Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Geometrie
Sommersemester 2014
Martin Nöllenburg/Benjamin Niedermann

Übungsblatt 3 - Polygontriangulierung (Lösung)

Ausgabe: 29. April 2014
Abgabe: 06. Mai 2014

1 Korrektheitsbeweis

In der Vorlesung wurde der Algortihmus **MakeMonotone** vorgestellt. Im Verlauf der Ausführung des Algorithmus wird die Funktion **handleMergeVertex** augefrufen. Beweisen Sie die Korrektheit der Funktion.
Hinweis: Für Details über **MakeMonotone** und die verwendeten Datenstrukturen siehe Vorlesungsfolien.

Algorithmus 1 : handleMergeVertex

1 $e \leftarrow$ rechte Kante
2 **if** *isMergeVertex(helper*(e)*)* **then**
3 $\quad \mathcal{D} \leftarrow$ füge $\overline{\text{helper}(e)v}$ ein
4 lösche e aus $\mathcal{T}$
5 $e' \leftarrow$ Kante links von v in $\mathcal{T}$
6 **if** *isMergeVertex(helper*(e')*)* **then**
7 $\quad \mathcal{D} \leftarrow$ füge $\overline{\text{helper}(e')v}$ ein
8 $\text{helper}(e') \leftarrow v$

Lösung: Wir wollen zeigen, dass **handleMergeVertex** keine Diagonalen in das Polygon einfügt, die anderen Diagonalen oder andere Begrenzungen schneidet.

Hier: Beweisskizze nur für die erste Diagonale (Zeile 3) des Algorithmus. Angenommen, dass Diagonale d eine Kante des Polygons P schneidet. Dann existiert ein Knoten u des Polygons, der oberhalb von v liegen muss. Wenn er unterhalb von $\text{helper}(e)$ liegt, dann wäre $u = \text{helper}(e)$. Wenn er oberhalb von $\text{helper}(e)$ liegt, dann gäbe es keine horizontale Verbindungslinie von $\text{helper}(e)$ zu v die vollständig in P liegt. Widerspruch. Ähnliche Argumentation zeigt, dass keine Diagonale die eingefügt wurde von der neuen Diagonale d geschnitten werden kann.

2 Lineare Laufzeit?

Der in der Vorlesung vorgestellte Algorithmus zur Partitionierung eines Polygons in y-monotone Teilpolygone hat eine worst-case Laufzeit von $\mathcal{O}(n \log n)$. Modifizieren Sie den Algorithmus so, dass die worst-case Laufzeit auf $\mathcal{O}(n)$ sinkt, unter der Annahme, dass die Anzahl der Wendeknoten in den Eingabe-Polygonen in $\mathcal{O}(1)$ ist.

Lösung: Siehe ebenfalls Folien vom 07.05.2014. Algorithmus funktioniert in 2 Phasen.

1. Phase: Zunächst läuft man das Polygon gegen den Uhrzeigersinn ab. Dabei fasst man alle Knoten die zwischen zwei Wendeknoten liegen zu einer Liste zusammen. Diese Liste ist dann automatisch sortiert (s. Eigenschaft von Wendeknoten). Da es nur $\mathcal{O}(1)$ Wendeknoten gibt, gibt es auch nur $\mathcal{O}(1)$ viele solcher Listen. Diese kann man dann in $\mathcal{O}(n)$ sortieren (s. Mergesort).

2. Phase: Wir müssen den Baum $\mathcal{T}$ (s. Algorithmus aus der VL) ersetzen. Dabei genügt es wenn wir für jede Kante 'schnell' die Kante direkt links von ihr bestimmen können. Das Verfahren nutzt nun wieder die Listen aus der 1. Phase. Ähnlich einer Sweepline geht man von oben nach unten alle Knoten durch. Dabei merken wir uns in jeder Liste welchen Knoten wir bereits besucht haben. In jeder Liste bildet der letzte besuchte Knoten mit seinem Nachfolger (Knoten unterhalb des zuletzt besuchten Knotens) die interessanten Kanten. Ist unsere 'Sweepline' bei einem Knoten angelangt gehen wir alle Listen durch und schauen welche der $\mathcal{O}(1)$ interessanten Kanten der linke Nachbar des aktuellen Knotens ist. Da wir nur n Knoten betrachten und jeder Test auf linken Nachbarschaft $\mathcal{O}(1)$ Zeit benötigt liegt auch diese Phase in $\mathcal{O}(n)$.

