LABORPROTOKOLL

Im Studiengang AI Engineering - Advanced Programming

Convex Hull Algorithms (Gift-wrapping, Quick Hull)

Ausgeführt von: Hannah Kohlhofer, MSc.
Elias Marcon, MSc.
Ing. Fabian Steiner, BSc.
Personenkennzeichen: 2310585014
2310585020
2310585027

PICs/fhtw_cover.png

BegutachterIn: FH-Prof. Dipl.-Ing. Alexander Nimmervoll

Wien, den 29. September 2023

Inhaltsverzeichnis

1 Die Algorithmen

Um die Konvexe Hülle einer Punktemenge zu bestimmen bedarf es verschiedener Algorithmen. Für dieses Projekt implementieren wir zwei verschieden Algorithmen - der Quickhull und der Giftwrapping Algorithmus. Beide Algorithmen haben das Ziel die Konvexe Hülle einer Punktmenge - also das kleinste konvexe Polygon zu bestimmen, wobei hier kein Punkt außerhalb dieser Hülle liegen darf. In anderen Worten: Jeder Punkt der Punktemenge liegt entweder auif der konvexen Hülle oder wird von dieser eingesperrt / umschlossen.

1.1 Quickhull

Der Quick Hull Algorithmus ist dazu da aus einer Punktemenge eine Convexe Hülle zu bilden. Die Grundidee hierbei ist das "Divide and ConquerPrinzip. Das bedeutet das Gesamtproblem wird zuerst in mehrere Teilprobleme zerlegt und diese werden dann für sich gelöst und abschließen wieder zusammengesetzt um eine Gesamtlösung zu finden.

1.1.1 Pseudocode

```
Funktion QuickHull(S)
{
    // Bestimmt die konvexe Huelle der Menge S
    Convex Hull := {}
    A := Punkt ganz links
    B := Punkt ganz rechts
    Fuege die Punkte A und B der konvexen Huelle hinzu
    // Die Gerade AB teilt die verbleibenden n - 2 Punkte in die Teilmengen
       S1 und S2
    S1 := Menge der Punkte in S, die auf der rechten Seite von AB sind
    S2 := Menge der Punkte in S, die auf der linken Seite von AB sind
    FindHull(S1, A, B)
    FindHull(S2, B, A)
    Ausgabe: Convex Hull
}

Funktion FindHull(Sk, P, Q)
{
    // Bestimmt die Punkte auf der konvexen Huelle der Menge Sk, die auf der
         rechten Seite der Geraden PQ sind
```

```
    Wenn Sk keine Punkte enthaelt dann
        Ausgabe: Convex Hull
    C := Der Punkt in Sk, der den groessten Abstand von der Geraden PQ hat
    Fuege den Punkt C in die konvexe Huelle zwischen den Punkten P und Q ein
    // Die drei Punkte P, Q und C teilen die verbliebenen Punkte von Sk in die Teilmengen S0, S1 und S2
    S0 := Die Punkte innerhalb des Dreiecks PCQ
    S1 := Die Punkte auf der rechten Seite der Geraden PC
    S2 := Die Punkte auf der rechten Seite der Geraden CQ
    FindHull(S1, P, C)
    FindHull(S2, C, Q)
    Ausgabe: Convex Hull
}
```

Listing 1.1: Quick Hull Pseudocode [**?**]

Hierbei ist gut zu sehen, wie der Algorithmus das Gesamtproblem aufteilt: beim ersten Aufruf wird die Funktion Quick Hull aufgerufen und diese teilt die Menge der Punkte in der Hälfte auf und ruft damit jeweils einmal die Funktion FindHull auf, welche die Punkwieder aufteilt und sich mit den zwei Hälften selbst erneut aufruft.

1.1.2 Beschreibung des Alogrithmus

Der Algorithmus beginnt damit, dass er den linkesten und den rechtesten Punkt auswählt und sie zur konvexen Hülle hinzufügt. Dadruch ensteht eine Linie (im Pseudocode bezeichnet als AB). Nun wird für die Punktemenge auf jeder Seite der Hülle weiterbetrachtet. Man sucht hier nun jeweils nach dem am weitesten entfernten punkt C zur Geraden. Die Auwahl des am weitesten Entfernten Punktes führt hier zu den Besten Ergebnissen wie in The Quickhull Algorithm for Convex Hulls"[**?**] beschrieben wird. Man verbindet nun die Gerade AB mit dem Punkt C und erhält daurch ein Dreieck ($S0$) in dem sich (wahrscheinlich) Punkte befinden, die nicht mehr weiter betrachtet werden müssen (da sie schon innerhalb der Hülle liegen). Wenn sich außerhalb des dreiecks linkerseits zu AC und rechterseits zu BC Punkte befinden, wird für diese der Schritt FindHull"wiederholt, bis sich alle Punkte in der Hülle befinden.

1.1.3 Aufwandsabschätzungen und Testfälle

Laut Barber, Dobkin und Huhdanpaa [**?**] liegt die obererste schranke bei $\mathcal{O}(n * log(n))$, wenn die Pivotpunkte gut gewählt werden (zum Beipsiel der am weitesten entfernte Punkt). Werden sie zufällig gewählt so liegt die oberste Schranke im 2D bei $\mathcal{O}(n^2)$.

