Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Geometrie
Sommersemester 2012
Martin Nöllenburg/Andreas Gemsa

Übungsblatt 6 - Point Location

1 Keine Vorberechnung

In der Vorlesung wurde ein Algorithmus besprochen, der das *poin location* Problem mit Hilfe eines Vorberechnungsschritts löst. Im Folgenden werden wir annehmen, dass sowohl die Polygon-Unterteilungen als auch der Punkt für die Punktanfrage gleichzeitig mitgeteilt werden. Deshalb lohnt sich der Vorberechnungsschritt nicht mehr. [**Gilt auch für Aufgaben 2 und 3**].

Sei P ein einfaches Polygon bestehend aus n Knoten und sei q der Anfrage-Punkt. Es folgt eine schriftliche Beschreibung eines Algorithmus' der bestimmt ob q im Inneren von P liegt:
Sei $\rho := \{(q_x + \lambda, q_y) : \lambda > 0\}$ (horizontale Kante die durch die Punkt q verläuft). Ermittle für jede Kante e des Polygons P ob sie ρ schneidet. Ist die Gesamtanzahl der Kanten aus P die ρ schneiden ungerade, so liegt q im Inneren von P.

a) Zeige die Korrektheit des Algorithmus.

b) Erkläre wie man mit degenierten Fällen umgeht (Ein Beispiel für einen degenierten Fall ist, dass ρ einen Endpunkt einer Kante schneidet.)

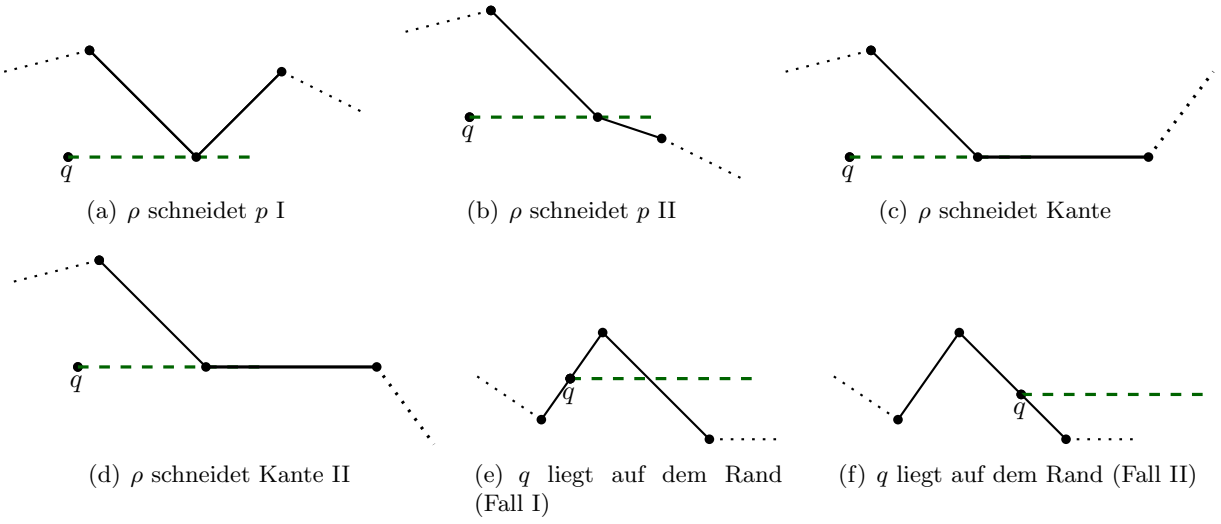
c) Bestimme die Laufzeit dieses Verfahrens.

Lösung:

Zu a) Wir wissen: Kante eines Polygons grenzt immer an innere und äußere Facette. Seien q und r Punkte in der Ebene und sei P ein Polygon. Wenn die Verbindungslinie qr das Polygon P eine gerade Anzahl schneidet, dann bedeutet es, dass es eine gerade Anzahl an wechseln von innerer zu äuerer Facette gegeben hat. Da es keine Kante von P gibt, die die zu beiden Seiten eine innere (oder äußere) Facette begrenzt muss r in derselben Facette wie q liegen. Analoge Argumentation: Wenn qr P eine ungerade Anzahl mal schneidet, dann liegen q und r in verschiedenen Facetten. Für q im inneren von P und r mit gleicher y-Koordinate wie q und genügend großer x-Koordinate folgt die Aussage.

zu b): Fünf degenerierte Fälle. Siehe Abbildung 1. Nur b), d) und e) wie Schnitt mit einer Kante betrachten.

