Algorithmische Geometrie
Wintersemester 2020/2021
Thomas Bläsius & Marcus Wilhelm

Karlsruher Institut für Technologie

Musterlösung zum Übungsblatt 2

Erstellt von Dominik Bez und Wendy Yi

Aufgabe 1: Triangulierung

6 Punkte

Teilaufgabe (a) Es bezeichne $t(n)$ ($n \geq 3$) die Anzahl der Dreiecke einer beliebigen Triangulierung eines Polygons mit n Ecken.

Behauptung 1. *Es gilt für alle $n \geq 3$: $t(n) = n - 2$.*

Beweis. Induktionsanfang: $t(3) = 1$

Induktionsschritt: Sei $t(m) = m-2$ für alle $m < n$ und P ein Polygon mit n Punkten. Laut Vorlesung lässt sich das Polygon in zwei Teilpolygone P_1, P_2 mit je n_1 beziehungsweise n_2 Ecken zerlegen. Dabei gilt $n_1 + n_2 = n + 2$, da zwei Ecken zu beiden Polygonen gehören und damit doppelt gezählt werden. Die Anzahl der Dreiecke in P entspricht der Summe der Dreiecke in P_1 und P_2. Damit gilt

$$t(n) = t(n_1) + t(n_2) = n_1 - 2 + n_2 - 2 = n_1 + n_2 - 4 = n + 2 - 4 = n - 2\,.$$

□

Teilaufgabe (b) Wir betrachten die Ecke v eines Lochs, die die größte y-Koordinate aller Ecken aller Löcher hat. Sei e die Kante des umgebenen Polygons, die direkt über v liegt. Wenn möglich, verbinden wir v mit der höchsten Ecke ℓ von e. Ohne Beschränkung der Allgemeinheit sei ℓ die linke Ecke von e. Falls das nicht geht, verbinden wir v mit der rechtesten Ecke u des umgebenen Polygons, die die folgenden Eigenschaften erfüllt:

1. Die y-Koordinate von u ist größer als die y-Koordinate von v.
2. Die x-Koordinate von u ist kleiner als die x-Koordinate von v.

Solch eine Ecke existiert, da v sonst mit ℓ verbunden werden könnte. Zwischen v und u kann keine Kante sein, da e direkt über v liegt und u der rechteste solche Punkt ist.

Wir betrachten das umgebene Polygon und das Loch von v als ein neues, verschmolzenes Polygon. Da dieses Polygon ein Loch weniger besitzt, können wir Induktion anwenden. Streng genommen ist es kein Polygon, da v und ℓ beziehungsweise u zweimal in der Kantenfolge vorkommen. Wir können uns aber die beiden Ecken dupliziert mit minimalem Abstand vorstellen.

Teilaufgabe (c) Sei P ein trianguliertes Polygon mit n Ecken (inklusive der Ecken der Löcher) und k Löchern, wobei jedes Loch i von n_i Ecken umrandet ist ($1 \leq i \leq k$). Sei n_0 die Anzahl der äußeren Ecken. Dann hat der zugehörige Graph G n Knoten, wovon n_0 Knoten zur äußeren Facette inzident sind. Wir fügen einen Knoten in die äußere Facette ein und verbinden diesen zu den n_0 äußeren Knoten. Damit hat der resultierende Graph G' $n+1$ Knoten und $|F(G)| + n_0 - 1$ Facetten und ist vollständig trianguliert. Also gilt mit der Eulerformel $|V(G)| - |E(G)| + |F(G)| = 2$ und dem Hinweis

$$2 = |V(G')| - |E(G')| + |F(G')| = |V(G')| - 3|V(G')| + 6 + |F(G')| \,.$$

Daraus folgt

$$|F(G)| = |F(G')| - n_0 + 1 = 2|V(G')| - 4 - n_0 + 1 = 2n - 1 - n_0 \,.$$

Dabei ist eine Facette die äußere Facette und muss noch abgezogen werden. Außerdem müssen wir noch die Facetten in den Löchern abziehen. Aus Aufgabe 1a) wissen wir, dass jedes Loch i in $n_i - 2$ Dreiecke unterteilt wird. Damit ergibt sich für die Anzahl der Dreiecke des triangulierten Polygons P

$$|F(G)| - 1 - \sum_{i=1}^{k}(n_i - 2) = 2n - 2 - n_0 - (n - n_0) + 2k = n + 2(k-1) \,.$$

Wie kommt man drauf?