1.2 Giftwrapping

Der Giftwrapping Algorithmus, dient, wie zuvor beschrieben, ebenfalls zur Berechnung der Konvexen Hülle (CH) einer Punktemenge (Q). Im Zwei-Dimensionalen Fall wird der Giftwrapping Algorithmus auch als Jarvis-March bezeiuchnet, benannt nach seinem ËntdeckerR. A. Jarvis im Jahr 1973 (siehe [**?**]).

1.2.1 Pseudocode

Im Folgenden Code geltend die Bezeichnungen P für den jeweiligen Startpunkt und S für die gesamte Punktemenge.

```
startpunkt = Punkt mit kleinster Ordinate
i = 0
wiederhole
    P[i] = startpunkt
    endpunkt = S[0]
    wenn startpunkt == endpunkt
      endpunkt = S[1]
    fuer j von 1 bis |S|
      ist (endpunkt == startpunkt) oder (S[j] links von der Geraden
          zwischen startpunkt und endpunkt)
        endpunkt = S[j]
    startpunkt = endpunkt
    i++
bis endpunkt == P[0]
```

Listing 1.2: Giftwrapping Pseudocode [**?**]

1.2.2 Beschreibung des Alogrithmus

Der Giftwrapping Algorithmus startet beim untersten Punkt der Punktemenge p (kleinster y-Wert). Sollten mehrere Punkte den gleichen y-Wert aufweisen, so wird jener Punkt gewählt, der zusätzlich den kleinsten x-Wert aufweißt. Dann werden iterativ alle anderen Punkte mittels des Winkels von p sortiert. Der Punkt mit dem kleinsten Winkel (a) liegt demzufolge auf der Konvexen Hülle und kann mit dem Ausgangspunkt verbunden werde. Infolgedessen steht Punkt a als neuer Ausgangspunkt zur Verfügung und es werden erneut alle Winkel von diesem Punkt aus berechnet. So wiederholt sich der Algorithmus bis am Ende die Konvexe Hülle geformt werden kann.

Die Bezeichnung "Winkel berechnen"kann hierbei etwas irreführend sein. Um den Algorithmus zu vereinfachen werden nicht tatsächliche Wiunkelwerte berechnet, sondern es werden einfache gerade Linien vom Ausgangspunbkt P zu allen anderen Punkten gezogen. Es wird somit für jeden Punkt bzw. jede gerade Überprüft, ob sich links von dieser Gerade noch eine weitere befindet. Sollte dies der Fall sein, wird jeweils die linkeste Geradefür die Konvexe Hülle verwendet, solange keine neue gefunden wird, die noch weiter links ist. Der Algorithmus arbeitet sich also entgegen dem Uhrzeigersinn voran, bis die gesamt Konvexe Hülle gebildet ist.

1.2.3 Aufwandsabschätzungen und Testfälle

Laut Jarvis ([**?**]) hat der Giftwrapping Algorithmus einen Aufwand von $\mathcal{O}(n(h+1))$, für n Punkt in der Punktmenge und $n \leq h$ auf der Hülle liegende Punkte. Andere Quellen berichten von einer Lauzeit von $\mathcal{O}(nh)$. In den meisten Fällen, wird jedoch nicht einmal dieser Aufwand benötigt werde, da der Algorithmus vereinfacht werden kann sowohl durch den zuvor beschriebenen Austausch der Winkelberechnungen mit simplen checks auf welcher Seite ein Punkt liegt, als auch durch den gezielten Auschluss von Punkten für alle weiteren Berechnungen. Punkt werden somit ausgeschossen, wenn sie entweder bereits als auf der Konvexen Hülle liegend bestimmt wurden, oder sie in dem Bereich liegen, der vom ersten zum letzten Punkt auf der Hülle eingegrenzt wird. Diese Vereinfachung ist im Pseudocode nicht zu sehen, kann jedoch ganz einfach implementiert werden.

Die Laufzeit des Giftwrapping Algorithmus hängt von der Größe des Outputs ab, was ihn zu einem sogenannten öutput sensitiveÄlgorithmus macht. Wie im vorherigen Absatz bereits erwähnt hängt der Algorithmus linear von der Anzahl der Punkte auf der Hülle ab. Demnach kann die Laufzeit nur dann, wie bei anderen Algorithmen, gleich schnell oder schneller als $\mathcal{O}(nlog(n))$ sein, wenn die Anzahl der auf der Hülle liegenden Punkte h kleiner ist als $nlog(n)$. Der Idealfall, für die kürzest Mögliche Laufzeit würde eintreten, wenn immer direkt der linkeste Punkt gefunden wird. Der "Best Case"hat somit eine Laufzeit von $\mathcal{O}(n)$. Der "Worst Case"hingegen tritt ein, wenn ale Punkte auf einem Kreis liegen. In diesem Fall ist $h = n$ und somit beträgt die Laufzeit $\mathcal{O}(n^2)$, da jeder Punkt mit jedem Punkt verglichen werden muss.

Abbildungsverzeichnis

Abkürzungsverzeichnis

ABC Alphabet

WWW world wide web

ROFL Rolling on floor laughing