehende Kanten hat, benötigen wir hierfür nur lineare Zeit in der Suchtiefe, also in der Anzahl der Dreiecke in D, die den Punkt enthalten.

Bestimmen von p_{-1} und p_{-2}, [dBnvKO08, S.210]

Um die für p_{-1} und p_{-2} nötigerweise großen Koordinaten zu vermeiden, behandeln wir diese beiden Punkte symbolisch. Wir sagen ein Punkt liegt höher als ein zweiter, wenn er lexikographisch größer ist. Seien ℓ_- und ℓ_+ horizontale Gerade unterhalb und oberhalb P.

Wir stellen uns p_{-1} hinreichend weit rechts auf ℓ_- liegend vor, d.h. außerhalb jedes Kreises durch drei nicht kollineare Punkte und so, daß die Ordnung der Winkel von p_{-1} zu den Punkten aus P der lexikographischen Ordnung der Punkte entspricht.

Analog stellen wir uns p_{-2} hinreichend weit links auf ℓ_+ liegend vor, d.h. außerhalb jedes Kreises durch 3 nicht kollineare Punkte aus $P \cup \{p_{-1}\}$ und die Ordnung des Winkels von p_{-2} zu diesen Punkten der lexikographischen Ordnung der Punkte entspricht.

Die Delaunay-Triangulierung von $P \cup \{p_{-1}, p_{-2}\}$ besteht aus jener von P, sowie Kanten von p_{-1} zu jeder Ecke des rechten Randes der konvexen Hülle von P und Kanten von p_{-2} zu jeder Ecke des linken Randes der konvexen Hülle von P und die Kante $\overline{p_{-1}p_{-2}}$. Der tiefste Punkt von P und der höchste Punkt $p_0 \in P$ werden beide sowohl mit p_{-1} als auch mit p_{-2} verbunden.

Lage eines Punktes bzgl. einer Geraden, [dBnvKO08, S.210]

Während der Punktsuche müssen wir die Position eines Punktes q bzgl. der orientierten Kante von p nach p' bestimmen. Nach der Wahl der Punkte p_{-1} und p_{-2} sind äquivalent:

- q liegt links der Geraden von p nach p_{-1};
- q ist lexikographisch größer als p;
- q liegt links der Geraden von p_{-2} nach p.

Tag 6.21

Test auf Illegalität in LegalizeEdge, [dBnvKO08, S.210]

Sei $\overline{pp'}$ die zu testende Kante und q und q' die weiteren Ecken auf den beiden angrenzenden Dreiecken. Dann gibt es folgende Fälle:

- Keiner der 4 Punkte p, p', q, q' ist virtuell: Dies ist der Normalfall, wo wir auf Illegalität mittels 9.4 testen können.

- $\overline{pp'}$ ist eine Kante des umfassenden Dreiecks $\triangle(p_0p_{-1}p_{-2})$: Diese Kanten sind immer legal, da es nichts zum Flippen gibt.
- Andernfalls ist $\overline{pp'}$ genau dann legal, wenn $\min\{\text{index}(q), \text{index}(q')\} < \min\{\text{index}(p), \text{index}(p')\}$, wobei $\text{index}(q_{-1}) := -1$, $\text{index}(q_{-2}) := -2$ und $\text{index}(q) := 0$ für alle $q \in P$.

Im letzten Fall ist höchstens einer der beiden Punkte p und p' virtuell (d.h. sein Index ist < 0). Einer der beiden Punkte q und q' ist der neu eingefügte Punkt aus P, also ist auch höchstens der andere virtuell. Falls insgesamt genau einer virtuell ist, dann liegt dieser außerhalb des Kreises durch die übrigen 3 und somit ist nach 9.4 der angegebene Test korrekt. Auch andernfalls ist nach 9.4 der Test korrekt, denn p_{-2} liegt außerhalb jedes Kreises durch 3 Punkte aus $P \cup \{p_{-1}\}$, also ist die Kante mit Endpunkt p_{-1} legal und jene mit Endpunkt p_{-2} illegal.

Die Analysis

Erwartungswert für Anzahl der erzeugten Dreiecke, [dBnvKO08, S.211]

Wir wollen nun die strukturellen Änderungen, die der Algorithmus erzeugt, untersuchen, d.h. die Anzahl der Dreiecke die erzeugt und entfernt werden. Sei $P_r := \{p_1, \ldots, p_r\}$ und $DG_r := DG(\{p_{-2}, p_{-1}, p_0\} \cup P_r)$.

9.11 Lemma.
Der Erwartungswert für die Anzahl der vom Algorithmus DelaunayTriangulation *erzeugten Dreiecke ist höchstens* $9n + 1$.

Beweis. Der Algorithmus beginnt mit dem Dreieck $\triangle(p_0, p_{-1}, p_{-2})$. Im r. Schritt werden 1 oder 2 Dreiecke zerlegt und dabei 3 oder 4 neue Dreiecke erzeugt. Dabei wird die gleiche Anzahl von Kanten in DG_r erzeugt, nämlich $\overline{p_rp}$, $\overline{p_rp'}$, $\overline{p_rp''}$ (und möglicherweise $\overline{p_rp'''}$). Weiters werden für jede Kante, die wir in LegalizeEdge flippen, zwei neue Dreiecke und eine neue Kante mit Endpunkt p_r in DG_r erzeugt. Wenn es in DG_r also k Kanten mit Endpunkt p_r gibt, so wurden höchstens $2(k-3)+3 = 2k-3$ (resp. $2k-4$) neue Dreiecke erzeugt. Wir bezeichnen den Grad k von p_r in DG_r mit $\deg(p_r, DG_r)$. Wir wollen nun dessen Erwartungswert bzgl. aller Permutationen von P durch rückwärts-Analyse abschätzen. Nach 7.3 hat der Delaunay-Graph DG_r höchstens $3(r+3)-6$ Kanten. Drei dieser Kanten sind jene von $\triangle(p_0p_{-1}p_{-2})$ und somit ist die Summe der Grade aller Ecken aus P_r höchstens $2(3(r+3)-6)-6 = 6r$. Der Erwartungswert für den Grad $\deg(p_r, DG_r)$ eines Punktes $p_r \in P_r$ ist somit höchstens 6. Der Erwartungswert für die Anzahl der Dreiecke, die im Schritt r erzeugt werden, ist somit höchstens

$$\begin{aligned} E\Big(\text{Anzahl der in Schritt } r \text{ erzeugten } \triangle\Big) &\le E\Big(2\deg(p_r, DG_r) - 3\Big) \\ &= 2E\big(\deg(p_r, DG_r)\big) - 3 \le 2 \cdot 6 - 3 = 9. \end{aligned}$$