Zu c): Schnitt von allen Kanten des Polygons mit ρ berechnen: $\mathcal{O}(n)$



(a) ρ schneidet p I (b) ρ schneidet p II (c) ρ schneidet Kante

(d) ρ schneidet Kante II (e) q liegt auf dem Rand (Fall I) (f) q liegt auf dem Rand (Fall II)

Abbildung 1: Halbgerade ρ ist die grün gestrichelte Linie. Kanten des Polygons sind schwarz

2 Spezielle Polygone

Sei P ein *konvexes* Polygon bestehend aus n Knoten. Die Knoten sind in sortierten Reihenfolge (bzgl. des Rands des Polygons) gegeben.

a) Zeige, dass es man in $\mathcal{O}(\log n)$ Zeit bestimmen kann, ob ein gegebener Punkt q innerhalb von P liegt.

b) Kann man dein Resultat aus a) für den Fall verallgemeinern, dass P kein konvexes sondern ein y-monotones Polygon ist? [Erläutern]

Lösung:
Zu a): Grundlegende Idee: Unterteile Suchraum wiederholt in zwei Hälften. Betrachte nur eine Hälfte.

Wähle beliebigen Punkt p_i und zusätzlich Punkt $p_j, j = (i + n/2) \mod n$ von P. Konstruiere Kante ℓ durch p_i und p_j. Teste ob q links oder rechts von ℓ liegt. Wiederhole bis entweder klar ist, dass q außerhalb von der konvexen Hülle von P liegt (Knoten immer rechts oder immer links von Kante), oder bis Kanten $e_1 = p_j p_g$ und $e_2 = p_j p_{g+1}$ bekannt sind für die gilt, dass q links von e_1 und rechts von e_2 (oder umgekehrt) liegt. Dann fehlt nur ein letzter Test: Liegt q links (rechts) von $p_g p_{g+1}$.

Laufzeit: $\mathcal{O}(\log n)$.

Zu b): Geht leider nicht mit diesem Ansatz.

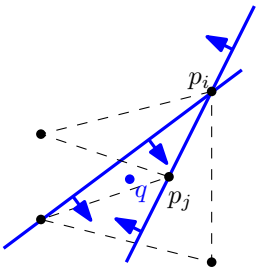


Abbildung 2: Gegenbeispiel für Aufgabe 2b)

3 Spezielle Polygone II

Ein Polygon P heißt *sternförmig*, wenn es einen Punkt p im Inneren von P gibt, so dass für jeden anderen Punkt q im Inneren von P gilt, dass die Verbindungsstrecke pq vollständig in P enthalten ist. Nehme im Folgenden an, dass solch ein Punkt p zusammen mit einem sternförmigen Polygon P gegeben ist.

a) Zeige, dass man in $\mathcal{O}(\log n)$ Zeit bestimmen kann ob ein Punkt q im Inneren von P liegt oder nicht.

b) Was ist, wenn der Punkt p nicht gegeben ist? Kann man dein Verfahren aus a) so anpassen, dass es auch ohne den Punkt p funktioniert? [Erläutern]

Lösung:
Zu a): Grundlegende Idee (binäre Suche) wie in 2a). Betrachte Kante von p_iq und p_jq (für p_i, p_j s. Aufgabe 2). Links/rechts-Test wie bei Aufgabe 2, aber da Kantensteigungen nicht gleich mehr Sorgfalt bei der Auswahl der korrekten Hälfte.

Zu b): Wieder ähnliches Problem wie in 2b).

4 Das Ray-Shooting Problem

Das *ray shooting* Problem tritt häufig beim Rendern von computergenerierten 3D-Grafiken auf. Die 2D Variante lautet wie folgt: Verwalte eine Menge S bestehend aus n sich nicht schneidenden Streckensegmenten so, dass man schnell eine Anfrage vom Typ: "Gegegeben ein query-Strahl ρ— eine Halbgerade die in einem Punkt startet. Finde des erste Segment aus S dass von ρ geschnitten wird."

Im Folgenden betrachten wir nur das *vertical* ray shooting Problem bei dem der query-Strahl nur vertikal "geschossen" wird. Das bedeutet insbesondere, dass für solch eine Anfrage nur der Startpunkt definiert werden muss.

a) Gebe eine Datenstruktur für das vertical ray shooting Problem an. Gebe eine möglichst scharfe Schranke für die worst-case Laufzeit und den worst-case Speicherplatzverbrauch an.

b) Kann man den Ansatz aus a) so modifizieren, dass er auch funktioniert, wenn die Streckensegmente sich schneiden dürfen?

Lösung:
Zu a): Verwende Trapezzerlegung. Wie müssen nur das die obere Begrenzung des Trapez finden in dem der Startpunkt liegt. Deshalb speichern wir bei der Konstruktion zu jedem Trapez die obere Begrenzung.

Laufzeiten und Speicherplatzverbrauch wie gehabt.

Zu b): In einem Durchlauf Schnitte bestimmen und Pseudoknoten bei Schnitten einfügen. Im schlimmsten Fall $\mathcal{O}(n^2)$ Schnitte und damit insgesamt im worst-case $\mathcal{O}(n^2)$ Knoten. Verfahren wie in a) anwenden.

Sei k Anzahl der Schnitte.

Laufzeit: Erwartet query-Zeit: $\mathcal{O}(\log(n+k))$, erwartete Konstruktionszeit $\mathcal{O}((n+k)\log(n+k))$.

Speicherplatz: $\mathcal{O}(n+k)$