YEVGENIYA ORLENKO, BSc.

ALGORITHMEN ZUM AUFBAU VON STRAIGHT SKELETONS

Masterarbeit

Informatik

Technische Universität Graz

Institut für Grundlagen der Informationsverarbeitung (7080)
Vorstand: O.Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Wolfgang Maass

Betreuer: Univ.-Prof. DI Dr.techn. Franz Aurenhammer

Graz, April 2014

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz am ....
Unterschrift

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either li terally or by content from the used sources.

Graz ....
date signature

Danksagung

An erster Stelle möchte ich mich bei Herrn Univ.-Prof. DI Dr.techn. Franz Aurenhammer für die Betreuung dieser Arbeit bedanken. Seine positive Einstellung und die zahlreichen Ideen und Anregungen begeisterten mich und gaben mir neue Motivation. Ein offenes Ohr zu jeder Zeit half auch über die schwierige Zeiten hinweg.

Außerdem einen riesigen Dank an MSc. Gunnar Schulze und Thorsten Lusser für die guten Ratschläge, die langen Diskussionen der diversen Ansätze und das Korrekturlesen der Arbeit.

Kurzfassung

Im Zentrum dieser Arbeit steht die Datenstruktur Straight Skeleton, eine interne Struktur für einfache Polygone, die erstmals 1995 in „A Novel Type of Skeleton for Polygons“[2] vorgestellt wurde. Diese wird unter anderem in der Architektur zum Modellieren von Dachflächen, sowie in geografischen Informationssystemen zur Berechnung der Mittellinien von Straßen und Flüssen eingesetzt.

Diese Arbeit untersucht zwei bekannte Algorithmen zum Aufbau von Straight Skeletons, die auf der Berechnung der Winkelhalbierenden beziehungsweise der Triangulierung[1] des Polygons basieren. Weiters wird ein neuer Algorithmus vorgestellt, welcher mittels der Pseudotriangulierung des Polygons arbeitet. Die drei Algorithmen werden in einem Programm implementiert, welches die Erstellung und Visualisierung von Straight Skeletons ermöglicht. Eine abschließende Diskussion vergleicht die verschiedenen Ansätze und geht auf Vor- und Nachteile bei der Implementierung ein.

Abstract

The focus of this work is the data structure straight skeleton, an internal structure for simple polygons that was first featured in „A Novel Type of Skeleton for Polygons“[2] in 1995. This is used, inter alia, in the architecture for modeling roofs and in geographical information systems for the calculation of the center lines of roads and rivers.

This thesis examines two well-known algorithms for the construction of Straight Skeletons, based on the computation of the angle bisector or the triangulation[1] of the polygon, respectively. Furthermore, a new algorithm is presented which operates by means of computing the pseudo-triangulation of the polygon. The three algorithms are implemented in a program that allows the creation and visualization of Straight Skeletons. A final discussion compares the different approaches and looks at the advantages and disadvantages in the implementation.

Inhaltsverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einleitung

Das Straight Skeleton ist eine Datenstruktur, welche 1995 in "A Novel Type of Skeleton for Polygons" von O. Aichholzer und F. Aurenhammer [2] vorgestellt wurde. Diese Datenstruktur bietet eine neue Möglichkeit, die internen Strukturen einfacher Polygone darzustellen. Die Form eines Polygons wird dabei durch sein topologisches Skelett repräsentiert, aus dem das ursprüngliche Polygon eindeutig rekonstruiert werden kann.

Straight Skeletons weckten somit das Interesse der Forscher, weshalb diese geometrische Datenstruktur aktiv erforscht und analysiert wurde. Dabei wurden Parallelen zu anderen geometrischen Strukturen gezogen und es entstanden verschiedene Algorithmen mit unterschiedlichen Ansätzen zur Erstellung von Straight Skeletons. Trotzdem bleiben mehrere wichtige Fragen in diesem Bereich noch offen. Über ein Straight Skeleton in 3D ist noch nicht viel bekannt.

Straight Skeletons werden bereits in einigen Bereichen effektiv genutzt. Sie finden unter anderem Anwendung in der Architektur als Verfahren für die Dachausmittlung, um die Schnittkanten zu bestimmen, die sich aus dem Zusammenschluss mehrerer Dachflächen ergeben. Weitere Anwendungen sind die Rekonstruktion von Flächen und die Berechnung von Straßen und Flüssen in geographischen Informationssystemen. Auch in der Medizininformatik dient diese Datenstruktur zur Objektrekonstruktion in der Bildverarbeitung. Weitere Anwendungen sind die Berechnung von Werkzeugwegen für CNC-Maschinen im CAD/CAM-Bereich sowie das Lösen von Falt- und Schneideproblemen in mathematischen Origami.

Dieses breite Spektrum an Anwendungen und unbeantworteten Fragen machen Straight Skeletons zu einem interessanten Forschungsbereich.

1.1 Zielsetzung

Die vorhandenen Algorithmen zur Erstellung von Straight Skeletons, welche in "A Novel Type of Skeleton for Polygons"[2] und "Straight Skeletons for general polygonal figures in the plane"[1] vorgestellt wurden, sind leider nicht sehr effizient und haben eine Laufzeit von $\mathcal{O}(n^2\ log(n))$ bzw. $\mathcal{O}(n^3)$. Ziel dieser Arbeit ist die Entwicklung eines neuen Verfahrens basierend auf der geometrischen Struktur Pseudotriangulierung, welches eine bessere Laufzeit bei der Erstellung von Straight Skeletons aufweisen soll. Ein weiteres Ziel ist die Implementierung eines Programms, welches den Aufbau von Straight Skeletons mit allen drei Algorithmen ermöglicht, und den Prozess des Aufbaus der Datenstruktur mit dem jeweiligen Algorithmus visualisiert. Der Algorithmus soll zur Laufzeit des Programms in jedem Schritt änderbar sein. Mit Hilfe dieses Programms soll eine Evaluierung des neuen Algorithmus durchgeführt werden.

1.2 Gliederung der Arbeit

Kapitel 2 ist der theoretische Teil dieser Arbeit und dient als Grundlage für Kapitel 3. Es beinhaltet alle wichtigen Definitionen und Informationen zu dieser Datenstruktur, die notwendig sind, um dem weiteren Verlauf dieser Arbeit folgen zu können. In Kapitel 3. befindet sich der praktische Teil dieser Arbeit. Dieser befasst sich mit dem Programm zur Erstellung von Straight Skeletons. Eine Evaluierung des neuen Ansatzes, welche mit Hilfe des implementierten Programms durchgeführt wurde, wird in Kapitel 4 dieser Arbeit erläutert. Kapitel 5 bietet einen kurzen Überblick über einige thematisch verwandte Arbeiten. Zum Abschluss werden in Kapitel 6 die theoretischen und praktischen Teile dieser Arbeit zusammengefasst.

Kapitel 2

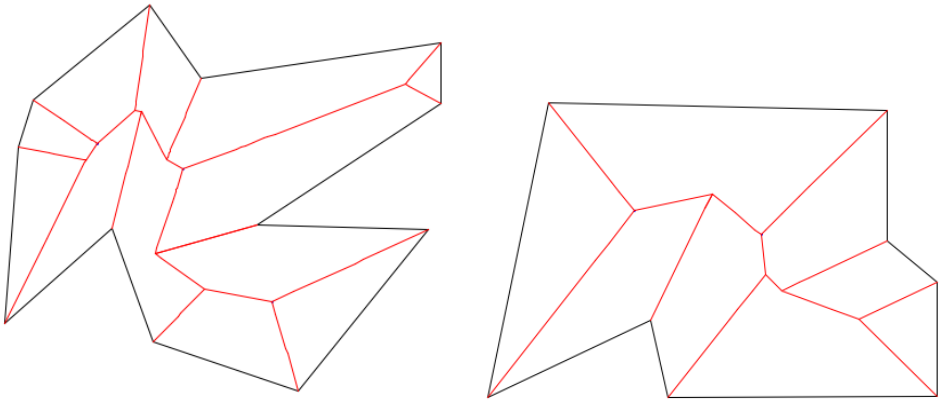
Theorie

Das Straight Skeleton wird in "Straight skeletons for general polygonal figures in the plane"[1] wie folgt definiert:

Definition 1 *Das Straight Skeleton $S(P)$ eines Polygons P ist die Vereinigung aller Winkelhalbierenden, die während des Schrumpfprozesses an den Punkten des Polygons entstehen. $S(P)$ ist eine eindeutige Struktur, die eine polygonale Zerlegung von P definiert.*

Es lässt sich erkennen, dass die Definition dieser Datenstruktur sehr stark an den Aufbauprozess (das Schrumpfen des Polygons) angelehnt ist. Zwei Beispiele für Straight Skeletons sind in Abbildung 2.1a und 2.1b dargestellt. Die schwarzen Linien entsprechen den Kanten des Basispolygons, das zugehörige Straight Skeleton ist rot eingezeichnet. Ein Straight Skeleton besteht aus geraden Liniensegmenten, die einen Baum aufspannen. Die Knoten des Baums haben in den meisten Fällen den Grad drei, Grade von vier oder höher kommen bei Straight Skeletons nur in speziellen Fällen vor.
Der Aufbau des Straight Skeletons erfolgt durch einen "Schrumpfprozess" (engl. shrinking process), bei dem alle Kanten des Polygons parallel zu ihren Ursprungskanten mit der gleichen konstanten Geschwindigkeit in das Innere des Polygons verschoben werden. Dieser Prozess wird so lange durchgeführt, bis alle Kanten des Polygons auf die Länge null geschrumpft sind. Alternativ könnte man sagen, dass die Eckpunkte des Polygons entlang ihrer Winkelhalbierenden nach innen verschoben werden. Die Verschiebungsgeschwindigkeit jedes einzelnen Punktes wird hierbei durch die Größe des Winkels zwischen den benachbarten Kanten des Punktes beeinflusst. Der Schrumpfprozess wird so lange durchgeführt, bis die Fläche des Polygons null ist.