Der Erwartungswert für die Gesamtzahl der erzeugten Dreiecke ist somit (wegen der Linearität)

$$1 + \sum_{r=1}^{n} E\Big(\text{Anzahl der in Schritt } r \text{ erzeugten } \triangle\Big) \le 1 + 9\,n. \quad \square$$

9.12 Theorem [dBnvKO08, S.212]**.**
Die Delaunay Triangulierung einer n-elementigen Menge $P \subseteq \mathbb{R}^2$ kann in erwarteter $O(n \log n)$-Zeit und erwarteten $O(n)$ Speicherbedarf bestimmt werden.

Beweis. Die Korrektheit von DelaunayTriangulation folgt aus 9.10 und der Bemerkung davor.

Der Platzbedarf ist wegen 9.1 höchstens für die Suchstruktur D nicht linear. Da aber jeder Knoten in D einem Dreieck entspricht, folgt aus 9.11, daß der erwartete Platzbedarf ein $O(n)$ ist.

Laufzeit: Abgesehen von der Suche in Schritt 5 von DelaunayTriangulation ist diese proportional zur Anzahl der erzeugten Dreiecke, also der Erwartungswert nach 9.11 ebenfalls ein $O(n)$. Laufzeit im Schritt 5: Die Zeit um das p_r enthaltende Dreieck zu lokalisieren ist proportional zur Anzahl der besuchten Knoten in D, das ist $O(1)$ für das p_r enthaltende Dreieck der aktuellen Triangulierung plus lineare Zeit in der Anzahl der p_r enthaltenden Dreiecke die bislang erzeugt und danach wieder entfernt wurden. Ein Dreieck $\triangle(pp'q)$ kann aus zwei Gründen entfernt worden sein:

1. Ein neuer Punkt liegt im Inneren (oder am Rand) und zerlegt das Dreieck in 3 (oder 2) neue.
2. Ein Kanten-Flip ersetzt das Dreieck und ein Nachbar-Dreieck $\triangle(pp'q')$ durch 2 neue.

Im ersten Fall war das Dreieck vor dem Einfügen des Punktes ein Delaunay-Dreieck. Im zweiten Fall war das Nachbar-Dreieck $\triangle(pp'q')$ ein Delaunay Dreieck $\triangle$ und q die eingefügte Ecke. Da $\overline{pp'}$ illegal war, liegt q und damit auch $p_r \in \triangle(pp'q)$ im Umkreis von $\triangle$ Im beiden Fällen haben wir zu dem besuchten Dreieck $\triangle(pp'q)$ ein assoziiertes Delaunay-Dreieck $\triangle$, dessen Umkreis p_r im Inneren enthält, im gleichen Iterationsschritt entfernt.

Tag 6.23

Sei $K(\triangle)$ die Menge der Punkte aus P, die im Umkreis des Dreiecks $\triangle$ liegen. Ordnen wir somit jedem besuchten Dreieck sein assoziiertes Delaunay-Dreieck $\triangle$ mit $p_r \in K(\triangle)$ zu, so ist diese Zuordnung injektiv, denn jedes Dreieck $\triangle$ kann höchstens einmal das zugehörige Delaunay Dreieck sein, da es dabei entfernt wird. Somit ist die Gesamtzeit für Schritt 5:

$$\sum_{r=1}^{n} O\Big(1 + |\{\triangle \text{ ist Delaunay} : p_r \in K(\triangle)\}|\Big) =$$

$$= O\Big(n + \sum_{r=1}^{n} |\{\triangle \text{ ist Delaunay} : p_r \in K(\triangle)\}|\Big) = O\Big(n + \sum_{\triangle} |K(\triangle)|\Big),$$

wobei über alle im Laufe des Algorithmus erzeugten Delaunay-Dreiecke $\triangle$ summiert wird. Wir werden in 9.13 und 9.15a zeigen, daß der Erwartungswert hierfür $O(n \log n)$ ist. □

[dBnvKO08, S.213]

Sei $\triangle \in DG_r$. Für $r = 1$ ist $|K(\triangle)| = n$ und für $r = n$ ist $|K(\triangle)| = 0$. Da P_r eine zufällige Auswahl ist, vermuten wir $|K(\triangle)| \approx O(n/r)$. Dies ist zwar nicht korrekt, aber in Summe doch:

9.13 Lemma [dBnvKO08, S.213]**.**
Falls P ein n-elementige Menge $P \subseteq \mathbb{R}^2$ in allgemeiner Lage ist, so ist

$$E\Big(\sum_{\triangle} |K(\triangle)|\Big) = O(n \log n),$$

wobei die Summation über alle Delaunay-Dreiecke $\triangle$ geht, die durch den Algorithmus erzeugt werden.

