

Proseminar

Algorithmen und Datenstrukturen
(Prof. Dr. Franz J. Brandenburg / SS2001)

Konvexe Hülle

David Schmitz

21.06.2001

Inhaltsverzeichnis

1	Einführung	3
1.1	Definition der konvexen Hülle	3
1.2	Anwendungen	4
2	Wichtige Eigenschaften	4
3	Untere Schranke	4
4	Innere Elimination	5
4.1	Beschreibung	5
4.2	Laufzeit	6
5	Wrap around	6
5.1	Beschreibung	7
5.2	Laufzeit	8
6	Graham Scan	8
6.1	Beschreibung	8
6.2	Laufzeit	10
7	Laufzeiten der bisher vorgestellten Algorithmen	10
8	Kombination von Innerer Elimination und Graham Scan	11

1 Einführung

Bei der konvexen Hülle, im englischen “convex hull” genannt, handelt es sich um eines der klassischen Probleme der algorithmischen Geometrie. Um sie geht es in der folgenden Ausarbeitung. Die konvexe Hülle beschreibt den “natürlichen Rand” einer Punktmenge $S \subset \mathbb{R}^d$, wobei im folgenden aber nur auf den Spezialfall des zweidimensionalen Raums eingegangen werden soll.

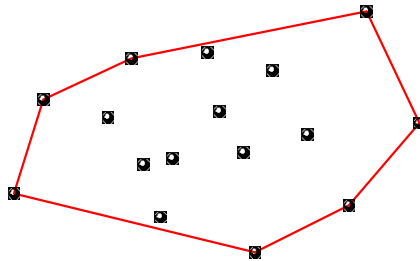


Abbildung 1: Punktmenge mit ihrer konvexen Hülle

1.1 Definition der konvexen Hülle

Sei S eine endliche Punktmenge im \mathbb{R}^2 , dann ist die Menge der Punkte aus S , die das kleinste konvexe Polygon (Vieleck) definieren, das S enthält, ihre konvexe Hülle $\text{CH}(S)$. Die Konvexitätseigenschaft für eine Menge M bedeutet, dass $\forall a, b \in M$ gilt: $\overline{ab} \subseteq M$. Angegeben wird die konvexe Hülle durch die Menge von Punkten aus S , die, wenn man sie entgegen dem Uhrzeigersinn verbindet, ein solches konvexes Polygon ergeben. Falls die konvexe Hülle eine Seite haben sollte, auf der mehr als zwei Punkte liegen, so sollen nur die beiden Punkte auf ihr mit dem größten Abstand zueinander angegeben werden, da durch sie die Seite bereits eindeutig definiert wird.

1.2 Anwendungen

Gut, nun ist klar, was konvexe Hüllen sind, doch kann man sie auch in der Praxis nutzen?

Heutzutage spielt die 3D-Programmierung eine große Rolle bei Anwendungen, wo oft das Problem auftritt, dass man wissen muss ob sich zwei Objekte berühren. Meist sind diese Objekte äußerst komplex und bestehen aus einer Vielzahl von einzelnen Punkten. Um die Anzahl der Rechenoperationen zu verringern berechnet man die konvexen Hüllen der Gegenstände, was weit weniger aufwendig ist, als jedesmal die detaillierte Figur zu bestimmen. Wenn sich die konvexen Hüllen schneiden muss diese kritische Stelle noch genauer betrachtet werden, um festzustellen ob sich beide Objekte auch bei feinerer Modellierung berühren.

2 Wichtige Eigenschaften

Um Algorithmen angeben zu können, die die konvexe Hülle berechnen, muss man einige ihrer typischen Eigenschaften kennen.

So gilt z.B. für zwei Punkte, die eine Seite der konvexen Hülle der Punktmenge S bilden, dass alle Punkte der Punktmenge entweder auf oder auf einer Seite der Geraden durch sie liegen, andernfalls könnte ein durch die Seite begrenztes Polygon nicht alle Punkte enthalten. Aus diesem Grund kann man auch sagen, dass es für jeden Punkt der konvexen Hülle eine Gerade durch ihn mit derselben Eigenschaft geben muss.