Das soeben beschriebene Verfahren ist nicht die einzige Möglichkeit Straight Skeletons zu berechnen. Im Folgenden wird im Detail auf drei Verfahren



(a) (b)

Abbildung 2.1: Straight Skeleton

Legende:
Schwarz: Linien des Basis-Polygons
Rot: Linien des Straight Skeletons

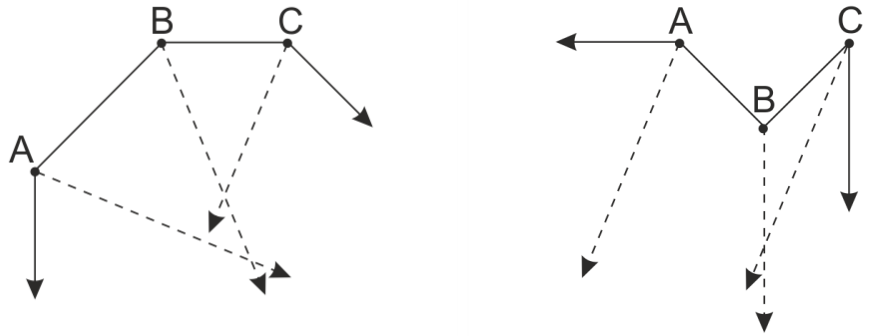
eingegangen: den oben beschriebenen Ansatz über die Winkelhalbierenden sowie zwei Verfahren, die auf der Triangulierung bzw. Pseudo-Triangulierung des Basispolygons beruhen.

2.1 Berechnung über die Winkelhalbierenden

Dieser Algorithmus wurde in "A Novel Type of Skeleton for Polygons" [2] von O. Aichholzer und F. Aurenhammer publiziert. Während der Berechnung mit diesem Algorithmus treten zwei verschiedene Typen von Ereignissen auf, welche im Folgenden beschrieben werden. Jedes Ereignis verändert die Topologie des Straight Skeletons, d.h. es entstehen neue Knoten in dessen Struktur. Das Schrumpfen des Polygons bis zum nächsten Event wird als Schritt bezeichnet. Falls mehrere Ereignisse im gleichen Punkt auftreten, entstehen Knoten mit Grad 4 oder höher.

2.1.1 Edge Event

Ein Edge Event tritt genau dann auf, wenn die Länge der Kante zwischen zwei benachbarten Punkten auf null schrumpft und diese zwei Punkte zu einem Punkt zusammenfallen. Dieser Fall kann sowohl bei konvexen als auch konkaven Ecken auftreten. Die Ermittlung von Edge Events erfolgt durch die Berechnung des Schnittpunktes zwischen den Winkelhalbierenden zweier benachbarter Punkte. Die Abbildungen 2.2a und 2.2b zeigen exemplarisch Zwei mögliche Situationen die bei der Ermittlung eines Edge Events.



(a) (b)

Abbildung 2.2: Zwei Beispiele für Edge Events

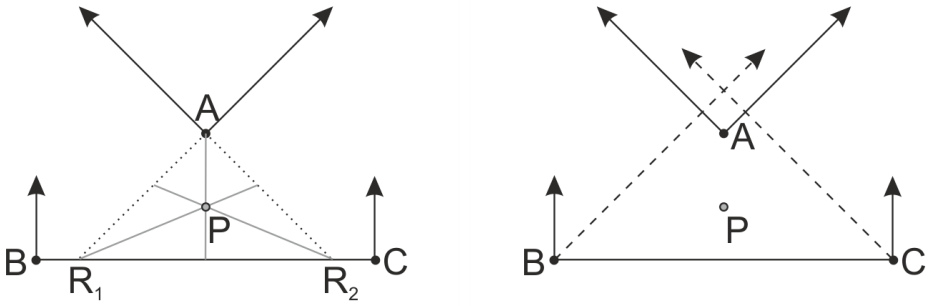
In Abbildung 2.2a sind die zwei Schnittpunkte der Winkelhalbierenden durch die Eckpunkte (A) und (B) sowie der Winkelhalbierenden durch die (B) und (C) dargestellt. Man sieht, dass sich die Winkelhalbierende durch Punkt

(B) zuerst mit der Winkelhalbierenden durch Punkt (C) schneidet und erst danach mit der Winkelhalbierenden durch Punkt (A). Somit findet das Edge Event der Kante (BC) früher statt als das Edge Event der Kante (AB).
In Abbildung 2.2b ist nur ein Edge Event, nämlich das der Kante (BC), dargestellt. Die Winkelhalbierenden der benachbarten Punkte (A) und (B) schneiden sich in dieser Situation nicht.

Die Schrittgröße für den Schrumpfprozess bei Edge Events ist der Normalabstand zwischen dem Schnittpunkt der Winkelhalbierenden zweier benachbarter Punkte und der dazwischenliegenden Kante.

2.1.2 Split Event

Ein Split Event tritt genau dann auf, wenn eine konkave Ecke des Polygons während des Schrumpfprozesses auf eine andere Kante des Polygons trifft und diese somit in zwei Teile spaltet. Abbildung 2.3 zeigt eine Situation, in der ein Schnittpunkt (P) zwischen der konkaven Ecke (A) und der gegenüberliegenden Kante (BC) entsteht.



(a) (b)

Abbildung 2.3: Beispiel für ein Split Event

Die Berechnung von Split Events wird wie folgt durchgeführt. Jede konkave Ecke des Polygons wird auf potenzielle Schnittpunkte mit allen anderen Kanten des Polygons geprüft. Die benachbarten Kanten der aktuell geprüften Ecke sind aus der Überprüfung ausgenommen. Hierzu werden die von der

konkaven Ecke ausgehenden Kanten in das Innere des Polygons verlängert, bis sie sich mit der aktuell ausgewählten Gegenkante schneiden. In Abbildung 2.3a sind die verlängerten Kanten als grau punktierte Linien (AR_1) und (AR_2) dargestellt. Für das entstandene Dreieck (AR_1R_2) wird der Inkreis-Mittelpunkt (P) berechnet, welcher ein potenzielles Split Event darstellt.

Als letztes muss geprüft werden ob sich der Punkt (P) in dem für die Kante (BC) erreichbaren Bereich befindet. Dieser Bereich ist in Abbildung 2.3b dargestellt und spannt in den meisten Fällen ein Dreieck auf, welches sich durch die ausgewählte Gegenkante (BC) und der Winkelhalbierenden ihrer Ecken (B) und (C) bildet. Falls sich die Winkelhalbierenden der Ecken (B) und (C) nicht schneiden, nimmt dieser Bereich die Form eines offenen Polygons an. Durch diese Fläche werden alle für die Kante (BC) erreichbaren Positionen repräsentiert, welche beim Schrumpfen des Polygons erreichbar sind. Liegt ein potenzielles Split Event außerhalb dieses Bereiches, ist es für die Kante (BC) nicht erreichbar und kann somit nicht stattfinden.

Die Schrittgröße für den Schrumpfprozess bei Split Events ist der Abstand zwischen dem Punkt (P) und der gegenüberliegende Kante (BC), welcher dem Radius des Inkreises des Dreiecks (AR_1R_2), entspricht.

2.1.3 Algorithmus

Algorithmus 1 berechnet das Straight Skeleton eines Polygons auf Basis der Winkelhalbierenden. Die einzelnen Schritte des Algorithmus sind in Abbildung 2.4 veranschaulicht.

Algorithmus 1 Erstellung von Straight Skeletons über die Winkelhalbierenden

INITIALISIERUNG

Bestimme die Winkelhalbierenden für alle Punkte des Polygons.

Berechne alle Edge- und Split Events und füge sie in eine Liste ein.

Sortiere die Liste aufsteigend.

