Algorithmische Geometrie
Wintersemester 2020/2021
Thomas Bläsius & Marcus Wilhelm

Karlsruher Institut für Technologie

Musterlösung zum Übungsblatt 0

Erstellt von Marcus

Aufgabe 1: Mysteriöse Meteoriten

5 Punkte

Wir suchen einen Algorithmus welcher als Eingabe den gemeinsamer Radius r von Kreisen mit Mittelpunkten $p_1, \dots, p_n$ erhält und eine zusammenhängende Fläche mit minimalem Umfang, welche alle Kreise enthält, ausgibt.

Die Idee für den Algorithmus ist, die konvexe Hülle der Kreise zu berechnen. Im Folgenden beschreiben wir, wie der Algorithmus umgesetzt werden kann, zeigen dessen Korrektheit und Argumentieren, wieso die konvexe Hülle der Kreise die zusammenhängende Fläche mit minimalem Umfang ist.

Der Algorithmus geht wie folgt vor:

- Berechne die konvexe Hülle H der Mittelpunkte $p_1, \dots, p_n$ mittels Graham Scan
- Iteriere über die Knoten von H und wandle so H in die konvexe Hülle H' der Kreise um:
  - Pro Kante (h_1, h_2) von (im Uhrzeigersinn) aufeinanderfolgenden Eckpunkten von H verschiebe diese Kante um r nach außen (d.h. orthogonal zu (h_1,h_2) nach außen)
  - Pro Eckpunkt h mit vorhergehendem Eckpunkt h' und nachfolgendem Eckpunkt h'' auf H füge außerdem den fehlen Kreisbogen in H' ein (siehe Abbildung 1). Der Kreisbogen kann z.B. durch Mittelpunkt und Anfangs- sowie Endpunkt gespeichert werden.

Die Laufzeit des Algorithmus ist in $\mathcal{O}(n \log n)$, da sie vom Graham Scan dominiert wird und anschließend einmal in Linearzeit über die konvexe Hülle iteriert.

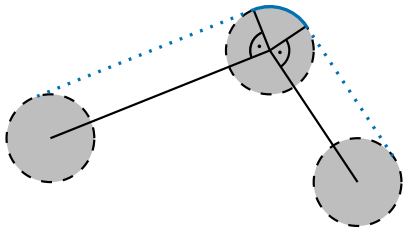

Abbildung 1: Kreisbogen in H' (blau) an einer Ecke von H.

Zur Korrektheit: H' ist offensichtlich konvex, da die Strecken tangential in die Kreisbögen übergehen und diese ebenfalls konvex sind. Da H' außerdem inklusionsminimal ist (sowohl die Kreisbögen als auch die Strecken zwischen den Kreisbögen müssen enthalten sein) berechnet der Algorithmus also tatsächlich die konvexe Hülle der Kreise.

Es bleibt zu klären, wieso die konvexe Hülle der Kreise tatsächlich die zusammenhängende Fläche mit minimalem Umfang ist. Sei A diese Fläche. Wir zeigen nun, dass A die konvexe Hülle sein muss. Wäre A nicht konvex, dann gäbe es einen Abschnitt an dem der Rand von A weiter innen verläuft als die konvexe Hülle von A. Wenn man zwischen den Endpunkten des Abschnittes (die also noch auf der konvexen Hülle von A liegen) eine gerade Strecke zieht und den Abschnitt durch diese ersetzt, findet man eine Fläche mit kleinerem Umfang, welche A enthält. Die gerade Strecke ist kürzer als der Abschnitt, da die Dreiecksungleichung gilt, d.h. für beliebige Punkte a, b, c gilt $\text{dist}(a,c) \leq \text{dist}(a,b) + \text{dist}(b,c)$ (wobei $\text{dist}(x, y)$ für den Abstand zweier Punkte x und y steht). Analog lassen sich auch Abschnitt abkürzen an denen A irgendwo über H hinaus läuft (in diesem Fall muss es eine Kante der konvexen Hülle geben, die man zu einer Geraden erweitern kann welche von A zwei mal gekreuzt wird; man kann den außen liegenden Abschnitt zwischen den Kreuzungen abkürzen).