Die **Teilaufgabe (a)** ist recht offensichtlich, wenn man an Induktion denkt. Für die **Teilaufgabe (b)** sollte man sich daran erinnern, wie wir das für normale Polygone in der Vorlesung gezeigt haben. Dann muss man sich aber noch überlegen, wie man Induktion anwenden kann. Eine Kante einfügen alleine reicht nicht aus, da das Ergebnis kein Polygon mehr ist. Für die **Teilaufgabe (c)** ist es hilfreich, das geometrische Problem in ein graphentheoretisches Problem zu übersetzen und vorhandenes Wissen wie die Eulerformel einzusetzen. Diesen Ansatz könnte man übrigens auch bei der **Teilaufgabe (a)** verwenden.

Aufgabe 2: y-monotone Triangulierung **5 Punkte**

Sei P ein y-monotones Polygon mit n Ecken $p_1, \ldots, p_n$, die aufsteigend nach ihrer y-Koordinate sortiert sind. Punkte auf der linken Seite des obersten Punktes werden vor Punkten auf der rechten Seite mit der gleichen y-Koordinate einsortiert. Punkte, die die gleiche y-Koordinate haben und sich auf der gleichen Seite befinden werden nach der Kantenfolge des Polygons sortiert.

Eine solche Sortierung bekommen wir, indem wir die Punkte in Punkte der Kantenfolge auf der linken Seite des obersten Punkten und Punkte der Kantenfolge auf der rechten Seite des obersten Punktes aufteilen. Da P y-monoton ist, sind die Punkte auf beiden Seiten bereits sortiert und können in Linearzeit zu einer sortierten Liste gemergt werden.

Wir gehen diese sortierte Liste durch und speichern uns dabei zu jedem Zeitpunkt die Punkte, die bereits gesehen wurden, aber noch für die Triangulierung relevant sind, in einer Liste L. Der Teil

des Polygons, der sich oberhalb der Punkte in L befindet, ist bereits trianguliert. Die Punkte in L bilden eine Kantenfolge des Polygons, die entweder aus zwei Knoten besteht oder eine konkave Kantenfolge bildet. Ist L eine Deque, können wir in $\mathcal{O}(1)$ auf das erste bzw. letzte Element zugreifen.

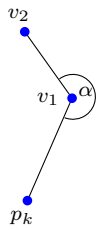
1. Füge die p_1 und p_2 in L ein.
2. Wir iterieren über die Punkte $p_3, \ldots, p_n$. Wenn ein Punkt p_k betrachtet wird, ist dieser entweder zum ersten oder letzten Punkt in L adjazent. Wir fügen p_k am Anfang oder am Ende von L hinzu, sodass die Punkte in L wieder eine Kantenfolge bilden.
3. Seien q_1, q_2 Punkte in L, sodass (p_k, q_1, q_2) eine Kantenfolge des Polygons mit zwei Kanten induzieren. Je nachdem, ob wir p_k am Anfang oder am Ende von L hinzugefügt haben, sind q_1 und q_2 also die direkten Vorgänger oder Nachfolger von p_k in L. Den Winkel zwischen den beiden Kanten, der im Inneren des Polygons liegt, bezeichnen wir als α. Wir unterscheiden die folgenden Fälle für α, die auch in Abbildung 1 zu sehen sind:
   (a) Ist $\alpha \geq 180°$, machen wir in Schritt 2 mit dem nächsten Punkt weiter.
   (b) Ist $\alpha < 180°$ und sind die y-Koordinaten von p_k und q_2 verschieden, fügen wir die dritte, noch fehlende Kante zwischen p_k und q_2 ein. Den Punkt q_1 löschen wir aus L. Dadurch induzieren die Punkte in L wieder eine Kantenfolge.
   (c) Ist $\alpha < 180°$ und sind die y-Koordinaten von p_k und q_2 gleich, haben wir einen Sonderfall. Falls es keinen Punkt mit der gleichen y-Koordinate zwischen p_k und q_2 gibt, behandeln wir den Fall wie in b). Ansonsten fügen wir keine Kante ein und machen in Schritt 2 mit dem nächsten Punkt weiter.

   Schritt 3 wiederholen wir mit p_k, bis $\alpha \geq 180°$, bis L nur zwei Punkte enthält.
4. Nach Betrachtung des letzen Punktes p_n enthält L genau drei Punkte, die ein Dreieck induzieren.

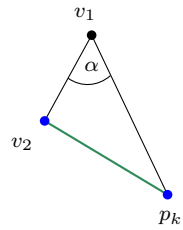
Der Algorithmus läuft in Linearzeit, da wir jede Ecke nur einmal betrachten, zu L hinzufügen und wieder löschen.