SCHRITT

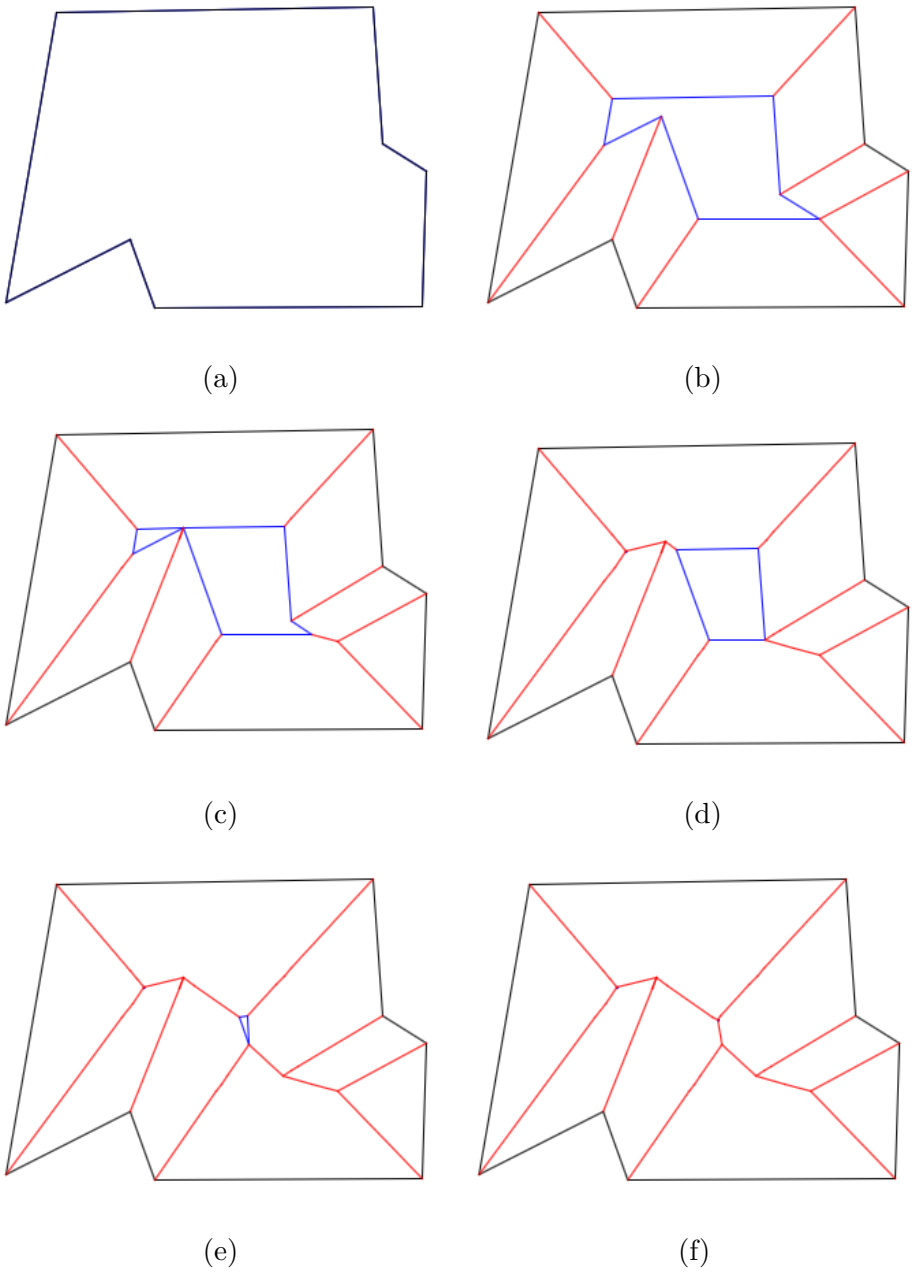
Solange der Flächeninhalt des Polygons größer null ist

Nimm das kleinste Element aus der Liste und schrumpfe das Polygon bis zu diesem Ereignis.

Berechne die Winkelhalbierenden für alle neuen Ecken, die durch das Event entstanden sind.

Berechne die neuen Events, die durch das Einfügen der neuen Winkelhalbierenden entstanden sind, und füge sie sortiert in die Liste ein.

Die bereits vorhandenen Winkelhalbierenden und Events müssen nicht aktualisiert werden, da sie sich durch Parallelverschiebung der Kanten des Polygons nicht verändert haben.



(a) (b)

(c) (d)

(e) (f)

Abbildung 2.4: Schritt für Schritt Aufbau des Straight Skeletons mit Hilfe von Algorithmus 1

Legende:
Schwarz: Linien des Basis-Polygons
Rot: Linien des Straight Skeletons
Blau: Das Polygon im aktuellen Schrumpfschritt

Laufzeitabschätzung:

Initialisierung

- Die Berechnung der Winkelhalbierenden für alle Ecken des Polygons kann wie die Berechnung aller Edge Events in $\mathcal{O}(n)$ durchgeführt werden.
- Die Berechnung der Split Events benötigt $\mathcal{O}(n^2)$ Schritte.
- Dabei entsteht eine Liste mit $\mathcal{O}(n^2)$ Events, welche in $\mathcal{O}(n^2 \log(n))$ sortiert werden kann.

Somit beträgt die gesamte Laufzeit für die Initialisierung des Algorithmus $\mathcal{O}(n^2 \log(n))$.

Schritt

- Das Auswählen des kleinsten Elements, die Berechnung der neuen Winkelhalbierenden sowie die Aktualisierung der Edge Events können in $\mathcal{O}(1)$ durchgeführt werden.
- Die Aktualisierung der Split Events benötigt $\mathcal{O}(n)$ Schritte.
- Der Schritt wird maximal $n-2$ mal wiederholt, da $n-2$ die maximale Anzahl der Knoten in einem Straight Skeleton ist.

Somit ergibt sich eine Laufzeit von $\mathcal{O}(n^2)$ für die Durchführung aller Schritte.

Die gesamte Laufzeit des Algorithmus beträgt somit $\mathcal{O}(n^2\ log(n))$.

2.2 Berechnung mittels Polygontriangulierung

Dieser Algorithmus wurde in "Straight Skeletons for general polygonal figures in the plane"[1] publiziert und basiert auf einer Triangulierung des Polygons. Um eine visuell bessere Triangulierung zu erzielen, d.h. lange und spitze Dreiecke zu vermeiden, wurde eine constrained Delaunay Triangulierung verwendet. Die Laufzeiten des Algorithmus werden durch diese Änderungen nicht beeinflusst.

Definition 2 *Eine Triangulierung oder Dreieckszerlegung eines Polygons ist eine Menge von Kanten, welche jeden Punkt des Polygons mit einem weiteren verbindent, ohne dass die Kanten sich schneiden. Zudem müssen die durch die Kanten begrenzten Flächen allesamt Dreiecke sein. Die Fläche außerhalb des Polygons stellt natürlich eine Ausnahme dar. [4]*

Die constrained Triangulierung ist wie folgt definiert:

Definition 3 *Gegeben ist ein planarer linearer Graph G. Eine constrained Triangulierung von G ist eine Triangulierung der Ecken von G, welche die Kanten von G beinhaltet und durch diese Kanten begrenzt ist.[7]*

Die constrained Delaunay-Triangulierung ist wie folgt definiert:

Definition 4 *Gegeben ist ein planarer linearer Graph G. Die constrained Delaunay-Triangulierung von G ist eine constrained Triangulierung von G mit einer zusätzlichen Eigenschaft. Für beliebige zwei Punkte (p,q) gilt dann:*

- *p,q bilden genau dann eine constrained Delaunay-Kante, wenn ein Kreis über (C(p,q)) existiert, sodass kein weiterer Punkt r, welcher von den Punkten p und q gesehen wird, sich in diesem Kreis befindet (Umkreis-Kriterium).*
- *Eine constrained Triangulierung ist genau dann eine constrained Delaunay-Triangulierung, wenn alle Kanten der constrained Triangulierung constrained Delaunay-Kanten sind.*

Die Kanten eines triangulierten Polygons können in äußere und innere Kanten unterschieden werden. Die Kanten des Ursprungspolygons werden als äußere Kanten bezeichnet, die Kanten, welche durch die Triangulierung neu entstanden sind und keine äußeren Kanten sind, als innere Kanten.

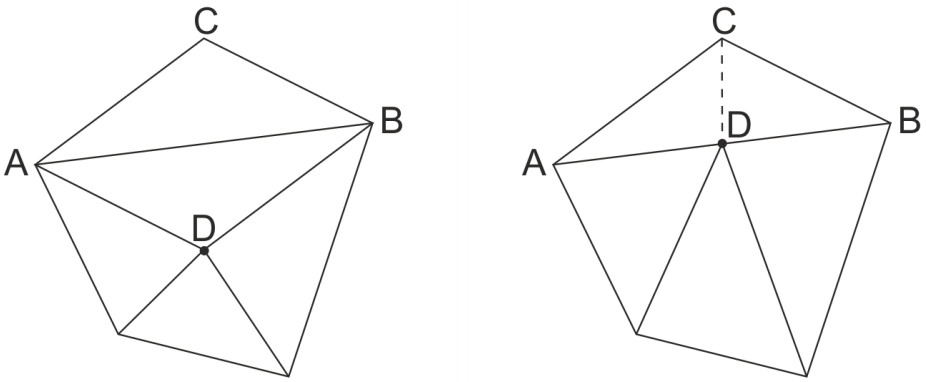
Ein Polygon besteht somit nur aus äußeren Kanten. Ein Dreieck kann aus nur inneren Kanten (liegt im Inneren des Polygons), nur äußeren Kanten (das

Polygon ist ein Dreieck), oder einer Mischung aus beiden (häufigste Situation bei einer Triangulierung) bestehen.

Innere und äußere Kanten unterscheiden sich während des Schrumpfens des Polygons stark in ihrem Verhalten. Die äußeren Kanten werden immer nur parallel verschoben. Punkte von inneren Kanten liegen nicht benachbart auf dem Polygon und bewegen sich somit unabhängig voneinander. Für die inneren Kanten bedeutet das, dass sie sich während des Schrumpfens in unterschiedliche Richtungen bewegen und ihre Neigung verändern können. Diese können dabei eine Drehung von bis zu 180° durchführen.

Die Events in diesem Algorithmus werden durch bestimmte Ereignisse in den Dreiecken der Triangulierung ausgelöst. Es wurde beobachtet, dass bei jedem Event aus Kapitel 2.1.3 ein Dreieck der Triangulierung zusammenfällt. Dabei kann jedes Dreieck auf drei verschiedene Arten zusammenfallen. Diese werden in den nächsten drei Kapiteln genauer ausgeführt. Die detaillierte Beschreibung der Berechnung aller drei Events befindet sich in Kapitel 2.2.4.

