

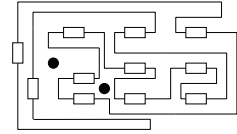
Geometrische Algorithmen

Einige einfache Definitionen:

- Punkt:** im n -dimensionalen Raum ist ein n -Tupel (n Koordinaten)
- Gerade:** definiert durch zwei beliebige Punkte auf ihr
- Strecke:** definiert durch ihre beiden Endpunkte
- Streckenzug:** Folge von Strecken (Kanten), wobei der Endpunkt (Knoten) einer Strecke der Anfangspunkt der nächsten ist
- Polygon:** (oder geschlossener Streckenzug) wenn der Anfangspunkt der ersten Strecke gleich dem Endpunkt der letzten ist
- konvex:** Polygon ist konvex wenn jede Strecke zwischen zwei Punkten in seinem Inneren komplett im Inneren liegt
- innen:** Ein Polygon, dessen Kanten sich nicht schneiden, umrahmt eine Region, deren Punkte *innen* sind
- Schreibweise:** Punkt (x,y) , Strecke $(u,v)-(x,y)$

Ist ein Punkt in einem Polygon?

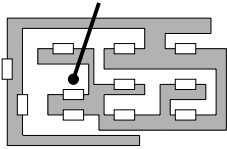
Motivation:
gegeben Schaltplan,
die Leitungen bilden
ein Polygon:



Frage: Kann man auf der Platine noch eine Leiterbahn von ● nach ● unterbringen?

Antwort: Ja, wenn beide Punkte innerhalb oder beide außerhalb des Polygons sind

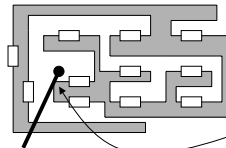
Punkt-in-Polygon-Problem



Ein erster Algorithmus:

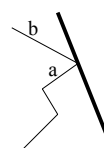
- wähle Punkt „unendlich“ weit außen und verbinde mit ●
- gehe auf der Verbindung in Richtung ●
- wenn eine Polygonkante gekreuzt wird und wir außen sind, gehen wir nach innen
- wenn eine Polygonkante gekreuzt wird und wir innen sind, gehen wir nach außen

Das Punkt-in-Polygon-Problem

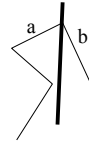


Beachte:

Was ist, wenn die Halbgerade einen Eckpunkt des Polygons schneidet?



beide angrenzenden Kanten auf der *selben* Seite:
innen/außen bleibt gleich



verschiedene Seiten:
innen/außen wechselt

Das Punkt-in-Polygon-Problem

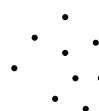
Ein Algorithmus für „ist (u,v) innerhalb von $P=(x_1, y_1), \dots, (x_n, y_n)$ “

Definiere eine Halbgerade H ausgehend von (u,v)
setze $Ort=0$ ($0=$ „innen“, $1=$ „außen“)
für alle Kanten $k=(x_i, y_i)-(x_{i+1}, y_{i+1})$
fall H und k sich schneiden aber nicht in (x_i, y_i)
dann setze $Ort=1-Ort$
falls H durch (x_i, y_i) geht
falls $(x_{i-1}, y_{i-1})-(x_i, y_i)$ und k auf gleicher Seite von H
dann setze $Ort=1-Ort$ {repariere Fehler bei $(x_{i-1}, y_{i-1})-(x_i, y_i)$ }

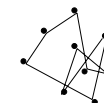
Aufwandsabschätzung: je Kante ein konstanter Aufwand also bei n Kanten: $O(n)$

Konstruktion von Polygonen

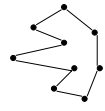
Eine Punktmenge, die die Eckpunkte eines Polygons darstellen soll, kann für viele Polygone stehen:



z.B.:



oder:



aber auch möglich:

Frage: Wie bekommt man garantiert ein Polygon mit überschneidungsfreien Kanten?

