

Triangulierung von einfachen Polygonen

Tobias Kyrion

Inhaltsverzeichnis

1.1	Die Problemstellung	1
2.1	Ein naiver Algorithmus	2
3.1	Zerlegung in monotone Teilpolygone	3
3.2	Triangulierung monotoner Polygone	4
4.1	Quellenangabe	5

1.1 Die Problemstellung

Die Problemstellung, die im Vortrag behandelt wird, ist die Zerlegung eines einfachen Polygons in Dreiecke. Ein Polygon ist durch endlich viele paarweise verschiedene Eckpunkte P_1, \dots, P_n , $n \in \mathbb{N}$ im \mathbb{R}^2 gegeben und die Verbindungsstrecken $\overline{P_i P_{i+1}}$, $i = 1, \dots, n-1$ und $\overline{P_n P_1}$ sind seine *Seiten*. Es ist *einfach*, wenn sich seine Seiten paarweise nicht schneiden.

Ein einfaches Polygon umgrenzt eine endlich große Fläche, die man sein *Inneres* nennt. Die Fläche, die das Innere umgibt, also der restliche \mathbb{R}^2 , heißt *Äußeres*. Das Polygon liefert folglich eine disjunkte Zerlegung des \mathbb{R}^2 in sein Inneres und Äußeres. Eine *Orientierung* der Seiten ist sinnvoll, da man mit ihrer Hilfe leichter Inneres und Äußeres festlegen kann. Im Folgenden seien die Seiten stets im Uhrzeigersinn angeordnet.

Eine Verbindungsstrecke von zwei Eckpunkten P_k, P_l , $k, l \in \{1, \dots, n\}$, $k \neq l$, die keine Seite ist und vollständig im Inneren verläuft, ist eine *Diagonale*. Wenn eine Diagonale eingefügt wird, zerfällt ein einfaches Polygon in zwei kleinere disjunkte einfache Teilpolygone, bei denen die Diagonale jeweils eine Seite bildet.

Dreiecke sind diejenigen einfachen Polygone mit geringstmöglicher Eckenanzahl. Offenbar kann man in Dreiecke keine Diagonale mehr einsetzen. Ziel ist es, durch iteratives Einfügen von Diagonalen in die jeweiligen Teilpolygone eine **Zerlegung in Dreiecke** zu erzeugen. Solch eine Zerlegung nennt man *Triangulation*.

2.1 Ein naiver Algorithmus

Ein naheliegender algorithmischer Zugang zur Triangulation ist der sogenannte **ear-clipping-Algorithmus**.

Zwei (bzgl. der Orientierung) aufeinanderfolgende Seiten bilden eine *Ecke*, d.h. eine Ecke ist stets durch drei Eckpunkte gegeben. Von diesen drei Eckpunkten ist derjenige, welcher mit beiden Seiten der Ecke inzidiert, der *mittlere Punkt*. Weiterhin ist der andere Eckpunkt auf der ersten Seite der *linke Punkt* und der auf der zweiten Seite der *rechte Punkt*. Eine Ecke ist ein *Ohr*, wenn kein anderer Eckpunkt im Dreieck zwischen dem linken, mittleren und rechten Punkt liegt.

Offenbar ist die Verbindungsstrecke vom linken und rechten Punkt eines Ohres eine Diagonale. Setzt man diese Diagonale ein, wird dadurch das Ohr vom Polygon abgetrennt. Das verbleibende Restpolygon besitzt dann einen Eckpunkt weniger als das Gesamtpolygon. Die **Grundidee** des ear-clipping-Algorithmus besteht nun darin, so lange Ohren vom jeweiligen Restpolygon anzuschneiden, bis nur noch ein Dreieck übrig bleibt, denn dann ist das Polygon vollständig trianguliert.

Der Algorithmus funktioniert iterativ. In jeder Iteration werden die Eckpunkte des Restpolygons der Reihe nach darauf überprüft, ob sie ein mittlerer Punkt eines Ohres sind. Sobald ein Ohr gefunden wird, wird es abgeschnitten und man fährt in gleicher Weise mit dem neuen Restpolygon fort.

Der Algorithmus ist korrekt, denn es ist bewiesen, dass jedes einfache Polygon mindestens ein Ohr besitzt. Nach Abschneiden eines Ohres ist das Restpolygon auch einfach, d.h., dass die Suche nach dem nächsten Ohr nach höchstens n Schritten erfolgreich ist, wobei $n \in \mathbb{N}$ die aktuelle Anzahl an Eckpunkten bezeichnet. In jeder Iteration wird also n um 1 verringert und somit terminiert der Algorithmus nach $n_{\text{gesamt}} - 3$ Iterationen.

Die Laufzeitanalyse ergibt, dass der ear-clipping-Algorithmus **quadratische** Laufzeit besitzt. Es stellt sich an dieser Stelle die Frage, ob es schnellere Algorithmen zur Triangulation gibt.

3.1 Zerlegung in monotone Teilpolygone

Die Strategie für einen schnelleren Algorithmus ist folgende: In einem ersten Schritt, der $O(n * \log(n))$ Zeit benötigt, wird das einfache Polygon in sogenannte *monotone Polygone* zerlegt. In einem zweiten Schritt werden diese in Linearzeit trianguliert.

Sei $g \subset \mathbb{R}^2$ eine Gerade. Ein einfaches Polygon ist *monoton* bzgl. g , wenn die Schnittmenge aller Geraden senkrecht zu g mit seinem Inneren entweder die leere Menge, ein Punkt oder eine Strecke ist. Man nennt g dann auch seine *Monotonieachse*. Falls die Monotonieachse parallel zur x - oder y -Achse des kartesischen Koordinatensystems im \mathbb{R}^2 ist, so spricht man von einem *x - bzw. y -monotonen* Polygon.