2.2.1 Flip Event



(a) (b)

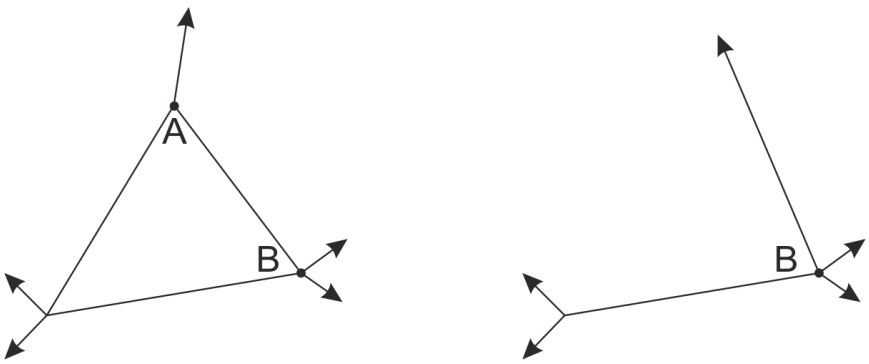
Abbildung 2.5: Flip Event

Ein Flip Event tritt genau dann auf, wenn ein Punkt (D) über die gegenüberliegende innere Kante (AB) springt. Dabei klappt das Dreieck (ABD) zusammen. Jetzt befinden sich die Punkte (A), (D) und (B) auf einer Linie,

und somit ist die Triangulierung des Polygons nach dieser Operation nicht mehr vollständig. Eine neue innere Kante (CD) wird eingefügt und vervollständigt die vorhandene Triangulierung.
Im Gegensatz zu Edge- und Split Events entstehen bei einem Flip Event keine neuen Ecken im Polygon und das Straight Skeleton verändert seine Struktur dabei nicht. Man könnte auch sagen, dass es sich um ein Zwischenevent handelt, welches aber sehr wichtig ist um die Triangulierung des Polygons aufrecht zu erhalten. Die Situation vor bzw. nach dem Flip Event ist in Abbildung 2.5a und 2.5b dargestellt.

2.2.2 Edge Event

Ein Edge Event tritt genau dann auf, wenn eine Kante (AB) des Dreiecks auf die Länge null schrumpft. Die Punkte (A) und (B) fallen zu einem Punkt (B) zusammen. Die Triangulierung des Polygons wird um ein Dreieck kleiner, bleibt dabei aber vollständig und muss somit nicht angepasst werden. Die Situation vor bzw. nach einem Edge Event ist in Abbildung 2.6a und 2.6b dargestellt.

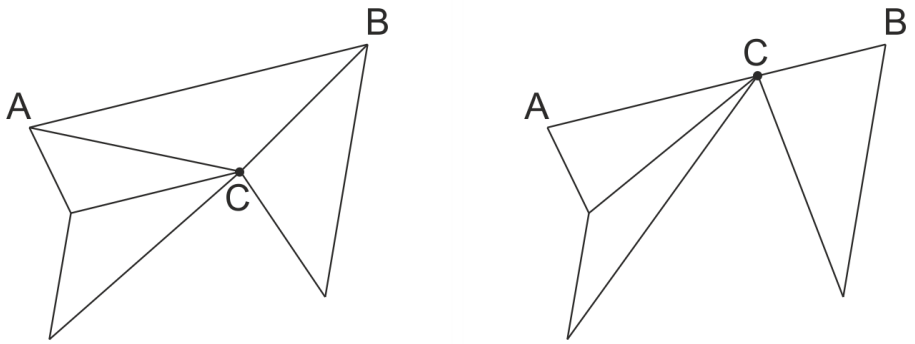


(a) (b)

Abbildung 2.6: Edge Event

2.2.3 Split Event

Ein Split Event tritt genau dann auf, wenn ein Punkt (C) auf eine gegenüberliegende Kante (AB) fällt. Dabei fällt das Dreieck (ABC) zu einer Kante (AB) zusammen. Die Kante (AB) wird in zwei Kanten (AC) und (BC) aufgeteilt. Die Triangulierung des Polygons wird um ein Dreieck kleiner. Ein Split Event unterscheidet sich von einem Flip Event dadurch, dass die Triangulierung des Polygons vollständig bleibt und das Einfügen einer neuen inneren Kante nicht notwendig ist. Die Situation vor bzw. nach dem Split Event ist in Abbildung 2.7a und 2.7b dargestellt.



(a) (b)

Abbildung 2.7: Split Event

2.2.4 Berechnung von Events

Für die Berechnung von Events in diesen Algorithmus werden vier verschiede Ansätze vorgestellt. Bei jedem Ansatz wird eine Gleichung aufgestellt, die für die Berechnung der Schrumpfschrittgröße verantwortlich ist. Der erste Teil findet in allen vier Ansätzen Anwendung und dient zum Erstellen einer Gleichung, die die Bewegungen von Punkten des Polygons in Abhängigkeit der Bewegungen der äußeren Kanten modelliert.

Dafür betrachtet man einen Teil des Polygons, welches aus drei Liniensegmenten (q),(w) und (r) besteht (siehe Abbildung 2.8). Durch das Schneiden von Liniensegmenten entstehen die Ecken (A) und (B) des Polygons. Die

Ecke (A) mit innerem Winkel (α)<180°, welche von den Liniensegmenten (q) und (w) gebildet wird, und die Ecke (B) mit innerem Winkel (β)>180°, welche durch die Liniensegmente (w) und (r) entsteht. Aus beiden Punkten geht eine Winkelhalbierende ins Innere des Polygons. Die Winkelhalbierende sind als strichlierte Linien in Abbildung 2.8 dargestellt. Die entsprechenden Eckpunkte bewegen sich beim Schrumpfen des Polygons entlang dieser Winkelhalbierenden.

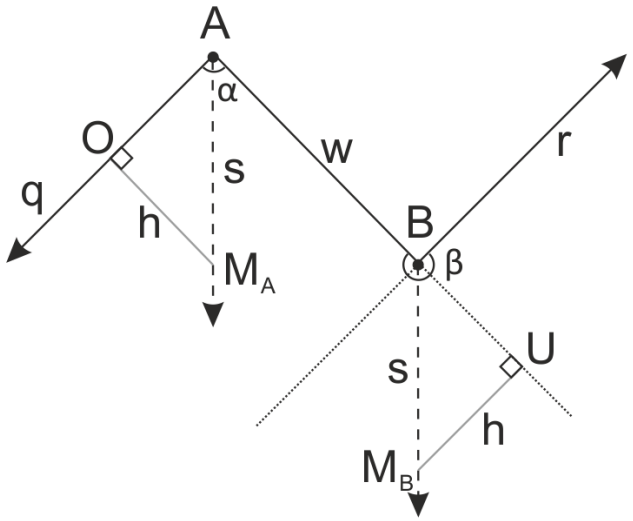


Abbildung 2.8: Ausschnitt aus einem Polygon

Man nimmt an, dass der Punkt (A) durch das Schrumpfen zu Punkt (M_A) verschoben wurde. Um zu berechnen, wie weit dabei die benachbarte Kante verschoben wurden, wird ein zusätzliches Liniensegment (OM_A) eingeführt. Dieses Liniensegment verbindet den Punkt (M_A) mit der äußeren Kante (q) in einem Winkel von 90°. Dabei entsteht ein rechtwinkeliges Dreieck (OAM_A), in dem die Kathete (h) die Verschiebung der äußeren Kante (q) und die Hypotenuse (s) die entsprechende Verschiebung des Punktes (A) darstellen. Die Abhängigkeit zwischen der Kathete und der Hypotenuse kann mit Hilfe des dazwischenliegenden Winkels (AM_AO) hergeleitet werden. Der Winkel (AM_AO) kann wie folgt berechnet werden.

Der Winkel (α) wird durch die Winkelhalbierende in zwei gleiche große Winkel (M_AAO) und (M_AAB) geteilt. Der Winkel (α) kann berechnet werden, da die Koordinaten der Ecken des Polygons bekannt sind. Somit können auch

die Winkel $(M_A AO)$ und $(M_A AB)$ berechnet werden. Wie bereits erwähnt, werden beim Schrumpfprozess die äußeren Kanten des Polygons immer parallel verschoben. Somit bleibt der Winkel zwischen zwei benachbarten äußeren Kanten konstant. Das Dreieck (OAM_A) hat somit zwei bekannte Winkel und der gesuchte dritte Winkel $(AM_A O)$ kann einfach berechnet werden.

$$180° = 90° + \measuredangle OAM_A + \measuredangle AM_A O \tag{2.1}$$

$$180° = 90° + \frac{\alpha}{2} + \measuredangle AM_A O \tag{2.2}$$

$$\measuredangle AM_A O = 90° - \frac{\alpha}{2} \tag{2.3}$$

Nach der Berechnung des unbekannten Winkels kann die Abhängigkeit zwischen der Bewegung der Kante (q), die durch die Kathete (h) repräsentiert wird, und der Bewegung des Punktes (A), die durch Hypotenuse (s) repräsentiert wird, mit Hilfe des Kosinus des Winkels $(AM_A O)$ berechnet werden.

$$cos(90° - \frac{\alpha}{2}) = \frac{h}{s} \tag{2.4}$$

$$s = \frac{h}{cos(90° - \frac{\alpha}{2})} \tag{2.5}$$

$$s = \frac{h}{sin(\frac{\alpha}{2})} \tag{2.6}$$

Im Fall von Punkt (B) werden ein paar zusätzliche Operationen durchgeführt. Die äußeren Kanten (w) und (r) werden in das Innere des Polygons verlängert (siehe Abbildung 2.8). Man nimmt an, dass der Punkt (B) beim Schrumpfen auf den Punkt (M_B) verschoben wurde. Um zu berechnen, wie weit die benachbarten Kanten verschoben wurden, wird ein zusätzliches Liniensegment (OM_B) eingefügt. Dieses Liniensegment verbindet den Punkt (M_B) mit der verlängerten äußeren Kante (w) unter einem Winkel von 90°. Dabei entsteht ein rechtwinkliges Dreieck (BUM_B), in dem die Kathete (h) die Verschiebung der Kante (w) und die Hypotenuse (s) die entsprechende Verschiebung des Punktes (B) darstellen. Die Abhängigkeit zwischen der Kathete und der Hypotenuse kann mit Hilfe des dazwischenliegenden Winkels $(BM_B U)$ hergeleitet werden. Der Winkel $(BM_B U)$ wird wie folgt berechnet.