Beweis. Da P in allgemeiner Lage vorausgesetzt ist, ist die Delaunay-Triangulierung nach Einfügen von p_r der eindeutige Delaunay-Graph DG_r. Es bezeichne T_r die Menge der Dreiecke von DG_r. Die im r. Iterationsschritt erzeugten Delaunay-Dreiecke sind $T_r \setminus T_{r-1} = \{\triangle \in T_r : p_r \text{ ist Ecke von } \triangle\}$ und somit

$$\sum_{\triangle} |K(\triangle)| = \sum_{r=1}^{n} \sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|$$

Für $q \in P$ sei

$$\begin{aligned} k(P_r, q) &:= |\{\triangle \in T_r : q \in K(\triangle)\}| \\ k(P_r, q, p_r) &:= |\{\triangle \in T_r : q \in K(\triangle),\ p_r \text{ ist Ecke von } \triangle\}| \end{aligned}$$

und somit ist

$$\begin{aligned} \sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)| &= \Big|\Big\{(\triangle, q) \in T_r \times (P \setminus P_r) : q \in K(\triangle), p_r \text{ ist Ecke von } \triangle\Big\}\Big| \\ &= \sum_{q \in P \setminus P_r} k(P_r, q, p_r). \end{aligned}$$

Sei vorerst P_r fixiert, d.h. wir betrachten nur solche zufällige Permutationen σ von P, wo $P_r = \{p_{\sigma(1)}, \ldots, p_{\sigma(r)}\}$ eine fixe Teilmenge von S ist. Der Wert von $\sum_{q \in P \setminus P_r} k(P_r, q, p_r)$ hängt dann nur von der Wahl von p_r ab. Ein Dreieck $\triangle \in T_r$ hat einen zufälligen Punkt $p \in P_r$ mit Wahrscheinlichkeit höchstens $3/r$ als Ecke und somit ist

$$E\Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) = E\Big(\sum_{q \in P \setminus P_r} k(P_r, q, p_r)\Big) \le \frac{3}{r} \sum_{q \in P \setminus P_r} k(P_r, q).$$

Jedes $q \in P \setminus P_r$ ist gleich wahrscheinlich p_{r+1} und somit ist

$$E\Big(k(P_r, p_{r+1})\Big) = \frac{1}{n-r} \sum_{q \in P \setminus P_r} k(P_r, q).$$

also

$$E\Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) \le 3\frac{n-r}{r} E\Big(k(P_r, p_{r+1})\Big).$$

Es ist $k(P_r, p_{r+1})$ nach Definition die Anzahl der Dreiecke $\triangle \in T_r$ mit $p_{r+1} \in K(\triangle)$. Nach 9.6 sind dies genau die $\triangle \in T_r$, die beim Einfügen von p_{r+1} entfernt werden, und nach 9.1 ist $|T_m| = 2(m+3) - 2 - 3 = 2m + 1$ und somit ist die Anzahl der beim Einfügen von p_{r+1} entfernten Dreiecke (d.h. in $T_r \setminus T_{r+1}$) um genau 2 kleiner als jene der erzeugten (d.h. in $T_{r+1} \setminus T_r$). Also ist

$$k(P_r, p_{r+1}) = |\{\triangle \in T_r : p_{r+1} \in K(\triangle)\}| = |T_r \setminus T_{r+1}| = |T_{r+1} \setminus T_r| - 2$$

und somit

$$E\Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) \leq 3\frac{n-r}{r} E\Big(|T_{r+1} \setminus T_r| - 2\Big).$$

Da diese Ungleichung für alle P_r gilt, können wir sie über die verschiedenen $P_r \subseteq P$ mitteln und sehen, daß sie auch für den Erwartungswert bzgl. aller Permutationen gültig ist. Da die Anzahl der Dreiecke, die beim Einfügen von p_{r+1} erzeugt werden, ident mit jener der Kanten in T_{r+1} mit Ecke p_{r+1} ist, und der Erwartungswert für diese Kanten höchstens 6 ist (siehe im Beweis von 9.11), folgt

$$E\Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) \leq 3 \cdot \frac{n-r}{r} \cdot (6-2) = 12\,\frac{n-r}{r}$$

und somit

$$\begin{aligned} E\Big(\sum_{\triangle} |K(\triangle)|\Big) &= \sum_{r=1}^{n} E\Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) \\ &\leq \sum_{r=1}^{n} 12\,\frac{n-r}{r} = 12n\Big(\sum_{r=1}^{n} \frac{1}{r} - 1\Big) \leq 12n \sum_{r=2}^{n} \frac{1}{r} \leq 12n\,\log n \quad \square \end{aligned}$$

Ein Rahmen für randomisierte Algorithmen [dBnvKO08, S.214]

Wir wollen nun ein gemeinsames Prinzip für die in den Kapitels 4, 6 und 9 behandelten randomizierten Algorithmen herausarbeiten.

Angenommen wir wollen eine geometrische Struktur $T(X)$ berechnen, welche durch eine Menge X von geometrischen Objekten festgelegt ist (z.B. die Delaunay-Triangulierung für eine Menge X von Punkten). Ein radomizierter inkrementeller Algorithmus erreicht dies durch Hinzufügen der Objekte aus X in zufälliger Reihenfolge und entsprechender Adaption der Struktur T. Dazu stellt es zuerst (im Lokalisierungsschritt) fest wo T geändert werden muß und paßt dann T (im Update-Schritt) entsprechend an. Die allgemeine abstrakte Methode ist jenes des sogenannten Konfigurationsraums:

Ein Konfigurationsraum ist ein Quadrupel (X, Π, D, K), wobei X den Input in Form einer endlichen Menge (geometrischer) Objekte bezeichnet. Sei $n := |X|$. Die Elemente von Π heißen Konfigurationen. Schließlich sind D und K Abbildungen, die jeder Konfiguration $\triangle \in \Pi$ Teilmengen $D(\triangle), K(\triangle) \subseteq X$ zuordnen. Von den Elementen in $D(\triangle)$ sagt man sie definieren $\triangle$ und von jenen in $K(\triangle)$ sie konfliktieren/killen $\triangle$. Die Konfliktgröße von $\triangle$ ist $|K(\triangle)|$ und $d := \max\{|D(\triangle)| : \triangle \in \Pi\}$ der sogenannte maximale Grad des Konfigurationsraums. Dabei soll gelten:

- $\forall \triangle \in \Pi : D(\triangle) \cap K(\triangle) = \emptyset$.
- $\{|D^{-1}(S)| : S \subseteq X\}$ sei beschränkt.