Eine weitere Eigenschaft ist, dass Punkte, die in einem von drei anderen Elementen aus S gebildeten Dreieck liegen nicht zur konvexen Hülle gehören, da wenn man einen solchen Punkt zum Dreieck hinzunimmt, das entstehende Polygon weder konvex, noch minimal ist.

Die festgestellten Eigenschaften werden wir später nutzen um konvexe Hüllen zu berechnen.

3 Untere Schranke

Es wäre nun gut zu wissen, welche Laufzeit man von den Algorithmen, die die konvexe Hülle berechnen, zu erwarten hat. Die absolute untere Schranke der Laufzeit für solche Algorithmen liegt bei $\Omega(O(n \log n))$.

Beweis Durch Reduktion auf das Sortieren von n reellen Zahlen.

Seien $x_1, \dots, x_n \in \mathbb{R}$ und Al , ein beliebiger Algorithmus zur Berechnung der konvexen Hülle, gegeben. Sei $S = \{ (x_i, x_i^2) \mid i \in \{1, \dots, n\} \}$, eine Menge, die

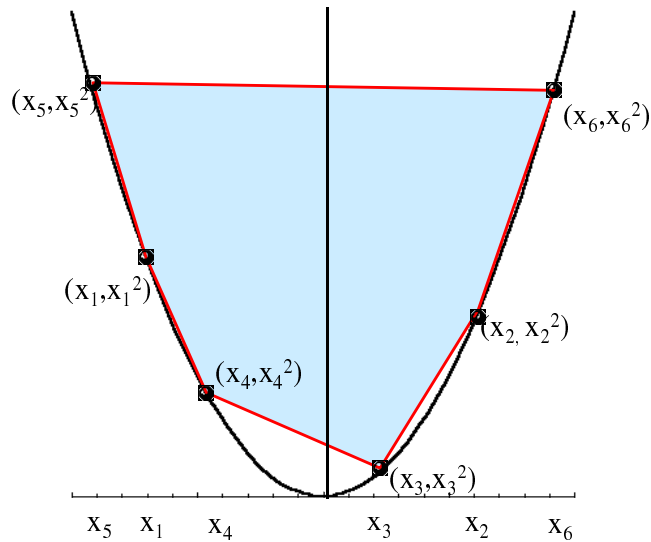


Abbildung 2: zur absoluten unteren Schranke

in $O(n)$ gebildet werden kann.

Mit $A1$ berechne man $CH(S)$, die alle Punkte aus S enthält, da sie alle auf der " x^2 -Parabel" liegen. Man durchläuft nun die Ecken der konvexen Hülle linear, beginnend bei der mit der kleinsten x -Koordinate, und erhält die x_i aufsteigend in sortierter Reihenfolge.

Der gesamte Vorgang hat also einen Zeitbedarf von $O(n) + O(x)$ ($O(n)$ zum Bilden der Paare und $O(x)$ für $A1$ zum Konstruieren von $CH(S)$), wenn $A1$ $CH(S)$ schneller als $O(n \log n)$ berechnet, könnte man durch ihn schneller sortieren als $O(n \log n)$. Doch da bekannt ist, daß die untere Schranke für Sortieralgorithmen bei $\Omega(n \log n)$ liegt, stellt dies einen Widerspruch dar.

4 Innere Elimination

4.1 Beschreibung

Der erste Algorithmus der vorgestellt werden soll, heißt "Innere Elimination". Er basiert auf der Tatsache, dass ein Punkt p nicht zu $CH(S)$ gehört, wenn er innerhalb eines von drei anderen Punkten aus S gebildeten Dreiecks liegt.

Dies nutzt man aus indem man sich die Punkte aus S sucht, für die kein solches Dreieck existiert, denn sie bilden $\text{CH}(S)$.