Der äußere und der innere Winkel, welche sich durch die äußeren Kanten (w) und (r) ergeben, bilden zusammen einen vollen Kreis und damit einen Winkel von 360°. Der Außenwinkel beträgt somit $360° - \beta$.

Wie bereits erwähnt, wurden die äußeren Kanten (w) und (r) in das Innere des Polygones verlängert und bilden einen neuen Winkel, dessen Größe auch $360° - \beta$ beträgt. Da der Winkel (β) und der neue Winkel durch die zwei gleichen Geraden gebildet werden, ist die Winkelhalbierende von Punkt (B) auch eine Winkelhalbierende des neuen Winkels. Daraus ergibt sich, dass der Winkel (UBM_B) gleich $\frac{360°-\beta}{2}$ ist. Das Dreieck (UBM_B) hat damit zwei bekannte Winkel und der gesuchte dritte Winkel $(BM_B U)$ kann einfach berechnet werden.

$$180° = 90° + \measuredangle UBM_B + \measuredangle BM_B U \tag{2.7}$$

$$180° = 90° + \frac{360° - \beta}{2} + \measuredangle BM_B U \tag{2.8}$$

$$180° = 90° + 180° - \frac{\beta}{2} + \measuredangle BM_B U \tag{2.9}$$

$$180° = 270° - \frac{\beta}{2} + \measuredangle BM_B U \tag{2.10}$$

$$\measuredangle BM_B U = \frac{\beta}{2} - 90° \tag{2.11}$$

Die Bewegung der Kante (w), die durch Kathete (h) repräsentiert wird, und die Bewegung von Punkt (B), die durch Hypotenuse (s) repräsentiert wird, kann nun mit Hilfe des Kosinus vom Winkel $(BM_B U)$ beschrieben werden.

$$cos(\frac{\beta}{2} - 90°) = \frac{h}{s} \tag{2.12}$$

$$s = \frac{h}{cos(\frac{\beta}{2} - 90°)} \tag{2.13}$$

$$s = \frac{h}{sin(\frac{\beta}{2})} \tag{2.14}$$

Damit wurde gezeigt, dass die Abhängigkeit zwischen der Bewegung der Ecken und den äußeren Kanten des Polygons, für die spitze und stumpfe Winkel gleich berechnet werden können, und keine Fallunterscheidung getroffen werden muss.

Desweiteren werden folgende Bezeichnungen verwendet:

(A), (B) und (C) sind beliebige Punkte des Polygons, welche ein Dreieck der Triangulierung des Polygons darstellen.

(M_A), (M_B), (M_C) sind die beim Schrumpfen des Polygons entsprechend verschobenen Punkte (A), (B) und (C).

(t_A) und (b_A) sind die Kofeffizienten der Geradengleichung der Winkelhalbierende des Punktes (A).

Ax und M_Ax sind die x-Koordinaten von Punkt A bzw. von M_A

Ay und M_Ay sind die y-Koordinaten von Punkt A bzw. von M_A.

Die Länge der Katheten (AM_A) bzw. (BM_B) kann als der Abstand zweier Punkte in der Ebene über die Koordinaten der Punkte (A) und (M_a) berechnet werden.

$$s = \sqrt[2]{(M_Ax - Ax)^2 + (M_Ay - Ay)^2} \tag{2.15}$$

Setzt man die Gleichungen 2.14 und 2.15 gleich, entsteht folgende Gleichung.

$$\sqrt[2]{(M_Ax - Ax)^2 + (M_Ay - Ay)^2} = \frac{h}{sin(\frac{\alpha}{2})} \tag{2.16}$$

Wenn man annimmt, dass der Parameter (h) bekannt ist, hat die Gleichung 2.16 nur noch zwei unbekannte Variablen: die Koordinaten (M_Ax) und (M_Ay) des verschobenen Punktes (M_A).

Als nächstes betrachtet man die Geradengleichung der Winkelhalbierenden durch Punkt (A).

$$M_Ay = t_A\ M_Ax + b_A \tag{2.17}$$

Die Koeffizienten (t) und (b) können aus den Koordinaten der Punkte des Polygons berechnet werden. Somit kann die Variable M_Ay der Gleichung 2.16 in die Gleichung 2.17 eingesetzt werden.

$$\sqrt[2]{(M_Ax - Ax)^2 + (t_A\ M_Ax + b_A - Ay)^2} = \frac{h}{sin(\frac{\alpha}{2})} \tag{2.18}$$

Die Lösung dieser Gleichung mit den entsprechenden Parametern ist die x-Koordinate für jeden geschrumpften Punkt (M) in Abhängigkeit des Schrumpfschrittes (h).

$$
\begin{aligned}
M_A x = & \frac{1}{(sin(\frac{\alpha}{2})^2) + (sin(\frac{\alpha}{2}))^2\ t} \\
& (A_x\ sin^2(\frac{\alpha}{2}) + A_y\ sin^2(\frac{\alpha}{2})\ t_A - sin^2(\frac{\alpha}{2})\ b_A\ t_A \pm \\
& (h^2\ sin^2(\frac{\alpha}{2}) - A_y^2\ sin^4(\frac{\alpha}{2}) + 2\ A_y\ sin^4(\frac{\alpha}{2})\ b_A - \\
& sin^4(\frac{\alpha}{2})\ b_A + 2\ A_x\ A_y\ sin^4(\frac{\alpha}{2})\ t_A - \\
& 2\ A_x\ sin^4(\frac{\alpha}{2})\ b_A\ t_A + h^2\ sin^2(\frac{\alpha}{2})\ t_A - A_x^2\ sin^4(\frac{\alpha}{2})\ t_A^2)^{\frac{1}{2}}
\end{aligned}
\tag{2.19}
$$

Die y-Koordinate kann mit Hilfe der Geradengleichung aus 2.17 berechnet werden.

$$
M_A y = t_A\ M_A x + b_A \tag{2.20}
$$

Die Gleichung 2.18 ist eine quadratische Gleichung und hat somit zwei Nullstellen. Nach der Berechnung der entsprechenden y-Koordinate bekommt man zwei unterschiedliche Punkte. Einer der Punkte befindet sich im Inneren des Polygons und der andere liegt außerhalb des Polygons (dies kann man als negatives Schrumpfen des Polygons bezeichnen: die äußeren Kanten bewegen sich nach außen). In den weiteren Berechnungen werden nur die Punkte verwendet, die im Inneren des Polygons liegen.

Im zweiten Schritt werden die Gleichungen für (M_Ax) und (M_Ay) zu größeren Gleichungen zusammengesetzt. Jede dieser Gleichungen modelliert die Bedingungen für ein oder mehrere Events. Die Lösung jeder Gleichung ist der Schrumpfschritt, mit dem man zu einem Event beim Schrumpfen des Polygons gelangen kann. In diesem Ansatz spielt die Art des Events keine Rolle und wird erst bei der Anpassung des Polygons nach der Durchführung eines Schrumpfschrittes miteinbezogen.

Gleichung 1: basierend auf den Kanten des Dreiecks

Das Kriterium für ein Event ist die Situation, in der zwei Punkte des Dreiecks beim Schrumpfen des Polygons die gleiche x- und y-Koordinaten erhalten. Anders ausgedrückt bedeutet dies, dass eine Kante des Dreiecks auf die Länge null schrumpft. Die Länge einer Kante wird als Abstand zwischen zwei Punkten in der Ebene berechnet. Es wird eine neue Gleichung für die Berechnung der Länge einer Kante basierend auf den Gleichungen 2.19 und 2.20 erstellt und auf null gesetzt. Zur Detektierung eines Events sind drei Gleichungen

notwendig, da jede der drei Kanten des Dreiecks auf Länge null schrumpfen und einen Event auslösen kann.

$$\begin{aligned}\sqrt[2]{(M_Ax - M_Bx)^2 + (M_Ay - M_By)^2} &= 0\\ \sqrt[2]{(M_Ax - M_Cx)^2 + (M_Ay - M_Cy)^2} &= 0\\ \sqrt[2]{(M_Bx - M_Cx)^2 + (M_By - M_Cy)^2} &= 0\end{aligned} \tag{2.21}$$

Die kleinste gefundene Lösung wird als ein Event für dieses Dreieck betrachtet.

Leider eignet sich diese Methode nicht zur Erstellung von Straight Skeletons in beliebigen Polygonen, da ausschließlich Edge Events erfolgreich erkannt werden können. Bei einem Split Event kann ein Dreieck zusammenfallen, in dem sich ein Punkt des Dreiecks auf die gegenüberliegende Kante positioniert (Abbildungen 2.6 und 2.7). Dabei schrumpft keine der drei Kanten des Dreiecks auf die Länge null und das Split Event kann nicht detektiert werden.