Sei

$$T(S) :== \{\triangle \in \Pi : D(\triangle) \subseteq S \text{ und } K(\triangle) \subseteq X \setminus S\} = \{\triangle \in \Pi : D(\triangle) \subseteq S \subseteq X \setminus K(\triangle)\}.$$

Ziel ist $T(X) = \{\triangle \in \Pi : K(\triangle) = \emptyset\}$ zu bestimmen.

Durchschnitte von Halbebenen als Beispiel, [dBnvKO08, S.215]**, vgl. 4.8.**

Hier ist X eine endliche Menge von Halbebenen in allgemeiner Lage. Es soll $T(X)$ den Durchschnitt von X beschreiben. Dazu setzen wir den Konfigurationsraum Π als die Menge der Schnittpunkte von Randgeraden der Halbebenen. Für $p \in \Pi$ sei $D(p)$ die Menge der (beiden) Halbebenen, die den Schnittpunkt p beschreiben und $K(p)$ die Menge jener Halbebenen, die Δ nicht enthalten. Für $S \subseteq X$ ist somit $T(S)$ die Menge der Ecken des Durchschnitts aller Halbebenen aus S, denn

$$T(S) := \Big\{p \in \Pi : \forall h \in X : p \in \partial h \Rightarrow h \in S \text{ und } h \in S \Rightarrow p \in h\Big\}.$$

Tag 6.24

Trapezoidale Zerlegungen als Beispiel, [dBnvKO08, S.216]**, vgl. 6.3.**

Hier ist X eine endliche Menge von Geradensegmenten in der Ebene. Der Konfigurationsraum Π besteht aus allen Trapezen $\triangle$, die in trapezoidalen Zerlegungen von Teilmengen $S \subseteq X$ vorkommen. Es ist $D(\triangle)$ die Menge der Segmente, die $\triangle$ beschreiben (siehe 6.1) und $K(\triangle)$ die Menge aller Segmente die $\triangle$ echt schneiden. Damit ist $T(S) := \{\triangle \in \Pi : D(\triangle) \subseteq S \subseteq X \setminus K(\triangle)\}$ die Menge der Trapeze der trapezoidalen Zerlegung für S.

Delaunay-Triangulierungen für Punkte in allgemeiner Lage als Beispiel, [dBnvKO08, S.216]**, vgl. 9.12.**

Hier ist X eine endliche Menge von Punkten in der Ebene in allgemeiner Lage. Der Konfigurationsraum Π besteht aus allen Dreiecken $\triangle$ von je 3 nicht kollineare Punkten in X. Es besteht $D(\triangle)$ aus den Ecken von $\triangle$ und $K(\triangle)$ aus jenen Punkten, die innerhalb des Umkreises von $\triangle$ liegen. Damit ist $T(S) := \{\triangle(pp'p'') \in \Pi : p, p', p'' \in S \subseteq X \setminus K(\triangle)\}$ die Menge der Dreiecke der (eindeutigen) Delaunay-Triangulierung von S.

Die strukturellen Änderungen, [dBnvKO08, S.216]

Sei $X = \{x_1, \ldots, x_n\}$ und $X_r := \{x_1, \ldots, x_r\}$ sowie $T_r := T(X_r)$ für $1 \leq r \leq n$. Dann ist

$$\begin{aligned} T_r \setminus T_{r-1} &= \Big\{\triangle : D(\triangle) \subseteq X_r \subseteq X \setminus K(\triangle) \text{ und nicht } D(\triangle) \subseteq X_{r-1} \subseteq X \setminus K(\triangle)\Big\} \\ &= \{\triangle \in T_r : x_r \in D(\triangle)\}, \quad \text{sowie} \\ T_r \setminus T_{r+1} &= \Big\{\triangle : D(\triangle) \subseteq X_r \subseteq X \setminus K(\triangle) \text{ und nicht } D(\triangle) \subseteq X_{r+1} \subseteq X \setminus K(\triangle)\Big\} \\ &= \{\triangle \in T_r : x_{r+1} \in K(\triangle)\}. \end{aligned}$$

9.14 Theorem. Abschätzung der Änderungen, [dBnvKO08, S.216]**.**
Sei (X, Π, D, K) ein Konfigurationsraum mit d und T_r wie zuvor definiert. Dann ist

$$E\Big(|T_r \setminus T_{r-1}|\Big) \leq \frac{d}{r} E\big(|T_r|\big).$$

Beweis. Rückwärts-Analyse: Wir benötigen eine Schranke für den Erwartungswert der Anzahl der Konfigurationen $\triangle \in T_r$, die verschwinden, wenn wir ein zufällig gewähltes Objekt x_r aus X_r entfernen. Sei vorerst $X_r = X_{r-1} \cup \{x_r\}$ fixiert. Nach Definition des maximalen Grad $d := \max\{|D(\triangle)| : \triangle \in \Pi\}$ des Konfigurationsraums ist dieser eingeschränkte Erwartungswert

$$\begin{aligned} E\Big(\Big|\Big\{\triangle \in T_r : x_r \in \triangle\Big\}\Big|\Big) &= \frac{1}{r} \sum_{x \in X_r} \Big|\Big\{\triangle \in T_r : x \in D(\triangle)\Big\}\Big| \\ &= \frac{1}{r} \cdot \Big|\Big\{(x, \triangle) : \triangle \in T_r,\ x \in D(\triangle)\Big\}\Big| \leq \frac{d}{r} \cdot |T_r|. \end{aligned}$$