Für die Korrektheit zeigen wir die folgenden Eigenschaften:

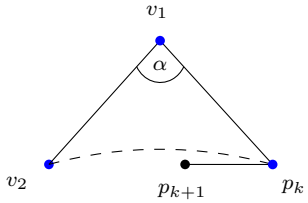
- Eine eingefügte Kante befindet sich nie komplett außerhalb des Polygons, da wir nur Kanten hinzufügen, wenn die beiden anderen Kanten, die betrachtet werden, eine konvexe Kurve bilden bezüglich des Polygons.
- Außerdem schneidet keine andere Kante die eingefügte Kante, da wir dann einen Teil von den zu betrachtenden Kanten schon trianguliert hätten. Dann gäbe es Punkte in L, die für die weitere Triangulierung irrelevant wären.



(a) Der Winkel α ist größer als 180°. Wir fügen keine Kante ein und betrachten den nächsten Punkt.



(b) Der Winkel α ist kleiner als 180°. Wir fügen die grüne Kante ein und löschen v_1 aus L.



(c) Der Winkel α ist kleiner als 180°. Trotzdem können wir die gestrichelte Kante von p_k zu v_2 nicht einfügen, da sie p_{k+1} enthalten würde. Wir fügen keine Kante ein und betrachten den nächsten Punkt.

Abbildung 1: Die drei Fälle, die in Schritt 2 auftauchen können bei Betrachtung von p_k. Punkte sind blau, wenn sie nach Betrachtung von p_k in L sind.

- Die Liste L enthält am Ende des Algorithmus genau drei Knoten, die ein Dreieck induzieren. Sonst wäre in L am Ende eine konkave Kantenfolge bezüglich des Polygons. Da die Kantenfolge im Polygon aber nicht nur konkav sein kann und irgendwann eine Kurve in die andere Richtung machen muss, ist das nicht möglich. Also werden alle Knoten des Polygons abgearbeitet.
- Wenn wir das Ergebnis von Aufgabe 1a) in die Eulerformel ($n - m + f = 2$) einsetzen, erhalten wir $m = 2n - 3$ als die Anzahl der Kanten eines triangulierten Polygons. Wir zeigen, dass das nach Ausführung des Algorithmus die Anzahl der Kanten des Polygons mit den hinzugefügten Kanten ist.

  Für jede Kante, die wir hinzufügen, löschen wir genau einen Punkt aus L. Da jeder Punkt irgendwann in L ist und am Ende genau drei Knoten nicht aus L gelöscht werden, fügen wir insgesamt $n - 3$ Kanten hinzu. Insgesamt hat das Ergebnis also $n + (n - 3) = 2n - 3$ Kanten.

Die Ausgabe mit den hinzugefügten Kanten hat also maximal viele Kanten, die sich im Inneren des Polygons befinden und keine andere Kante schneiden. Damit ist gezeigt, dass das Polygon trianguliert ist.

Wie kommt man drauf?

Wir wollen greedy Kanten einfügen, so lange es möglich ist. Dabei kommt es eigentlich nur darauf an, ob wir gerade einen Punkt in einer konkaven Kantenfolge betrachten oder in einer konvexen.

Aufgabe 3: 2D-LP vervollständigen **4 Punkte**

Im Folgenden betrachten wir nur Eingaben, für die eine Lösung existiert. Existiert keine Lösung, ist die Ausgabe des Algorithmus undefiniert. Die Maximierungsfunktion sei gegeben als ein Vektor $\vec{m}$. Die Nebenbedingungen seien gegeben durch die Normalen $\vec{u}_1, \dots, \vec{u}_n$ der begrenzenden Geraden der Halbebenen $g_1, \dots, g_n$. Diese zeigen von der jeweiligen Halbebene weg in Richtung ungültiger Lösungen, da die Nebenbedingungen $\leq$-Ungleichungen sind.

Wir suchen zwei Normalen $\vec{u}_i$ und $\vec{u}_j$ (dabei ist $i = j$ explizit erlaubt), die die folgenden Bedingungen erfüllen:

- Das Skalarprodukt von $\vec{u}_i$ und $\vec{m}$ ist positiv ($\vec{u}_i$ zeigt in eine ähnliche Richtung wie $\vec{m}$): $\vec{u}_i \cdot \vec{m} > 0$.
- Das Skalarprodukt von $\vec{u}_j$ und $\vec{m}$ ist nicht negativ ($\vec{u}_j$ zeigt in eine ähliche Richtung wie oder ist orthogonal zu $\vec{m}$): $\vec{u}_j \cdot \vec{m} \geq 0$.
- Der Vektor $\vec{u}_i$ zeigt relativ zu $\vec{m}$ nach links oder in die gleiche Richtung und $\vec{u}_j$ nach rechts oder in die gleiche Richtung, oder andersherum.