Gleichung 2: basierend auf dem Abstand zwischen einer Kante und dem gegenüberliegenden Punkt des Dreiecks

Das Kriterium für ein Event ist die Situation, in der sich ein Punkt des Dreiecks beim Schrumpfen des Polygons auf die gegenüberliegende Kante des Dreiecks positioniert. Man wählt eine beliebige Kante des Dreiecks, z.B. die Kante (AB), und berechnet den Abstand zum gegenüberliegenden Punkt (C), basierend auf den Gleichungen 2.19 und 2.20. Zuerst wird die Fläche des Parallelogramms, das über die Vektoren von (AB) und (AC) definiert wird, berechnet.

$$F = |\overrightarrow{AB} \times \overrightarrow{AC}| \tag{2.22}$$

Die Fläche eines Parallelogramms kann auch über die Multiplikation der Länge seiner Kante (AB) zur Höhe (d) des Parallelogramms berechnet werden.

$$F = |\overrightarrow{AB}|\ d \tag{2.23}$$

Nach dem Zusammensetzen der Formeln 2.22 und 2.23 ergibt sich die Höhe des Parallelogramms bzw. der Abstand zwischen der Kante (AB) und dem Punkt (C).

$$d = \frac{|\overrightarrow{AB} \times \overrightarrow{AC}|}{|\overrightarrow{AB}|} \tag{2.24}$$

Nach dem Einsetzen der Koordinaten der Punkte (A), (B) und (C) wird die Gleichung auf null gesetzt und erhält folgende Form:

$$\frac{(M_A y - M_B y)\ M_C x + (M_B x - M_A x)\ M_C y + (M_A x\ M_B y - M_B x\ M_A y)}{\sqrt[2]{(M_B x - M_A x)^2 + (M_B y - M_A y)^2}} = 0 \tag{2.25}$$

Wenn Gleichung 2.25 erfüllt ist, liegt der Punkt (C) auf der Kante (AB) und das Event dieses Dreiecks wurde gefunden.

In diesem Fall besteht keine Notwendigkeit drei Gleichungen pro Dreieck aufzustellen. Die Gleichung 2.25 ist genau dann erfüllt, wenn alle drei Punkte des Dreiecks auf einer Geraden liegen, und deckt somit alle drei möglichen Auswahl-Kombinationen von einem Punkt und einer Kante ab.

Die Drehung von inneren Kanten, die bereits im Kapitel 2.2.5 erwähnt wurde, hat Einfluss auf das Verhalten der Funktion, die in die Gleichung 2.25 definiert ist. Findet eine Drehung der Kante (AB) statt, hat die Gleichung im positiven x-Bereich zwei Lösungen. Eine solche Situation ist in Abbildung 2.9 dargestellt.

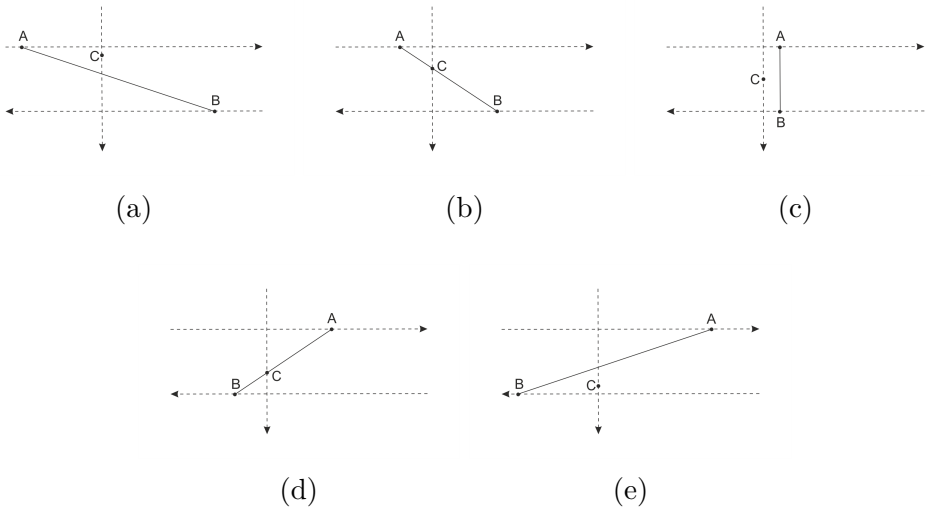


Abbildung 2.9: Drehung einer inneren Kante (AB), Gleichung 2.25

Legende:
Strichliert: Linien der Winkelhalbierenden von einem Punkt
Durchgezogen: Linien der Kanten eines Dreiecks

Der Punkt (A) bewegt sich in Richtung seiner Winkelhalbierenden nach rechts. Punkt (B) bewegt sich entlang seiner Winkelhalbierenden nach links und Punkt (C) wandert entlang seiner Winkelhalbierenden langsam nach unten (Abbildung 2.9a). Nach einem Schrumpfschritt befindet sich der Punkt (C) auf der Geraden (AB) (Abbildung 2.9b). Im nächsten Schritt entfernt sich die Kante (AB) vom Punkt (C) (Abbildung 2.9c). Durch die Drehung der Kante (AB) trifft der Punkt (C) wieder auf diese Kante (Abbildung 2.9d). Im letzen Schritt entfernt sich die Kante (AB) endgültig von dem Punkt (C)(Abbildung 2.9e). Die Abbildungen 2.9b und 2.9d sind daher beide Lösungen der Gleichung 2.25.

Gleichung 3: basierend auf zwei Kanten des Dreiecks

Ein weiteres Kriterium für ein Event ist die Situation, in der zwei Kanten eines Dreiecks die gleiche Neigung beim Schrumpfen des Polygons bekommen. Anders ausgedrückt, positionieren sich die zwei Kanten des Dreiecks übereinander. Beim Auswählen von zwei aus drei Kanten des Dreiecks muss folgendes beachtet werden: Beinhaltet das Dreieck eine oder mehrere innere Kanten, so muss mindestens eine ausgewählte Kante die innere Kante des Polygons sein. Hat das Dreieck keine inneren Kanten (ein Polygon, das nur aus einem Dreieck besteht), werden zwei beliebige Kanten ausgewählt. Diese Bedingung ist wichtig, da die äußeren Kanten des Polygons immer nur parallel verschoben werden und sich ihre Neigung nicht ändert. Diese Eigenschaft trägt dazu bei, dass sich äußere Kanten schlecht für diese Gleichung eignen.
Man nimmt an, die Kanten (AB) und (AC) eines Dreiecks wurden ausgewählt. Eine dieser Kanten ist eine innere Kante des Polygons. Die Neigung (t) für beide Geraden wird aus den Koordinaten der Punkte berechnet.

$$\begin{aligned} t_{AB} &= \frac{M_B y - M_A y}{M_B x - M_A x} \\ t_{AC} &= \frac{M_C y - M_A y}{M_C x - M_A x} \end{aligned} \tag{2.26}$$

Die Neigungen t_{AB} und t_{AC} aus Gleichung 2.26 werden gleichgesetzt.

$$(M_B y - M_A y)(M_B x - M_A x) = (M_C y - M_A y)(M_C x - M_A x) \tag{2.27}$$

$$(M_B y - M_A y)(M_B x - M_A x) - (M_C y - M_A y)(M_C x - M_A x) = 0 \tag{2.28}$$

Ein Event des Dreiecks wurde gefunden wenn die Gleichung 2.28 erfüllt ist bzw. die Kanten (AB) und (AC) die gleiche Neigung besitzen.

Die aufgestellte Gleichung kann nicht als Event-Kriterium dienen, wenn das Polygon nur aus einem Dreieck besteht. So ein Polygon und das einzige Dreieck seiner Triangulierung schrumpfen zu einem Punkt zusammen, was dazu führt das die Neigung der Kante undefiniert ist. Somit ist es unmöglich den Wert der Funktion beim Eintreten eines Events zu berechnen. Diese Eigenschaft beeinflusst auch die Berechnung von Edge Events im allgemeinen Fall. Schrumpft eine der ausgewählten Kanten auf die Länge null, ist ihre Neigung undefiniert und kann somit nicht mit der Neigung einer anderen Kante verglichen werden.

Ähnlich wie in Gleichung 2.25 spielt auch hier die Drehung von inneren Kanten eine Rolle und erhöht die Anzahl der Lösungen im positiven x-Bereich auf zwei. Eine solche Situation ist in Abbildung 2.10 dargestellt.

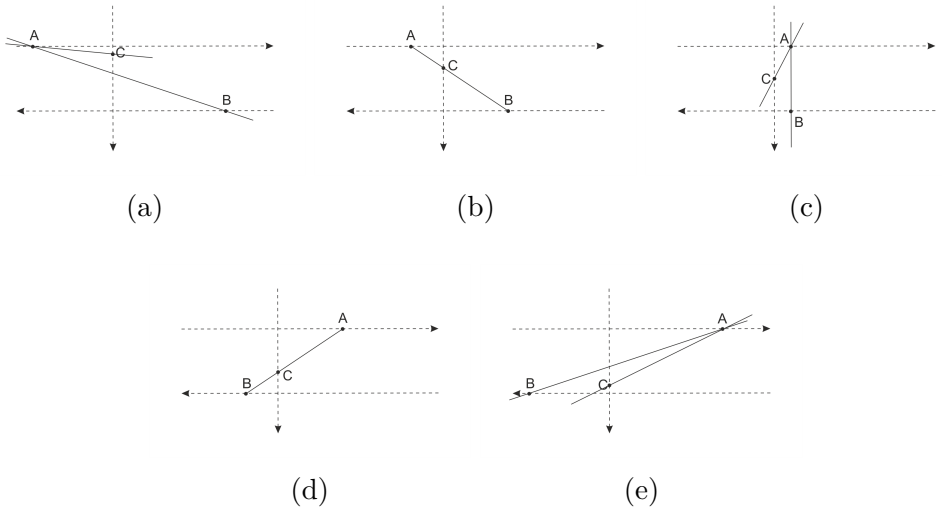


Abbildung 2.10: Drehung einer inneren Kante (AB), Gleichung 2.28

Legende:
Strichliert: Linien der Winkelhalbierenden eines Punkts
Durchgezogen: Linien der Kanten eines Dreiecks

Als Ausgangssituation dient Abbildung 2.10a. Nach einem gewissen Schrumpfschritt haben die Kanten (AB) und (AC) die gleiche Neigung (Abbildung 2.10b). Nach einem weiteren Schrumpfschritt verändert sich die Neigungen der beiden Kanten in unterschiedliche Richtungen (Abbildung 2.10c). Durch die Drehung der Kante (AB) bekommen beide Kanten wieder dieselbe Neigung (Abbildung 2.10d). Im nächsten Schritt verändern sie endgültig ihre

Neigungen in unterschiedliche Richtungen (Abbildung 2.10e). Die Abbildungen 2.10b und 2.10d sind daher die beiden Lösungen von Gleichung 2.28.