Wenn wir nun noch über alle Teilmengen $X_r \subseteq X$ mitteln, erhalten wir

$$E\Big(|T_r \setminus T_{r-1}|\Big) = E\Big(|\{\triangle \in T_r : x_r \in D(\triangle)\}|\Big) \leq \frac{d}{r} E\big(|T_r|\big). \quad \square$$

9.15 Theorem, [dBnvKO08, S.217]**.**
Sei (X, Π, D, K) ein Konfigurationsraum mit d und T_r wie zuvor definiert. Dann ist

$$E\Big(\sum_{\triangle \in \bigcup_{r=1}^{n} T_r} |K(\triangle)|\Big) \leq \sum_{r=1}^{n} d^2\, \frac{n-r}{r^2}\, E(|T_r|).$$

Beweis. Wir gehen analog zu 9.13 vor:

$$\sum_{\triangle \in \bigcup_{r=1}^{n} T_r} |K(\triangle)| = \sum_{r=1}^{n} \Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)| \Big).$$

Sei

$$\begin{aligned} k(X_r, y) &:= |\{\triangle \in T_r : y \in K(\triangle)\}| \\ k(X_r, y, x_r) &:= |\{\triangle \in T_r : y \in K(\triangle), x_r \in D(\triangle)\}|. \end{aligned}$$

Wegen $T_r \setminus T_{r-1} = \{\triangle \in T_r : x_r \in D(\triangle)\}$ ist wieder

$$\begin{aligned} \sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)| &= \Big|\Big\{(\triangle, y) \in T_r \times (X \setminus X_r) : y \in K(\triangle),\ x_r \in D(\triangle)\Big\}\Big| \\ &= \sum_{y \in X \setminus X_r} k(X_r, y, x_r). \end{aligned}$$

Fixieren wir vorerst X_r. Da für $\triangle \in T_r$ die Wahrscheinlichkeit von $x \in D(\triangle)$ höchstens d/r ist, gilt

$$E\Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) = E\Big(\sum_{y \in X \setminus X_r} k(X_r, y, x_r)\Big) \leq \frac{d}{r} \sum_{y \in X \setminus X_r} k(X_r, y)$$

Da jedes $y \in X \setminus X_r$ gleich-wahrscheinlich als x_{r+1} auftritt, ist

$$E\Big(k(X_r, x_{r+1})\Big) = \frac{1}{n-r} \sum_{y \in X \setminus X_r} k(X_r, y)$$

und somit

$$E\Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) \le d\,\frac{n-r}{r}\, E\Big(k(X_r, x_{r+1})\Big) = d\,\frac{n-r}{r}\, E\Big(|T_r \setminus T_{r+1}|\Big).$$

Indem wir über alle Wahlen von $X_r \subseteq X$ mitteln, sehen wir das die letzte Ungleichung auch für den Erwartungswert bzgl. alle Permutationen gilt.

Nun müssen wir anders als im Beweis von 9.13 vorgehen, denn wir können jetzt die Anzahl der Konfigurationen, die im Schritt $r+1$ zerstört werden, nicht mit jener, der in diesem Schritt erzeugten Konfigurationen, abschätzen. Wenn $j(\triangle)$ den Schritt bezeichnet, in welchem $\triangle$ zerstört wird, so ist $\triangle \in T_r \setminus T_{r+1} \Leftrightarrow r+1 = j(\triangle)$ und somit

$$\begin{aligned}\sum_{r=1}^{n} d\,\frac{n-r}{r}\,|T_r \setminus T_{r+1}| &= \sum_{r=1}^{n} d\,\frac{n-r}{r} \sum_{\triangle \in T_r \setminus T_{r+1}} 1 \\ &= \sum_{\triangle} \sum_{r+1=j(\triangle)} d\,\frac{n-r}{r} = \sum_{\triangle} d\,\frac{n-(j(\triangle)-1)}{j(\triangle)-1},\end{aligned}$$

wobei $\triangle$ all jene Konfigurationen in Π durchläuft, die erzeugt und später zerstört werden. Sei $i(\triangle)$ jener Schritt, in welchem $\triangle$ erzeugt wird. Dann ist $i(\triangle) < j(\triangle)$ und somit

$$\frac{n-(j(\triangle)-1)}{j(\triangle)-1} = \frac{n}{j(\triangle)-1} - 1 \le \frac{n}{i(\triangle)} - 1 = \frac{n-i(\triangle)}{i(\triangle)},$$

also wegen $\triangle \in T_r \setminus T_{r-1} \Leftrightarrow r = i(\triangle)$:

$$\sum_{r=1}^{n} d\,\frac{n-r}{r}\,|T_r \setminus T_{r+1}| \le \sum_{\triangle} d\,\frac{n-i(\triangle)}{i(\triangle)} = \sum_{r=1}^{n} d\,\frac{n-r}{r}\,|T_r \setminus T_{r-1}|$$

Folglich ist

$$E\Big(\sum_{r=1}^{n} \sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) = \sum_{r=1}^{n} E\Big(\sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big) \le \sum_{r=1}^{n} d\,\frac{n-r}{r}\, E\Big(|T_r \setminus T_{r-1}|\Big)$$

und nach 9.14:

$$E\Big(\sum_{r=1}^{n} \sum_{\triangle \in T_r \setminus T_{r-1}} |K(\triangle)|\Big)\Big) \le \sum_{r=1}^{n} d\,\frac{n-r}{r}\,\frac{d}{r}\, E(|T_r|). \quad \square$$

9.15a Anwendung auf Delaunay-Triangulierung im Allgemeinen, [dBnvKO08, S.219]

Für Punkte, die sich nicht in allgemeiner Lage befinden, müssen wir anders als zuvor vorgehen. Sei P eine endliche Menge von Punkten in der Ebene und $X := P \setminus \{p_0\}$.