Konstruktion von Polygonen

Erster Gedanke: Ein „Greedy“ Algorithmus

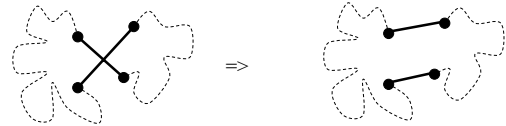
fange mit beliebiger Kante an
solange Polygon nicht fertig
füge beliebige Kante hinzu, die keine vorhandene Kante schneidet

geht das?



Konstruktion von Polygonen

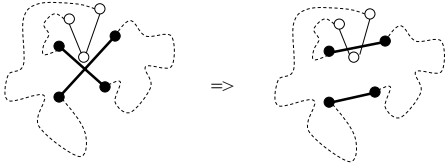
Zweiter Gedanke: Wir erinnern uns an den Handlungsreisenden.
Beliebig verbinden, dann Kreuzungen auflösen.



geht das?

Konstruktion von Polygonen

Auflösen von Kreuzungen kann neue Kreuzungen erzeugen:



klappt es trotzdem?

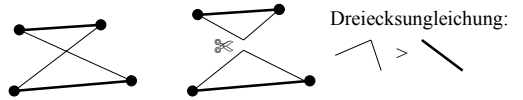
Konstruktion von Polygonen

Konstruktion durch Beseitigung von Kreuzung funktioniert.

Beweis: In jedem Schritt wird die Gesamtlänge aller Kanten um mindestens einen positiven Wert d verringert (s.u.).

Würde man beliebig oft Kreuzungen entfernen können, würde die Gesamtlänge negativ werden.

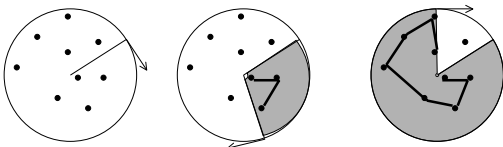
Also terminiert der Algorithmus und es gibt irgendwann keine Kreuzungen mehr.



Konstruktion von Polygonen

Beseitigung von Kreuzungen ist aufwendig. Geht es auch einfacher?

Dritter Gedanke: Radius einer Kreisscheibe über alle Punkte rotieren.



Füge immer Kante zum neu „überstrichenen“ Punkt hinzu.

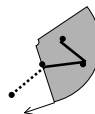
Schließlich:



Geht das?

Konstruktion von Polygonen

Beweisidee für rotierenden Radius:



Jede neu hinzugenommene Kante liegt komplett außerhalb des bis zur vorherigen Kante „überstrichenen“ Bereiches.

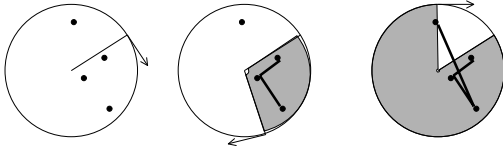
Also sind Überschneidungen ausgeschlossen.

Ist dieser Beweis korrekt?

Konstruktion von Polygonen

Das Verfahren des rotierenden Radius funktioniert nicht immer.

Gegenbeispiel:

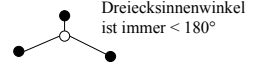


Wenn der Radius 180° zurückgelegt hat, ohne einen Punkt zu überstreichen, kann eine Kreuzung entstehen.

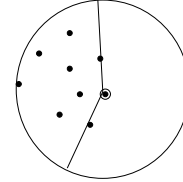
Konstruktion von Polygonen

Rotierender Radius funktioniert mit leichter Modifikation:

z.B. Wähle Kreismittelpunkt
innerhalb dreier Punkte:



oder: Wähle einen Randpunkt (äußerst rechts, ggf. obersten der rechten)
als Mittelpunkt des rotierenden Kreises:



Der gesamte zu überstreichende Winkel ist nun unter 180°

Konvexe Hülle

Definition: Die Konvexe Hülle einer Punktmenge ist das kleinste konvexe Polygon, das alle Punkte in seinem Innern hat.