Es kann in $\mathcal{O}(n)$ Zeit für alle Normalenvektoren $\vec{u}_1, \dots, \vec{u}_n$ entschieden werden, ob sie ein möglicher Kandidat für $\vec{u}_i$ und/oder $\vec{u}_j$ sind und in welche Richtung relativ zu $\vec{m}$ sie zeigen. Für $\vec{u}_i$ merken wir uns nur je einen Kandidaten für links und rechts, falls ein solcher existiert. Zum Schluss müssen wir nur für die beiden $\vec{u}_i$-Kandidaten nach $\vec{u}_j$-Kandidaten suchen, die in die andere Richtung zeigen. Wenn kein passendes Paar (i,j) gefunden wurde, ist das lineare Programm unbeschränkt. Ansonsten wird (i,j) ausgegeben.

Der Algorithmus aus der Vorlesung kann vervollständigt werden, indem zuerst überprüft wird, ob überhaupt eine gültige Lösung existiert. Danach wird unser neuer Algorithmus ausgeführt. Wenn das LP beschränkt ist, erhalten wir die zwei Halbebenen $\vec{g}_i$ und $\vec{g}_j$. Wenn $i = j$ gilt, dann beschränkt g_i bereits den Lösungsraum. Wähle $h_1 = g_i$ im Algorithmus aus der Vorlesung und für v_1 einen beliebigen Punkt auf der begrenzenden Gerade von h_1, da jeder davon optimal in h_1 ist. Wenn $i \neq j$, dann wähle $h_1 = g_i$, $h_2 = g_j$ und für v_2 den Schnittpunkt der begrenzenden Geraden von h_1 und h_2. Für die restlichen Halbebenen wird wieder eine zufällige Reihenfolge gewählt.

Wie kommt man drauf?

Der Lösungsraum ist genau dann beschränkt, wenn es für beide Komponenten des Maximierungsvektors $\vec{m}$ eine Halbebene gibt, die die Komponente beschränkt. Dazu muss man sich überlegen, wie man das formalisieren kann. Dazu bieten sich die Normalenvektoren an. Dabei darf man den Spezialfall nicht vergessen, bei dem eine einzige Halbebene orthogonal zu $\vec{m}$ ist und die Menge möglicher Lösungen beschränkt.

Aufgabe 4: Skihütten **5 Punkte**

Sei P die Menge der Polygone und S die Menge der Skihütten. Wir wollen die Anzahl der Punkte in S zählen, die sich in genau x Polygonen befinden. Dazu modifizieren wir den Sweep-Line-Algorithmus aus der Vorlesung. Die Ecken aller Polygone in P und die Punkte in S werden absteigend nach ihrer y-Koordinate (bei gleicher y-Koordinate nach ihrer x-Koordinate) sortiert. Zu jeder Ecke eines Polygons gehören zwei inzidente Kanten, von denen die Ecke jeweils entweder ein Startpunkt oder ein Endpunkt ist. Der Startpunkt einer Kante ist dabei die Ecke, die in der Sortierung zuerst kommt. Die Sweep-Line ℓ iteriert über alle Ecken und Punkte in sortierter Reihenfolge. Zusätzlich speichern wir uns die Höhe für jedes Polygon, das von ℓ geschnitten wird und die Höhe der bereits gesehenen Punkte in S. Die Höhe eines Polygons ist die Anzahl der Polygone, in denen sich das Polygon befindet.

Ist das zu behandelnde Event v eine Ecke, werden die beiden inzidenten Kante in den Sweep-Line-Status eingefügt oder daraus entfernt (je nachdem, ob v ein Startpunkt oder ein Endpunkt der jeweiligen Kante ist). Für Kanten, die in den Sweep-Line-Status eingefügt werden, wird außerdem gespeichert, auf welcher Seite der Kante sich das zugehörige Polygon befindet.

Ist v die Ecke eines Polygons, das zum ersten Mal von ℓ geschnitten wird, oder ein Punkt in S, müssen wir die Höhe von v bestimmen. In beiden Fällen betrachten wir dazu die zwei benachbarten Kanten im Sweep-Line-Status. Ist v bei beiden Kanten auf der „äußeren“ Seite der zugehörigen Polygone (also nicht in den benachbarten Polygonen enthalten), speichern wir die Höhe der benachbarten Polygone als Höhe von v. Ist v in einem benachbarten Polygon in Höhe e enthalten, speichern wir für v die Höhe $e + 1$. Ist v die Ecke eines Polygons, setzen wir die Höhe des Polygons auf die Höhe von v.

Die Anzahl der Punkte in S, die in Höhe x liegen, ist genau die gesuchte Zahl.

Wie kommt man drauf?

Der erste Gedanke bei solchen Aufgaben ist inzwischen vermutlich meistens Sweep-Line. Da sich die gegebenen Polygone nicht schneiden, genügt es, jeweils die benachbarten Kanten im Sweep-Line-Status zu betrachten, um die jeweilige Höhe zu bestimmen.