Wie kommt man drauf?

Zuerst habe ich die Aufgabenstellung formalisiert, d.h. klar gemacht was gegeben ist und was gesucht wird. Im nächsten Schritt war intuitiv recht schnell klar, dass die konvexe Hülle der Krater / Kreise gesucht ist. Nun musste nur noch ein Algorithmus formuliert werden und bewiesen werden, dass dieser das richtige tut. Ich habe mich dazu entschieden den Algorithmus als modulare Erweiterung des Graham Scan zu formulieren, da es einfacher und formal sauberer ist einen vorhandenen Algorithmus aufzurufen und dessen Ausgabe zu modifizieren als einen „komplett neuen" Algorithmus zu beschreiben (dessen Korrektheit erstmal komplett unbewiesen wäre).

Die Anschließenden Beweise habe ich für mehr Modularität aufgeteilt in „Algorithmus berechnet H' korrekt" und „man will H' berechnen". Um den ersten Teil formell in die Hand zu bekommen habe ich mich auf sehr grundlegende geometrische Operationen beschränkt (skalierung, Schnitt mit Halbebenen), allerdings gebe ich zu dass, dass ein Schnitt mit *unendlich* vielen Halbebenen je nach Geschmack nicht die eleganteste Lösung ist. Der zweite Teil des Korrektheitsbeweises ist relativ einfach, sobald man die Idee mit der Dreiecksungleichung gefunden hat.

Aufgabe 2: Luftaufnahmen **5 Punkte**

Wir suchen einen Algorithmus der bestimmt ob ein Punkt p in, auf dem Rand, oder außerhalb eines Polygons P liegt.

Die Idee ist es von p aus in eine beliebige Richtung eine Halbgerade H zu ziehen und zu zählen, wie oft diese das Polygon schneidet: bei einer ungeraden Zahl befindet sich p innerhalb des Polygons bei einer geraden Zahl außerhalb. Die Laufzeit des Algorithmus ist linear in der Anzahl Knoten

des Polygons, da über alle Kanten iteriert wird und pro Kante eine konstante Anzahl Operationen genügt, um auf einen Schnitt mit der Halbgeraden zu prüfen.

Sonderfälle: Zuerst fangen wir den Fall ab, in dem p auf dem Rand von P liegt (jeweils durch Überprüfung ob p auf der gerade betrachteten Kante liegt).

Ein weiterer Sonderfall besteht darin, dass H das Polygon in einem der Knoten schneidet: hier darf der Schnitt nur dann gezählt werden, wenn die beiden anliegenden Kanten von einer Seite der Halbgeraden auf die andere wechselt. Auf ähnliche Art und Weise könnte es ein Problem sein, wenn eine der Kanten des Polygons genau auf H verläuft.

Wir lösen beide Probleme wie folgt: Sei x der betrachtete Schnittpunkt von H und P an einem der Knoten der Kante e und sei p' der andere Knoten von e. Wir zählen den Schnittpunkt genau dann, wenn p' auf der *rechten* Seite von H liegt. Dies löst beide Probleme, da im Falle einer Kante die auf H liegt kein Schnittpunkt gezählt wird und für die Knoten p' der an x anliegenden Kanten entweder doppelt oder gar nicht gezählt wird, falls beide links oder rechts von H liegen. Nur falls einer der Punkte links und der andere rechts liegt wird genau ein Schnitt gezählt, was korrekt ist.

Im Falle von Rundungsfehlern kann es trotzdem noch zu falschen Ausgaben kommen. Um dies zu vermeiden könnte man eine weitere Ausgabe „p liegt nah an P" einführen, welche ausgegeben wird sobald p näher als ein passendes $\varepsilon > 0$ an einer Kante von P liegt.

Wie kommt man drauf?

Der Algorithmus kann (hoffentlich) recht einfach gefunden werden, wenn man sich nicht zu sehr von konvexen Hüllen, Triangulierungen des Polygons oder anderen möglichen Strukturen Ablenken lässt ;-).