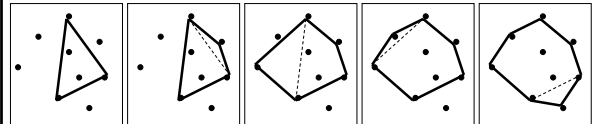


Motivation:

- leichtere 2-dimensionale Projektion
- Konstruktion eines „Zauns“ um alle Punkte herum

Konvexe Hülle

Erste Idee: Wir probieren es induktiv (Ausdehnungsalgorithmus):

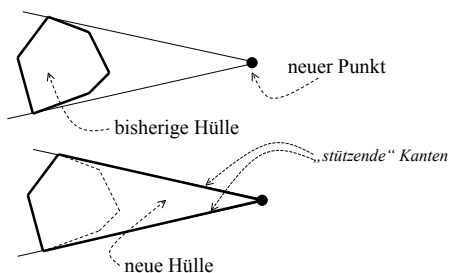


Anfang: Beginne mit Polygon aus beliebigen drei Punkten.

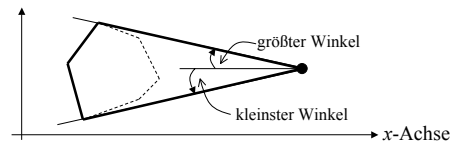
Schritt: Wähle beliebigen noch nicht bearbeiteten Punkt. Liegt er außerhalb des bisher erzeugten Polygons, dann modifiziere die bisher gefundene Hülle, so daß sie den neuen Punkt enthält.

Konvexe Hülle

Der Ausdehnungsalgorithmus (stretching algorithm) im Detail:



Konvexe Hülle

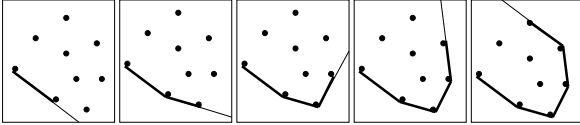


beginne mit beliebigem Dreieck
für alle Punkte P
wenn P nicht in der bisherigen Hülle,
dann füge Kante zum bisherigen Hüllpunkt mit dem kleinsten Winkel zu P und zum Punkt mit dem größten Winkel zu P ein;
entferne alle alten Kanten im Inneren der neuen Hülle

Aufwand: für alle Punkte alle Winkel, also $O(n^2)$

Konvexe Hülle

Ein „Einpackalgorithmus“ (Gift Wrapping)



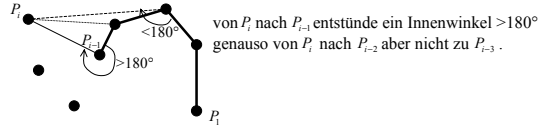
Beginne mit einem äußersten Punkt (z.B. äußerst links)
Verbinde immer zu dem Punkt,
den die Einwickelschnur als nächstes erreicht (stützende Kante).

Aufwandsabschätzung: wie beim Ausdehnungsalgorithmus
 $O(n^2)$ Winkelberechnungen zum
Finden der stützenden Kanten

Konvexe Hülle

Graham's Scan (ein schnellerer Hüllenalgorithmus)

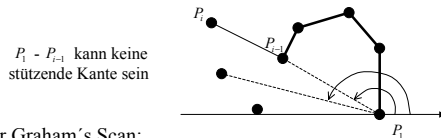
beginne mit einem äußersten Punkt P_i (z.B. äußerst rechts unten)
berechne die Winkel der Verbindungen aller Punkte zu P_i mit der x -Achse
sortiere alle Punkte entsprechend dieser Winkel in eine Liste
für alle Punkte P_j in der Liste
füge neue Kante von P_{i-1} zu P_j hinzu falls der Innenwinkel $< 180^\circ$
sonst entferne P_{i-1} und alle P_{i-2} aus der Hülle für die der Innenwinkel $> 180^\circ$



Konvexe Hülle

Aufwand für Graham's Scan:

Bemerkung: Sortieren ist nötig, damit ein zu entfernender Punkt
nicht später doch noch Teil der Hülle sein kann:



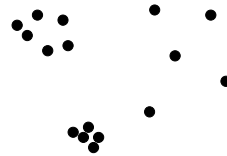
Aufwand für Graham's Scan:

Jeder Punkt wird höchstens einmal zur Hülle hinzugefügt
und höchstens einmal wieder entfernt => $O(n)$
Das Sortieren der Winkel kostet $O(n \log n)$
gesamt also $O(n \log n)$

Ballung

Ballung (engl. Clustering)

ist das Finden von Klassen in einer Punktmenge.