Gleichung 4: basierend auf der Fläche des Dreiecks

Das Kriterium für ein Event ist die Situation, in der der Flächeninhalt eines Dreiecks beim Schrumpfen des Polygons null ist. Dieses Kriterium deckt alle Edge-, Split- und Flip-Events ab. Es beinhaltet das Kriterium mit zwei Kanten und das Kriterium mit einer Kante und dem gegenüberliegenden Punkt, da in beiden Fällen die Fläche des Dreieck gleich null ist.

Die Längen der Kanten des Dreiecks werden als Abstand zwischen zwei Punkten in der Ebene berechnet.

$$\begin{aligned} l_{AB} &= \sqrt[2]{(M_A x - M_B x)^2 + (M_A y - M_B y)^2} \\ l_{AC} &= \sqrt[2]{(M_A x - M_C x)^2 + (M_A y - M_C y)^2} \\ l_{BC} &= \sqrt[2]{(M_B x - M_C x)^2 + (M_B y - M_C y)^2} \end{aligned} \tag{2.29}$$

Eine Zwischenberechnung des Hilfsparameters (l) :

$$l = \frac{l_{AB} + l_{AC} + l_{BC}}{2} \tag{2.30}$$

Die Formel für die Berechnung der Fläche des Dreiecks wird basierend auf den Gleichungen 2.29 und 2.30 aufgestellt und gleich null gesetzt.

$$\sqrt[2]{l\ (l - l_{AB})\ (l - l_{AC})\ (l - l_{BC})} = 0 \tag{2.31}$$

Ist der Flächeninhalt des Dreiecks (ABC) gleich null, ist die Gleichung 2.31 erfüllt und ein Event des Dreiecks wurde gefunden.

Wie in den Gleichungen 2.25 und 2.28 spielt auch hier die Drehung der inneren Kanten eine große Rolle und erhöht die Anzahl der Lösungen im positiven x-Bereich auf zwei. Eine solche Situation ist grafisch in Abbildung 2.11 dargestellt.

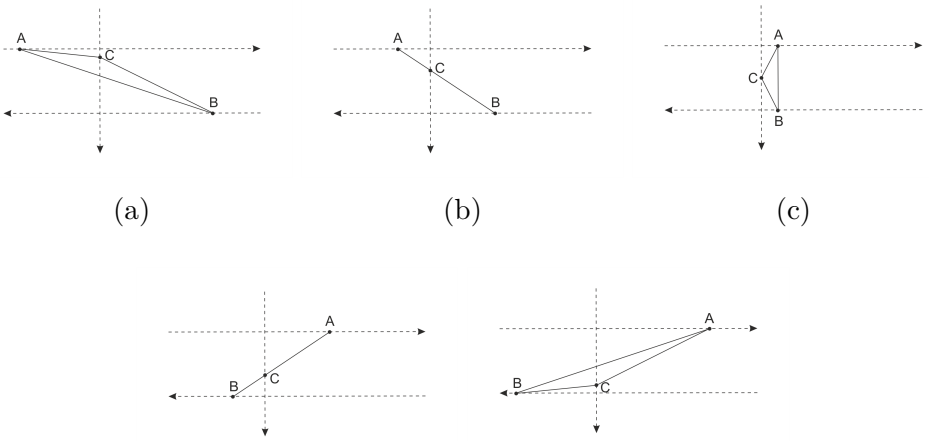


Abbildung 2.11: Drehung einer inneren Kante (AB), Gleichung 2.31

Legende:
Strichliert: Linien der Winkelhalbierenden eines Punkts
Durchgezogen: Linien der Kanten eines Dreiecks

Als Ausgangssituation dient Abbildung 2.11a. Nach einem gewissen Schrumpfschritt liegt der Punkt (C) auf der Geraden (AB) und die Fläche des Dreiecks (ABC) ist gleich null (Abbildung 2.11b). Als nächstes entfernt sich die Gerade (AB) vom Punkt (C) und die Fläche des Dreiecks ist deutlich größer als null (Abbildung 2.11c). Durch die Drehung der Kante (AB) positioniert sich der Punkt (C) wieder auf der Kante (AB) und die Fläche des Dreiecks schrumpft wieder auf null (Abbildung 2.11d). Letztendlich entfernen sich die Punkte endgültig in unterschiedlichen Richtungen voneinander und die Fläche des Dreiecks steigt streng monoton (Abbildung 2.11e). Die Abbildungen 2.11b und 2.11d sind daher beide Lösungen der Gleichung 2.31.

2.2.5 Algorithmus

Algorithmus 2 berechnet das Straight Skeleton eines Polygons auf Basis dessen Triangulierung. Die einzelnen Schritte des Algorithmus sind in Abbildung 2.12 veranschaulicht.

Algorithmus 2 Erstellung von Straight Skeletons mittels Triangulierung

INITIALISIERUNG

Erstelle eine Triangulierung des Polygons.

Berechne für jedes Dreieck alle Flip-, Edge- und Split-Events und füge sie in eine Liste ein.

Sortiere die Liste aufsteigend.

SCHRITT

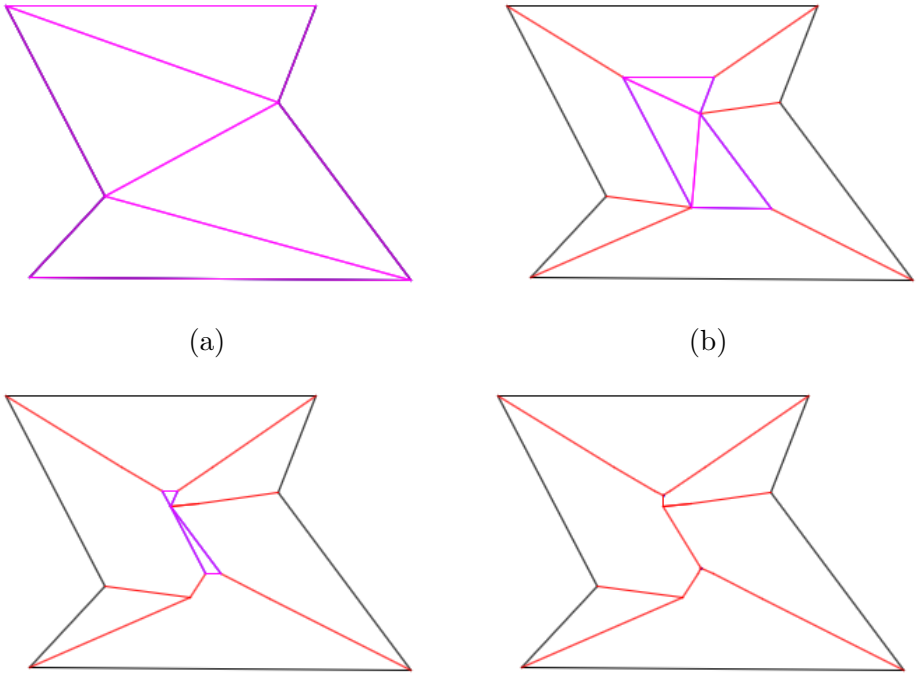
Solange die Triangulierung des Polygons mindestens ein Dreieck mit einer Fläche größer null beinhaltet

Nimm das kleinste Element aus der Liste und schrumpfe das Polygon bis zu diesem Ereignis.

Passe die Triangulierung entsprechend an.

Berechne die Events für neu entstandene Dreiecke .

Füge die neuen Events sortiert in die Liste ein.



(a) (b)

(c) (d)

Abbildung 2.12: Schritt für Schritt Aufbau eines Straight Skeletons mit Hilfe von Algorithmus 2

Legende:
Schwarz: Linien des Basis-Polygons
Rot: Linien des Straight Skeletons
Purpur: Linien der Triangulierung im aktuellen Schritt

Laufzeitabschätzung:

Initialisierung:

- Eine Triangulierung kann in $\mathcal{O}(n\ log(n))$ erstellt werden.
- Die Anzahl von Events entspricht zu diesem Zeitpunkt der Anzahl der Dreiecke der Triangulierung und beträgt $\mathcal{O}(n)$.
- Die Sortierung der Event-Liste kann in $\mathcal{O}(n\ log(n))$ durchgeführt werden.
- Das Lösen der Gleichungen 2-4 wird in die Laufzeitabschätzung nicht miteinbezogen.

Somit kann die Initialisierung des Algorithmus in $\mathcal{O}(n\ log(n))$ durchgeführt werden.