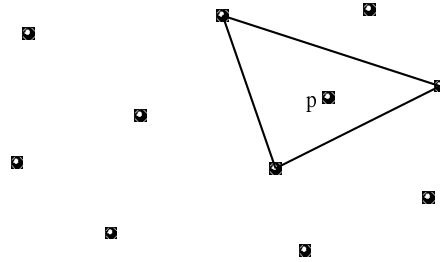


Abbildung 3: Innere Elimination

4.2 Laufzeit

In S gibt es $O(n \cdot (n - 1) \cdot (n - 2)) = O(n^3)$ durch die Punkte bestimmte Dreiecke. Da man die Prozedur für alle n Elemente wiederholen muss landet man bei einer Gesamtlaufzeit von $n \cdot O(n^3) = O(n^4)$, was noch sehr weit von $O(n \log n)$ entfernt ist.

5 Wrap around

Dieser Algorithmus, der nach seinem Erfinder auch Jarvis' march genannt wird, ist der eigentlich intuitivste: Man stelle sich die Elemente von S als Pins auf einer Pinwand vor. An einem der äußersten Punkte befestigt man eine Schnur und wickelt sie um die anderen Punkte herum (daher der Name). Ist man wieder am Ausgangspunkt angekommen berührt die Schnur genau die Punkte, die zur konvexen Hülle von S gehören.

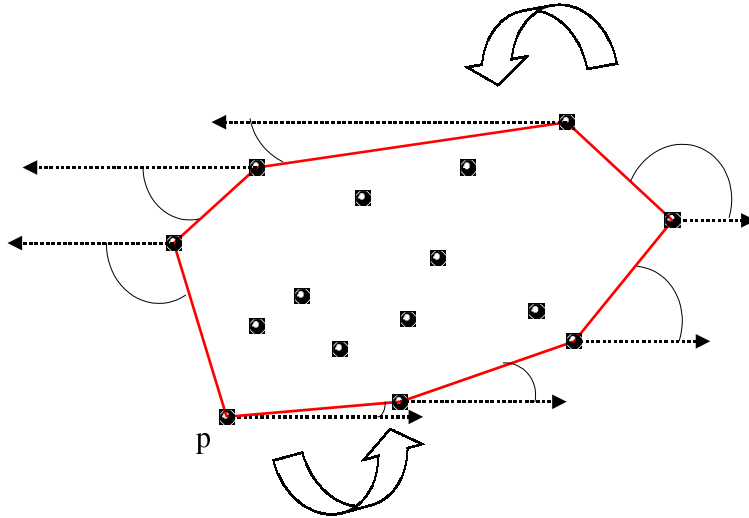


Abbildung 4: Wrap around

5.1 Beschreibung

- Suche einen Punkt p_0 , der garantiert zu $CH(S)$ gehört (z.B. den mit der kleinsten y-Koordinate)
- Suche den Punkt p_1 mit dem kleinsten Winkel zu p_0 und positiver x-Achse (dies entspricht dem "Wickeln"), und füge ihn zusammen mit p_0 in $CH(S)$ ein
- Suche den Punkt p_2 mit dem kleinsten Winkel zu p_1 und positiver x-Achse und füge ihn in $CH(S)$ ein, usw...
- Nachdem der höchste Punkt p_i in $CH(S)$ eingefügt wurde, suche den Punkt mit dem kleinsten Winkel zu p_i und negativer x-Achse, füge ihn in $CH(S)$ ein, usw...
- Fahre fort bis p_0 wieder erreicht ist

Es sei noch erwähnt, dass man sich bei mehreren Punkten mit dem gleichen Winkel auf den mit der größten Entfernung zum vorhergehenden Punkt beschränkt um nicht mehrere kollineare Punkte aufzunehmen. Desweiteren

kann auf das explizite Berechnen der Winkel verzichtet werden, da es genügt Werte zur Verfügung zu haben, die die gleiche Ordnung wie die Winkel wiedergeben. Dies kann durch das Berechnen einer Determinanten erreicht werden, was weniger aufwendig ist als trigonometrische Funktionen auszuführen.