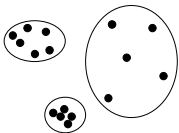


In wie viele Klassen
würden Sie diese
Punkte einteilen?

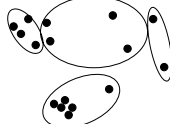
Warum?

Ballung

besser so



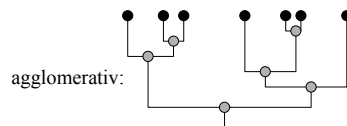
oder so



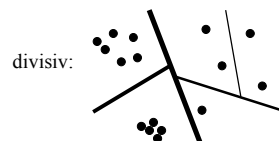
Ziel der Ballung könnte z.B. sein, Klassen so zu definieren,
daß der durchschnittliche Abstand von Punkten einer Klasse
möglichst klein wird (für eine gegebene Anzahl von Klassen).

Ballung

Verschiedene Arten: agglomerativ und divisiv



Füge in jedem Schritt
die zwei am wenigsten
entfernten Klassen
zusammen.

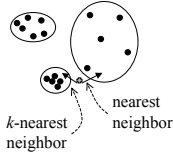


Beginne mit einer Klasse.
Teile in jedem Schritt eine
Klasse in zwei Unterklassen.

Klassifikation

Wozu Ballung?

- Unüberwachtes (d.h. ohne a-priori-Wissen) Bestimmen, was zusammen gehört, bzw. ähnlich ist.
- Klassifikation (ein bisher nicht gesehener Punkt wird einer Klasse zugeordnet, d.h. klassifiziert)
 - z.B. durch Zuordnen zur Klasse zu der auch der ihm am nächsten liegende bekannte Punkt gehört (nearest neighbor)
 - oder durch Zuordnen zur Klasse zu der die meisten der k ihm am nächsten liegenden Punkte gehören (k -nearest neighbor)



Nächstes Paar

Problem: Wie findet man, z.B. für agglomerative Ballung die beiden Punkte mit kleinstem Abstand zueinander?

Ein erster Algorithmus:

Berechne alle Distanzen
wähle das Paar mit der kleinsten Distanz

Dieser Algorithmus ist recht einfach, aber aufwendig:

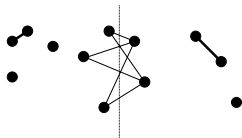
Aufwand für Berechnen aller $\frac{n(n-1)}{2}$ Distanzen ist $O(n^2)$.

Geht es auch schneller?

Nächstes Paar

Ein teile-und-herrsche-Algorithmus

- Die Idee:
- teile den Raum in zwei Teile ein,
 - berechne für jede Hälfte das nächste Paar
 - überprüfe alle in Frage kommenden „grenzüberschreitenden“ Paare
 - wähle von allen das mit der kleinsten Distanz

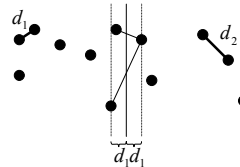


Frage: Welche „grenzüberschreitenden“ Paare kommen den in Frage?

Nächstes Paar

In Frage kommende grenzüberschreitende Paare:

Nur solche, deren Abstand zur Grenze kleiner ist, als der kleinste bisher gefundene Abstand der nicht grenzüberschreitenden Paare



Nächstes Paar

Wir machen es uns einfach:

Jede Trennebene wird immer senkrecht zur x -Achse gewählt.

=> leichtes Feststellen, auf welcher Seite ein Punkt liegt.

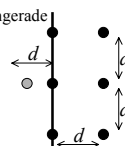
Und einmaliges Sortieren aller x -Koordinaten genügt, um immer eine Trennebene wählen zu können, so daß auf beiden Seiten gleich viele (± 1) Punkte liegen.

Nächstes Paar

Ist die Berechnung der Grenzüberschreitenden Paare nicht aufwendig?

Muß man im Worst-Case vielleicht fast jeden Punkt aus der einen Hälfte mit fast jedem Punkt aus der anderen Hälfte paaren?