Weiters sei $\{p_{-2}, p_{-1}, p_0\} =: \Omega$ wie früher beschrieben. Jedes Tripel (p, p', p'') von Punkten $p, p', p'' \in X \cup \Omega$, die nicht auf einer Gerade liegen und ein positiv orientiertes Dreieck definieren, sei eine Konfiguration $\triangle$ mit $D(\triangle) := \{p, p', p''\} \cap X$ (also $d = 3$) und $K(\triangle)$ sei die Menge der Punkte aus X, die im Inneren des Umkreises von $\triangle$ oder auf dem Bogen von p über p' nach p'' liegen. Wir nennen eine Konfiguration $\triangle \in T(S)$ eine Delaunay-Ecke für $S \subseteq X$, denn dies ist genau dann der Fall, wenn p, p', p'' aufeinanderfolgende Punkte des Randes einer Fläche des Delaunay-Graphens $DG(\Omega \cup S)$ sind. Beachte, daß je 3 nicht kollineare Punkte 3 verschiedene Konfigurationen definieren.

Berechnung des Erwartungswerts, [dBnvKO08, S.219]

Wann immer DelaunayTriangulation eines neues Dreieck erzeugt, so ist dieses von der Form $\triangle(pp_rp')$, wobei p_r der in diesem Schritt eingefügte Punkt ist und $\overline{p_rp}$ und $\overline{p_rp'}$ Kanten des Delaunay-Graphen $DG(\Omega \cup P_r)$ nach Lemma 9.10 sind. Somit ist (p, p_r, p') eine Delaunay-Ecke von $DG(\Omega \cup P_r)$, also in T_r. Die Menge $K(\triangle)$ dieser Konfiguration $\triangle$ enthält alle im Umkreis des Dreiecks enthaltenen Punkte. Somit können wir die ursprüngliche Summe $\sum_{\triangle} |K(\triangle)|$ durch die nun definierte abschätzen, wobei die Summe über alle Delaunay-Ecken des Delaunay-Graphens $DG(\Omega \cup P_r)$ für ein $r \leq n$ genommen wird. Die Delaunay-Ecken sind genau die Ecken der Flächen des Delaunay-Graphen. Sei k die Eckenanzahl einer solchen Fläche. Dann ist diese in jeder Delaunay-Triangulierung in $k-2$ Dreiecke trianguliert und somit die Anzahl der Delaunay Ecken höchstens das Dreifache der Anzahl der Dreiecke. Für $r := |S|$ ist somit die Anzahl aller Delaunay-Ecken höchstens $3(2r+1) = 6r+3$, denn jede Delaunay-Triangulierung hat $2(r+3) - 5 = 2r+1$ Dreiecke nach 9.1.

Aus 9.15 folgt somit

$$\begin{aligned} E\Big(\sum_{\triangle} |K(\triangle)|\Big) &\leq \sum_{r=1}^{n} 3^2 \, \frac{n-r}{r^2} \, E(|T_r|) \leq \sum_{r=1}^{n} 3^2 \, \frac{n-r}{r^2} \, 3\,(2r+1) \\ &= 3^3 \sum_{r=1}^{n} \frac{n-r}{r} \, \frac{2r+1}{r} = 3^3 \cdot \Big(2 \sum_{r=1}^{n} \frac{n-r}{r} + \sum_{r=1}^{n} \frac{n-r}{r^2}\Big) \\ &\leq 3^3 \cdot \Big(2\,n\,\log n + n\big(1 + \sum_{r=2}^{\infty} \frac{1}{r(r-1)}\big)\Big) = 54\,n\,(\log n + 1). \end{aligned}$$

Dies vervollständigt den Beweis von 9.12.

Nachbemerkungen [dBnvKO08, S.220]

Triangulierungen in Dimension 2 und 3 spielen eine entscheidende Rolle sowohl in der numerischen Analysis (z.B. bei der Methode der finiten Elemente) als auch in der Computergraphik.

Wenn man anstelle der gegebenen Punkte auch zusätzliche Ecken (sogenannte Steiner-Punkte) für die Triangulierung zuläßt, dann spricht man von Meshing Problem, siehe [dBnvKO08, Kapitel 14].

In [Law72] wurde bewiesen, daß je zwei ebene Triangulierungen durch Kantenflips ineinander übergeführt werden können, siehe Aufgabe 48. Es wurde schon lange bemerkt, daß für gute Triangulierungen dünne Dreiecke vermieden werden sollten, siehe

[Bar77]. Daß es nur eine in Bezug auf den Winkelvektor optimale Triangulierung gibt, stammt von [Sib78]. Diese Methode den Winkel-Vektor zu verwenden ignoriert völlig die Höhen in den Punkten. In [Rip90] wurde gezeigt, daß die Rauheit, definiert als das Integral des Quadrats der L^2-Norm des Gradientens des Terrains, durch die Delaunay-Triangulierung unabhängig von den Höhendaten minimiert wird. Seit [DLR90] versucht man verbesserte Triangulierungen, die auch die Höhendaten mitberücksichtigen, zu erhalten.

Der hier behandelte radomizierte inkrementelle Algorithmus stammt von [EW86] und die Bestimmung von $\sum_{\triangle} |K(\triangle)|$ aus [Mul94]. In [BDS+92], [BT93] und [CS89] findet man andere solche Algorithmen.

Einige weitere geometrische Graphen sind Teilgraphen der Delaunay Triangulierung: Der Euklidisch minimal aufspannende Baum [Sha78], der Gabriel-Graph [GS69] und der relative Umgebungsgraph [Tou80].

Eine andere wichtige Triangulierung ist die minimale Gewichtstriangulierung (wobei das Gewicht die Summe der Länge aller Kante ist), siehe [AAC+96, BKMS96, DMM95, DM96]. Diese zu bestimmen ist NP-vollständig, nach [MR06].