5.2 Laufzeit

Um einen Extrempunkt von S als Anfangspunkt zu finden, benötigt man einen linearen Durchlauf, also $O(n)$. Für jeden Punkt, den man in $CH(S)$ aufnimmt muss man n Tests ausführen um den Punkt mit dem kleinsten Winkel zu finden, insgesamt also h mal n Tests, falls h die Anzahl der Knoten in $CH(S)$ ist. Insgesamt kommt man also auf eine Laufzeit von $O(n) + O(nh) = O(nh)$, wobei man den worstcase Fall (alle Punkte liegen auf der konvexen Hülle) von $O(n^2)$ nicht vergessen darf. Der Jarvis' march läuft also auf jeden Fall schneller als die vorher vorgestellte Innere Elimination, und hat außerdem den Vorteil, dass man ihn auf den mehrdimensionalen Fall ausweiten kann. Man wickelt dann einfach mit Geschenkpapier statt einem Band, d.h. mit Ebenen statt Geraden.

6 Graham Scan

Dieses 1972 von Graham vorgestellte Verfahren basiert darauf, dass alle Punkte nach ihren Winkeln zu einem Ausgangspunkt a und x-Achse sortiert werden. Wenn man sie in der erhaltenen Reihenfolge verbindet erhält man ein Polygon aus dem man nur noch die Punkte entfernen muss, die nicht zur konvexen Hülle gehören können.

Doch woher weiß man ob sie zu $CH(S)$ gehören?

Die meisten Polygone aus denen die überflüssigen Punkte noch nicht entfernt sind weisen eine "Zickzack" - Struktur auf, die den konvexen Hüllen fehlt. Betrachtet man drei aufeinanderfolgende Punkte des Polygons, von denen einer im Inneren der Punktmenge liegt, stellt man fest, daß der Winkel zwischen ihnen $<$ als 180° ist. Man kann also durch das Überprüfen der Winkel zwischen den einzelnen Punkten feststellen ob sie zu $CH(S)$ gehören.

6.1 Beschreibung

- Sortiere die Punkte aus S nach ihrem Winkel zum Ausgangspunkt a und der x-Achse (bei 2 Punkten mit gleichem Winkel wähle den mit größtem Abstand zu a) und nummeriere sie entsprechend

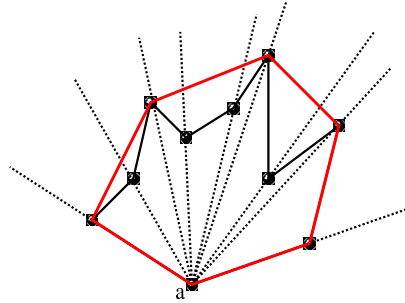


Abbildung 5: Punktmenge S , sortiert nach den Winkeln zum Startpunkt a , mit Polygon (schwarz) und fertiger konvexer Hülle (rot)

- Durchlaufe alle Punkte aus S der Reihe nach, versuche jeden Punkt in $CH(S)$ (repräsentiert vom Stack $CHtemp$) aufzunehmen und entferne vorher in $CHtemp$ aufgenommene Punkte, die nicht zur konvexen Hülle gehören können
 - Nehme p_{i+1} in $CHtemp$ auf
 - Überprüfe ob der Winkel zwischen den oberen drei Elementen auf dem Stack $(p_{i-1}, p_i, p_{i+1}) \leq$ oder $>$ als 180° ist

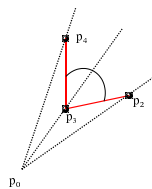


Abbildung 6: Winkel $\leq 180^\circ$

- * \leq : Entferne p_i aus $CHtemp$ und prüfe den Winkel der nun oberen drei Punkte auf dem Stack $(p_{i-2}, p_{i-1}, p_i) \dots$

(weil es \leq und nicht $<$ heißt entfernt man auch kollineare Punkte)

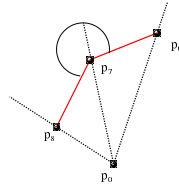


Abbildung 7: Winkel $> 180^\circ$

- * $>$: Nehme p_{i+2} auf und prüfe den Winkel $(p_i, p_{i+1}, p_{i+2}), \dots$
 - Fahre solange fort, bis p_0 wieder erreicht ist
- Gib die Punkte aus dem Stack CHtemp in umgekehrter Reihenfolge aus, dann entsprechen sie den Eckpunkten von $CH(S)$ entgegen dem Uhrzeigersinn