Nein! Maximal 6 Punkte (d.h. konstante Zahl) aus der anderen Hälfte, denn auch wenn der Abstand in y -Richtung größer als d ist, kommt das Paar nicht in Frage, und innerhalb des Raumes der 6 Punkte kann kein weiterer sein (weil d minimal).



Nächstes Paar

Aufwandsabschätzung für teile-und-herrsche-Algorithmus:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n \log n) \quad T(2) = 1$$

Aufwand für jede Hälfte

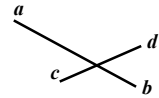
Nur 2 Punkte, dann trivial.

Sortieren aller Punkte, so daß die Trennung auch wirklich zwei gleich große Hälften machen kann. (muß nur einmalig gemacht werden, wenn alle Trennebenen jeweils parallel sind, z.B. achsenparallel)

Etwas Mathematik liefert: $T(n) = O(n \log^2 n)$

Schnitte von horizontalen und vertikalen Strecken

Schnitte von beliebigen Strecken:



z.B. Strecke $a-b$ mit Strecke $c-d$

Berechne $a+u(b-a) = c+v(d-c)$

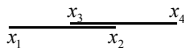
Das ist ein Gleichungssystem und hat als Lösung für u, v

- keine Lösung \Rightarrow die Strecken sind parallel
- unendl. viele Lösungen \Rightarrow die Strecken liegen übereinander
- $0 \leq u \leq 1, 0 \leq v \leq 1$ \Rightarrow die Strecken schneiden sich
- sonst \Rightarrow sie schneiden sich nicht

Schnitte von horizontalen und vertikalen Strecken

Einfacher, wenn die Strecken horizontal oder vertikal sind:

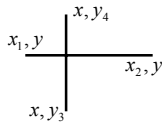
Beide horizontal:



prüfe $x_1 \leq x_3 \leq x_2$ oder $x_3 \leq x_1 \leq x_4$

Beide vertikal: entsprechend mit y statt x

verschieden ausgerichtet:



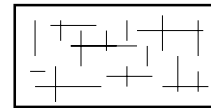
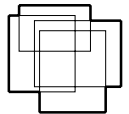
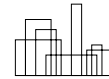
prüfe $x_1 \leq x_2$ oder $y_3 \leq y_4$

Schnitte von horizontalen und vertikalen Strecken

Schnitte vieler horizontaler und vertikaler Strecken

Motivation:

- Berechnen einer Skyline
- Berechnen der Form überlagerter Rechtecke
- VLSI-Design



Schnitte von horizontalen und vertikalen Strecken

Aufwandsabschätzung:

Seien n horizontale und m vertikale Strecken gegeben. Es können maximal $m \cdot n$ Schnittpunkte vorhanden sein.

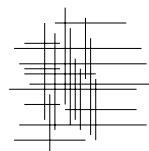
(Vereinfachende Annahme: keine Überschneidung von Strecken gleicher Ausrichtung)

Im schlimmsten Fall müssen alle Schnittpunkte ausgegeben werden. \Rightarrow Es gibt keinen Algorithmus der für alle Fälle garantiert weniger Aufwand hat $O(mn)$.

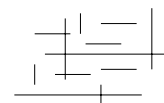
Aufwand ist aber auch höchstens $O(mn)$, weil je Streckenpaar nur eine konstante Zeit für die Schnittpunktprüfung benötigt wird.

Schnitte von horizontalen und vertikalen Strecken

Wenn der Aufwand $O(mn)$ nicht unterschritten werden kann, warum dann nicht einfach jede vertikale Strecke mit jeder horizontalen auf Schnittpunkt prüfen?



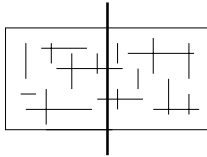
hier okay,



aber geht es hier vielleicht einfacher?

Schnitte von horizontalen und vertikalen Strecken

Überlegung: Können wir das Problem induktiv lösen?



z.B. senkrechte Gerade (|) von links nach rechts über alle
Strecken führen und alles, was komplett links liegt als
erledigt betrachten