Literaturverzeichnis

[AAC+96] O. Aichholzer, F. Aurenhammer, S.-W. Cheng, N. Katoh, M. Taschwer, G. Rote, and Y.-F. Xu. Triangulations intersect nicely. *Discrete Comput. Geom.*, 16:339–359, 1996. 106

[Agg84] A. Aggarwal. *The art gallery problem. Its variations, applications, and algorithmic aspects.* PhD thesis, Johns Hopkins Univ., Baltimore, MD, 1984. 26

[Bal95] I.J. Balaban. An optimal algorithm for finding segment intersections. *Proc.* 11^{th} *Annu. ACM Sympos. Comp. Geom.*, pages 211–219, 1995. 18

[Bar77] R. E. Barnhill. Representation and approximation of surfaces. In J. R. Rice, editor, *Math. Software III*, page 69120. Academic Press, New York, 1977. 106

[BDS+92] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom*, 8:5171, 1992. 106

[Ben75] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975. 53

[Ben79] J. L.. Bentley. Decomposable searching problem. *Inform. Process. Lett,*, 8:244–251, 1979. 53

[BKMS96] P. Belleville, M. Keil, M. McAllister, and J. Snoeyink. On computing edges that are in all minimum-weight triangulations. In *12th Annu. ACM Sympos. Comput. Geom.*, page V7V8, 1996. 106

[BO79] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979. 18

[BT93] J.-D. Boissonnat and M. Teillaud. On the randomized construction of the delaunay tree. *Theoret. Comput. Sci.*, 112:339354, 1993. 106

[Cha84] B. Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13:488–507, 1984. 26

[Cha86] B. Chazelle. Filtering search: a new approach to query-answering. *SIAM J. Comput.*, 15:703–724, 1986. 53

[Cha90a] A. Chazelle. Triangulating a simple polygon in linear time. *Proc. 31st Annu. IEEE Sympos. Found. COmput. Sci.*, pages 220–230, 1990. 26

[Cha90b] B. Chazelle. Lower bounds for orthogonal range searching, i: the reporting case. *J. ACM*, 37:200–212, 1990. 53

[Cha90c] B. Chazelle. Lower bounds for orthogonal range searching, ii: the arithmetic model. *J. ACM*, 37:439–463, 1990. 53

[Cha91] A. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991. 26

[Cha93] B. Chazelle. Geometric discrepancy revisited. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci.*, page 392399, 1993. 89

[Chv75] V. Chvátal. A combinatorial theorem in plane geometry. *J. Combin. Theory*, B, 18:39–41, 1975. 26

[CJY95] J. Choi, Sellen J., and C.-K. Yap. Precision-sensitive euclidean shortest path in 3-space. In *11th Annu. ACM Sympos. Comput. Geom.*, pages 350–359, 1995. 42

[Col86] R. Cole. Searching and storing similar lists. *J. Algorithms*, 7:202220, 1986. 66

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, ii. *Discrete Comput. Geom.*, 4:387421, 1989. 89, 106

[Dan63] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963. 42

[dB96] M. de Berg. Computing half-plane and strip discrepancy of planar point sets. *Comput. Geom. Theory Appl.*, 6:6983, 1996. 89

[dBnvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin Heidelberg New York, 3rd ed. edition, 2008. 74, 75, 76, 77, 78, 79, 80, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105

[dBvKOS97] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin Heidelberg New York, 1nd ed. edition, 1997. 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 77, 79

[dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin Heidelberg New York, 2nd ed. edition, 2000.

[DE93] D. Dobkin and D. Eppstein. Computing the discrepancy. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, page 4752, 1993. 89

[Dey97] T. K. Dey. Improved bounds for k-sets and k-th levels. In *Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci.*, page 156161, 1997. 88

[Dey98] T. K. Dey. Improved bounds on planar k-sets and related problems. *Discrete Comput. Geom.*, 19:373383, 1998. 88

[Dir50] G. L. Dirichlet. Über die reduktion der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *J. Reine Angew. Math.*, 40:209-227, 1850. 79

[DLR90] N. Dyn, D. Levin, and S. Rippa. Data dependent triangulations for piecewise linear interpolation. *IMA J. Numer. Anal.*, 10:137154, 1990. 106

[DM93] D. Dobkin and D. Mitchell. *Graphics Interface '93*, chapter Random-edge discrepancy of supersampling patterns. 1993. 89

[DM96] M. T. Dickerson and M. H. Montague. A (usually?) connected subgraph of the minimum weight triangulation. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, page 204213, 1996. 106

[DMM95] 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.*, page 238247, 1995. 106

[Dye86] M.E. Dyer. On a multidimensional search technique and its application to the euclidean one-centre problem. *SIAM J. Comput.*, 15:725–738, 1986. 42

[Ede87a] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987. 42

[Ede87b] H. Edelsbrunner. *H. EdelsbrunnerAlgorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987. 88

[EGS86] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317340, 1986. 66

[ELSS73] P. Erdös, L. Lovász, A. Simmons, and E. Straus. *A Survey of Combinatorial Theory*, chapter Dissection graphs of planar point sets, page 139154. North-Holland, Amsterdam, 1973. 88

[EOS86] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15:341363, 1986. 88

[ES86] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete Comput. Geom.*, 1:2544, 1986. 79

[ESS93] H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangements. *SIAM J. Comput.*, 22:418429, 1993. 88

[EW86] H. Edelsbrunner and E. Welzl. Halfplanar range search in linear space and $o(n^{0.695})$ query time. *Inform. Process. Lett.*, 23:289293, 1986. 106

[Fis78] S. Fisk. A short proof of chvátal's watchman theorem. *J. Combin. Theory*, B, 24:374, 1978. 26

[For87] S. J. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2:153174, 1987. 79

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme i. *Monatshefte für Mathematik und Physik*, 38:173?198, 1931. 19