Die 'interessanten' Kanten sind die einzigen die wir betrachten müssen, da jede Kante die linker Nachbar eines Knotens p ist einen Endknoten oberhalb und einen Knoten unterhalb von p haben muss.

Insgesamt ist die Laufzeit linear in der Anzahl der Eingabeknoten.

3 Das Kunstgalerie-Problem – Randabdeckung

Beweisen, oder widerlegen Sie folgende Aussage:
Sei $\mathcal{P}$ ein einfaches Polygon. Positioniert man Kameras so, dass alle Kameras zusammengenommen den vollständigen Rand von $\mathcal{P}$ abdecken, dann decken sie auch automatisch das gesamte Innere von $\mathcal{P}$ ab.

Lösung: Nein. Siehe Übungsfolien.

4 Polygon-Splitting

Geben Sie einen $\mathcal{O}(n \log n)$ Algorithmus an, der ein einfaches Polygon, bestehend aus n Knoten, in zwei einfache Polygone auftrennt wobei jedes aus höchstens $\lfloor 2n/3 \rfloor + 2$ Knoten besteht.
Hinweis: Triangulieren Sie das Eingabe-Polygon und verwenden Sie dann den Dualgraphen dieser Triangulierung.

Lösung: Der Dualgraph der Triangulierung der Polygon ist ein Baum mit maximal Grad 3. Zunächst legt man für jeden Knoten des Baums ein Gewicht mit Wert 1 fest. Dann wählt man ein beliebiges Blatt des Baumes und traversiert durch den Baum bis zu einem Knoten mit Grad drei. Bei der Traversierung werden die Gewichte der besuchten Knoten aufaddiert. Das Gewicht entspricht dann der Anzahl der Knoten des Teilbaums. Ist zu irgendeinem Zeitpunkt Gewicht erreicht das zwischen $n - (\lfloor 2n/3 \rfloor + 2)$ und $\lfloor 2n/3 \rfloor + 2$ liegt wird der Algorithmus abgebrochen und das Ergebnis ausgegeben. Trifft man auf einen Knoten mit Grad 3 wird das Gewicht des aktuellen Teilbaums zum Gewicht des Knotens mit Grad 3 hinzu addiert und der Teilbaum wird gelöscht. Es wird ein neues Blatt gewählt und das Verfahren so lange wiederholt bis der Algorithmus eine Lösung gefunden hat.

Um sicherzustellen, dass der Algorithmus korrekt ist müssen wir prüfen, ob die Vereinigung zweier Teilbäume (das passiert bei Knoten mit Grad 3) ein zu hohes Gewicht haben kann. Nehme dazu an, dass ein Teilbaum Gewicht x_1 und der zweite Teilbaum Gewicht x_2 hat. Da beide Teilbäume für sich keine Lösung sind muss deren Gewicht jeweils kleiner als $n-(\lfloor 2n/3 \rfloor+2)$ sein. Daraus folgt:

$$\begin{aligned} x_1 + x_2 &\leq 2 \cdot (n - (\lfloor 2n/3 \rfloor + 2)) + 1 \\ &= 2n - \lfloor 4n/3 \rfloor - 3 \\ &\leq 2n - 4n/3 - 3 \\ &= 2n/3 - 3 \\ &< \lfloor 2n/3 \rfloor + 2 \end{aligned}$$

Laufzeit: $\mathcal{O}(n \log n)$ für Triangulierung, Bildung des Dualgraphens.ss Der Rest des oben beschriebenen Algorithmus läuft in $\mathcal{O}(n)$.