6.2 Laufzeit

Für das Sortieren, beispielsweise durch Mergesort, benötigt man $O(n \log n)$. Hinzu kommt noch das Einfügen in den Stack ($O(1)$), die Tests nach den Winkeln ($O(1)$), sowie gegebenenfalls das Entfernen aus dem Stack ($O(1)$). Für jeden der n Punkte aus S braucht man also, zuzüglich zum Sortieren, $3 \cdot O(1) = O(1)$, somit für alle Punkte $O(n)$. Insgesamt belaufen sich die Kosten für den Graham Scan auf $O(n \log n) + O(n) = O(n \log n)$.

7 Laufzeiten der bisher vorgestellten Algorithmen

Bei einer Punktmenge S , die aus n Elementen besteht und deren konvexe Hülle h Punkte hat, kann man mit den bisher vorgestellten Algorithmen folgende Laufzeiten erreichen:

Innere Elimination:	$O(n^4)$
Wrap around:	$O(nh)$
Graham Scan:	$O(n \log n)$

Die Innere Elimination wird man in der Praxis also nicht einsetzen, da $O(n^4)$ viel langsamer ist als die beiden Alternativen. Die Wahl zwischen Wrap around und Graham Scan fällt da schon schwerer, man sollte sich bei der Entscheidung auf das stützen was man von der “einzurahmenden” Menge weiß, denn falls $CH(S)$ weniger als $\log n$ Punkte enthält ist der Wrap around vorzuziehen, andernfalls der Graham Scan. Daß man durch geschickte Kombination mehrerer Verfahren eine noch bessere Laufzeit erwarten kann soll im abschließenden Kapitel gezeigt werden.

8 Kombination von Innerer Elimination und Graham Scan

Untersuchungen der stochastischen Geometrie haben ergeben, dass die meisten Elemente einer Punktmenge auf ein Viereck in ihrem Inneren entfallen und nach deren Elimination nur noch $O(\sqrt{n})$ Punkte übrig bleiben. Darum kann man eine sehr gute Laufzeit erreichen, wenn man vier Punkte aus der Menge nimmt, die ein möglichst großes Viereck bilden, und alle Punkte aus ihrem Inneren entfernt.

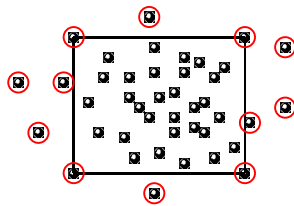


Abbildung 8: Nach dem Entfernen der inneren Punkte können nur noch die eingekreisten Punkte zu $CH(S)$ gehören

Die komplette konvexe Hülle kann man nun berechnen indem man die übrigen Punkte mit dem Graham Scan “behandelt”. Bei diesem Vorgehen

erhält man einen Algorithmus der eine durchschnittliche Komplexität von $O(n)$ hat und im schlimmsten Fall immer noch garantiert in einer zu $n \log n$ proportionalen Zeit abläuft.

Literatur

1. R. Sedgewick: Algorithmen, Addison Wesley, Bonn, 1991
2. M.T. Goodrich, R. Tamassia: Data Structures and Algorithms in Java, John Wiley&Sons, 1998
3. F.P.Preparata, M.I.Shamos: Computational Geometry, Springer, New York, 1988
4. T.H.Cormen, C.E.Leiserson, R.L.Rivest: Introduction to Algorithms, The MIT Press, Cambridge, Mass., 1990
5. T.Ottmann, P.Widmayer: Algorithmen und Datenstrukturen, Spektrum verlag Heidelberg, 1990
6. H.Engesser, V.Claus: Duden Informatik, Dudenverlag, Mannheim, 1993
7. <http://www.info1.informatik.uni-wuerzburg.de/vorlesungen/ws9899/algorithm/vorlesung.pdf>
8. <http://www.uni-paderborn.de/artur/TEACHING/GEOMETRIE99/Overview.html>