[GS69] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259278, 1969. 106

[GS88] 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*, volume 40 of *NATO ASI Series F*. Springer-Verlag, 1988. 79

[GSS89] L. J. Guibas, M. Sharir, and S. Sifrony. On the general motion planning problem with two degrees of freedom. *Discrete Comput. Geom.*, 4:491-521, 1989. 89

[Hal04] D. Halperin. *Handbook of Discrete and Computational Geometry*, chapter Arrangements. Chapman & Hall/CRC, 2004. 88

[HS95a] D. Halperin and M. Sharir. Almost tight upper bounds for the single cell and zone problems in three dimensions. *Discrete Comput. Geom.*, 14:385410, 1995. 89

[HS95b] D. Halperin and M. Sharir. Arrangements and their applications in robotics: Recent developments. In K. Goldbergs, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Proc. Workshop on Algorithmic Foundations of Robotics*, Boston, MA, 1995. 89

[Jia09] Xiaoyi Jiang. Informatik II, Datenstrukturen und Algorithmen. Universität Münster, SS 2009. 14

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. 42

[Kha80] L. G. Khachiyan. Polynomial algorithm in linear programming. *U.S.S.R. Comput. Math. and Math. Phys.*, 20:53–72, 1980. 42

[Kir83] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12:2835, 1983. 66

[KLPS86] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:5971, 1986. 89

[Law72] C. L. Lawson. Transforming triangulations. *Discrete Math*, 3:365372, 1972. 105

[LL86] D. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inform. Theory*, 32:276–282, 1986. 26

[LP77] D. T. Lee and F.P. Preparata. Location of a point in a planar subdivisio and its applications. *SIAM J. Comput.*, 6:594–606, 1977. 26

[Lue78] G.S. Lueker. A data structure for orthogonal rangequeries. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 28–34, 1978. 53

[LW80] D.T. Lee and C.K. Wong. Quintary trees: a file structure for multidimensional database systems. *ACM Trans. Database Syst.*, 5:339–353, 1980. 53

[Mat92] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2:169186, 1992. 66

[Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31:114–127, 1984. 42

[Mei75] G. Meisters. Polygons have ears. *Amer. Math. Monthly*, 82:648–651, 1975. 26

[MR06] W. Mulzer and G. Rote. Minimum weight triangulation is np-hard. In *22nd Annu. ACM Sympos. Comput. Geom.*, page 110, 2006. 106

[Mul90] K. Mulmuley. A fast planar partition algorithm, i. *Journal of Symbolic Computation*, 10:253280, 1990. 66

[Mul94] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994. 106

[OBS92] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 1992. 79

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985. 66

[PS91] J. Pach and M. Sharir. On vertical visibility in arrangements of segments and the queue size in the Bentley-Ottman line sweep algorith. *SIAM J. Comput.*, 20:460–470, 1991. 18

[PSS92] J. Pach, W. Steiger, and E. Szemerédi. An upper bound on the number of planar k-sets. *Discrete Comput. Geom.*, 7:109123, 1992. 88

[Rip90] S. Rippa. Minimal roughness property of the delaunay triangulation. *Comput. Aided Geom. Design*, 7:489497, 1990. 106

[RS92] J. Ruppert and R. Seidel. On the difficulty of triangulating three-dimensional non-convex polyhedra. *Discrete Comput. Geom.*, 7:227–253, 1992. 26

[Sch28] E. Schönhardt. Über die zerlegung von dreieckspolyedern in tetraeder. *Math. Annalen*, 98:309?312, 1928.

[Sei91a] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom.Theory Appl*, 1:5164, 1991. 66

[Sei91b] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6:423–434, 1991. 42

[SH75] M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci.*, page 151162, 1975. 79

[Sha78] M. I. Shamos. *Computational Geometry.* PhD thesis, Dept. Comput. Sci., Yale Univ., New Haven, CT, 1978. 106

[Shi91] P. Shirley. Discrepancy as a quality measure for sample distributions. In F. H. Post and W. Barth, editors, *Proc. Eurographics'91.* Elsevier, 1991. 89

[Sib78] R. Sibson. Locally equiangular triangulations. *Comput. J.*, 21:243245, 1978. 106

[SS89] J. T. Schwartz and M. Sharir. *Geometric Reasoning*, chapter A survey of motion planning and related geometric algorithms, page 157169. MIT Press, Cambridge, MA, 1989. 89

[SS90] J. T. Schwartz and M. Sharir. *Algorithms and Complexity. Handbook of Theoretical Computer Science, vol. A*, chapter Algorithmic motion planning in robotics, page 391430. Elsevier, 1990. 89

[ST86] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669679, 1986. 66

[Tou80] G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recogn.*, 12:261268, 1980. 106

[Tur36] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230?265, 1936. 18

[TVW88] R.E. Tarjan and C.J. Van Wyk. An $o(n \log \log n)$-time algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17:143–178, 1988. 26

[Vor07] G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire: Sur quelques propri'etés des formes quadratiques positives parfaites. *J. Reine Angew. Math.*, 133:97178, 1907. 79

[Vor08] G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire: Recherches sur les parallélloèdres primitifs. *J. Reine Angew. Math*, 134:198287, 1908. 79

[Wel91] E. Welzl. Smallest enclosing disk (balls and ellipsoids). In *New Results and New Trends in Computer Science. Lecture Notes in Computer Science*, volume 555, pages 359–370. Springer-Verlag, 1991. 42

[Wil78] D.E. Willard. *Predicate-oriented database search algorithms.* PhD thesis, Aiken Comput. Lab., Harvard Univ., Cambridge, MA, 1978. 53

[Wil79] D.E. Willard. The super-b-tree algorithm. Technical report, Aiken Comput. Lab., Harvard Univ., Cambridge, MA, Report TR-03-79, 1979. 53

Index