Man bemerkt, dass ein einfaches Polygon zwei Eckpunkte P_l und P_r besitzt, die am weitesten links bzw. rechts liegen. Ferner hat es eine *obere* und *untere Kette*, also zwei Streckenzüge aus Seiten, die beide bei P_l beginnen und bei P_r enden. Alle Eckpunkte auf der oberen Kette, ausgenommen die gemeinsamen Start- und Endpunkte, liegen oberhalb der unteren Kette.

Wandert man auf einer der Ketten von links nach rechts, kann es passieren, dass sich die **Durchlaufrichtung** von links nach rechts bzw. von rechts nach links ändert. Einen Eckpunkt, bei dem dieser Fall eintritt, nennt man *turn-vertex*. Man unterteilt die turn-vertices je nach Lage ihrer Nachbarpunkte und Größe ihrer *Innenwinkel* - das sind die Winkel, die von den inzidenten Seiten begrenzt werden und im Inneren liegen - in vier unterschiedliche Typen. Von besonderem Interesse sind dabei die sogenannten *merge-vertices* (Nachbarn beide links, Innenwinkel größer als 180°) und *split-vertices* (Nachbarn beide rechts, Innenwinkel größer als 180°), denn ein einfaches Polygon ist x -monoton, falls es **keine** split- oder merge-vertices aufweist.

Die Eckpunkte werden in einer **priority-queue** gespeichert, wobei die Priorität die Größe der x -Koordinate ist. Die Punkte werden dann aufsteigend nach ihrer x -Koordinate verarbeitet, wobei nur auf die Punkte mit den jeweils kleineren x -Koordinaten zugegriffen wird. Dieses Vorgehen ist als **scan-line-Prinzip** bekannt, denn es ist damit vergleichbar, dass man eine gedachte zur y -Achse parallele Linie von links nach rechts über das Polygon wandern lässt und stets nur die Punkte links von dieser bearbeitet.

Durch ausgeklügeltes Einsetzen von Diagonalen werden nun bei diesem Vorgang die split- und merge-vertices eliminiert, wodurch das Polygon in x -monotone Teile zerfällt. Dies ist stets ein korrekter Schritt. Die priority-queue-Operationen benötigen für jeden der $n \in \mathbb{N}$ Eckpunkte $O(\log(n))$ Zeit, wodurch die behauptete Laufzeit von $O(n * \log(n))$ für die Zerlegung zustande kommt.

3.2 Triangulierung monotoner Polygone

Weil ein x -monotones Polygon keine merge- oder split-vertices besitzt, sind die Eckpunkte in der oberen und unteren Kette aufsteigend nach der Größe ihrer x -Koordinate geordnet.

Außerdem stellt man fest, dass sich beliebige x -monotone Polygone in *monotone Streifen* und *Berge* zerlegen lassen. Ein monotoner Berg ist ein x -monotones Polygon, dessen obere oder untere Kette nur aus den beiden Eckpunkten P_l und P_r , die am weitesten links bzw. rechts liegen, besteht. Ein monotoner Streifen hat einen Eckpunkt abwechselnd in der oberen oder unteren Kette, wenn man die Eckpunkte von links nach rechts durchläuft.

Die Berge kann man mit einer Abwandlung des ear-clipping-Algorithmus mit Hilfe eines Stapels für die *konvexen* (d.h. Innenwinkel kleiner als 180°) Eckpunkte des aktuellen Restpolygons in Linearzeit triangulieren. Es wird hierbei der Umstand ausgenutzt, dass ein konvexer Eckpunkt eines Berges schon automatisch ein mittlerer Punkt eines Ohres ist.

Auch für die Triangulierung allgemeiner x -monotoner Polygone wird ein Stapel verwendet. Auf diesen wird immer derjenige nächste Punkt aus der oberen oder unteren Kette mit der kleineren x -Koordinate gelegt. Man verwendet die beiden Zählvariablen UpCntr und DownCntr, die mit 0 initialisiert werden. UpCntr wird um 1 erhöht und DownCntr gleich 0 gesetzt, wenn der neue Punkt für den Stapel aus der oberen Kette stammt, andernfalls wird DownCntr um 1 erhöht und UpCntr gleich 0 gesetzt.

Bevor die Zählvariablen gleich 0 gesetzt werden, haben sie entweder den Wert 0 (Initialisierung), sind gleich 1 oder größer als 1. Im ersten Fall ist nichts zu tun. Im zweiten Fall bilden der aktuelle Punkt und der oberste Punkt im Stapel eine Diagonale eines monotonen Teilstreifens. Diese wird dann direkt eingesetzt und der oberste Punkt im Stapel durch den aktuellen ersetzt. Im dritten Fall gehören der aktuelle Punkt und die Punkte im Stapel zu einem monotonen Teilberg. Man trianguliert diesen Teilberg in linearer Laufzeit und leert anschließend den Stapel.

Die Teilstreifen werden also direkt trianguliert. Korrektheit und lineare Laufzeit sind für diese offensichtlich. Der Algorithmus für die Teilberge ist korrekt, demnach sind deren Triangulierungen auch korrekte Schritte. Die Laufzeitanalyse ergibt insgesamt lineare Laufzeit für die Triangulierung monotoner Polygone.

Fazit:

Der soeben gänzlich beschriebene zweischrittige Triangulierungsalgorithmus für einfache Polygone hat eine Laufzeit von $O(n * \log(n))!$

4.1 Quellenangabe

M. de Berg, O. Cheong, M. van Krefeld, M. Overmars:

Computational Geometry, Algorithms and Applications, p. 45 - 60

http://valis.cs.uiuc.edu/~sariel/teach/2004/b/webpage/lec/24_triang_II.pdf