Schritt

- Das Auswählen des nächsten Events und Anpassung der Triangulierung kann in $\mathcal{O}(1)$ durchgeführt werden.
- Die Schritt wird mindestens $\mathcal{O}(n)$ mal wiederholt, da er für jedes Dreieck der Triangulierung ausgeführt wird. Wie bereits erwähnt sinkt im Fall eines Flip Events die Anzahl der Dreiecke der Triangulierung nicht. Potenziell sind $\mathcal{O}(n^3)$ Flip Events in einer Triangulierung möglich. Somit beträgt die obere Schranke für die Anzahl der Wiederholungen $\mathcal{O}(n^3)$.

Als Laufzeit ergibt sich somit $\mathcal{O}(n^3)$ für die Durchführung aller Schritte.

Die gesamte Laufzeit des Algorithmus beträgt $\mathcal{O}(n^3)$.

2.3 Berechnung mittels Polygonpseudotriangulierung

Dieser Algorithmus basiert auf der Pseudotriangulierung eines Polygons. Die Definition, Eigenschaften und den Algorithmus zum Aufbau einer Pseudotriangulierung befinden sich in den Kapiteln 2.3.1 und 2.3.4. Aufgrund der besonderen Eigenschaften von Pseudotriangulierungen verspricht dieser Algorithmus eine durchschnittlich bessere Laufzeit als Algorithmus 2. Im Worst-Case ist die Pseudotriangulierung eine normale Triangulierung und bringt somit keine Performancesteigerung im Vergleich zu Algorithmus 2.

Die Events in diesem Algorithmus werden durch bestimmte Ereignisse in den Pseudodreiecken der Pseudotriangulierung ausgelöst. Die Idee: Da ein Pseudoreieck aus einem oder mehreren Dreiecken besteht, würde das Zusammmenfallen von einem Dreieck der Triangulierung sich in dem entsprechenden Pseudodreieck der Pseudotriangulierung auswirken. Es gibt zwei verschiedene Arten der Events in einem Pseudodreieck. Diese werden in den Kapiteln 2.3.2 und 2.3.3 ausgeführt.

2.3.1 Pseudotriangulierung

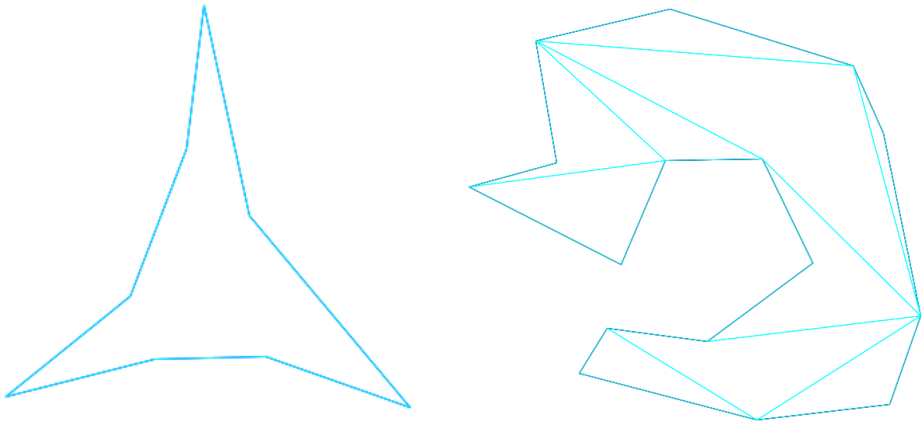
Das Pseudodreieck ist eine relativ neue geometrische Struktur, die ursprünglich von Michel Pocchiola and Gert Vegter in "The visibility complex" [17] vorgestellt wurde. Diese Datenstruktur findet bereits Anwendung in Gebieten wie Sichtbarkeitsgrafen, Collision Detection oder Bewegungen von Gelenksystemen usw..

Definition 5 *Ein Pseudodreieck ist ein Polygon, welches mindestens drei Knoten besitzt, die durch konvexe Ketten verbunden sind. Auf diesen Ketten können also weitere Knoten liegen, die jedoch im Uhrzeigersinn nur Knicke nach rechts darstellen dürfen [4] (Abbildung 2.13a).*

Die Pseudotriangulierung ist wie folgt definiert:

Definition 6 *Eine Pseudotriangulierung ist eine Zerlegung einer planaren Region in Pseudodreiecke [5] (Abbildung 2.13b).*

Im Grunde genommen ist jedes Dreieck auch ein Pseudodreieck, und eine Pseudotriangulierung ist die Verallgemeinerung einer Triangulierung. Die Anzahl von Pseudodreiecken einer Pseudotriangulierung ist keiner gleich der Anzahl von Dreiecken in einer Triangulierung. Da die Laufzeit der Algorithmen



(a) (b)

Abbildung 2.13: Pseudodreieck und Pseudotriangulierung

2 und 5 direkt von der Anzahl der Dreiecke bzw. Pseudodreiecke abhängig ist, sollte der Austausch der Triangulierung durch die Pseudotriangulierung eine Verbesserung der Laufzeiten im Vergleich zu Algorithmus 2 mit sich bringen.

Definition 7 *Eine balancierte Triangulierung ist eine Triangulierung mit folgender Eigenschaft: Kein Liniensegment geschnitten mit dem Polygon schneidet mehr als $\mathcal{O}(log(n))$ Dreiecke der Triangulierung.[5]*

Ein konvexes Polygon ist wie folgt definiert:

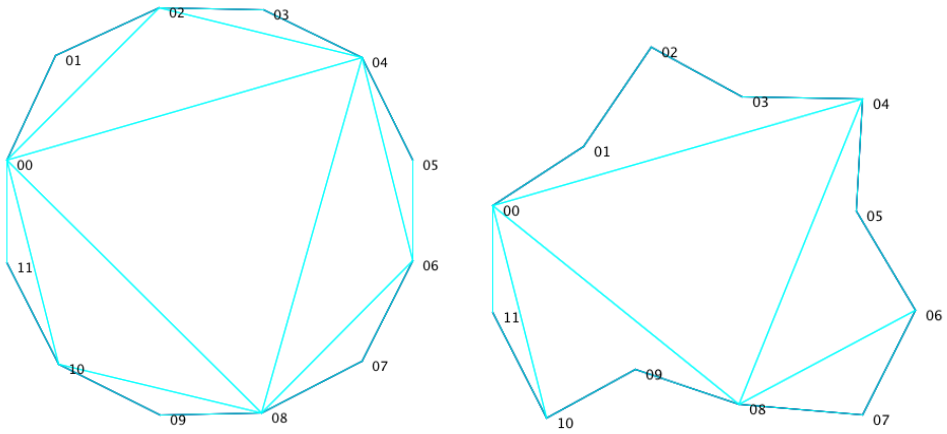
Definition 8 *Gegeben ist ein Polygon. Man nimmt zwei beliebige Punkte innerhalb oder auf den Kanten des Polygons und verbindet sie zur einer Strecke. Verlässt die Verbindungsstrecke an irgendeiner Stelle das Polygon, so ist dieses nicht konvex. Verläuft die Verbindungsstrecke jedes möglichen Punktepaares jedoch innerhalb des Polygons, so ist es konvex.[14]*

Die Erstellung der Pseudotriangulierung eines Polygons verläuft wie folgt: Man nimmt ein konvexes Polygon mit der gleichen Anzahl von Punkten wie das Basis-Polygon und erstellt dafür eine balancierte Triangulierung mit Hilfe von Algorithmus 3.

Algorithmus 3 Erstellung einer balancierten Triangulierung

```
1 /*
  pointList - eine Liste, die alle Punkte des Polygons sortiert im
  Uhrzeigersinn beinhaltet
  triangleList - ein Liste, die alle erstellten Dreiecke beinhaltet und
  somit die Triangulierung darstellt
  */
2 for (i=0; i < pointList.size; i=i+2) do
3     triangleList.add(pointList.get(i),pointList.get(i+1),
  pointList.get(i+2));
4     pointList.add(pointList.get(i));
5 end for
```

Nach jeder Iteration der For-Schleife werden ausgewählte Punkte, welche bereits in der pointList vorhanden sind, nochmals in die Liste eingefügt. Die Triangulierung wird somit von außen nach innen aufgebaut. Der Algorithmus bricht ab, wenn die Liste pointList weniger als drei nicht verarbeitete Punkte beinhaltet. Das Ergebnis ist in Abbildung 2.14a dargestellt.



(a) (b)

Abbildung 2.14: Balancierte Triangulierung und Pseudotriangulierung

Im nächsten Schritt überträgt man die erstellte Triangulierung auf das Basis-Polygon. Dabei betrachtet man alle Kanten der Triangulierung als eine Art

Gummibänder, die sich an die Form des Polygons anpassen können. Einige Dreiecke können dabei vollständig zu einer Linie zusammenfallen oder befinden sich außerhalb des Polygons. Mehrere Beispiele dafür sind in Abbildung 2.14b dargestellt. Die Dreiecke (00-01-02), (02-03-04), (04-05-06) und (08-09-10) befinden sich außerhalb des Polygons und wurden deswegen aus der Pseudotriangulierung entfernt. Die Kanten der anderen Dreiecke bekommen dabei neue konkave Ecken und bilden somit die Pseudodreiecke. In Abbildung 2.14b beinhaltet das Pseudodreieck (00-02-04) jetzt auch die Punkte (01) und (03). Das Pseudodreieck (04-06-08) beinhaltet den Punkt (05) und letztendlich beinhaltet das Pseudodreieck (00-08-10) den Punkt (09). Somit entsteht eine Pseudotriangulierung des ursprünglichen Polygons (Abbildung 2.14b).

In “Pseudodreiecks-Zerlegung” [4] werden zwei weitere Algorithmen zum Aufbau von Pseudotriangulierungen vorgestellt.

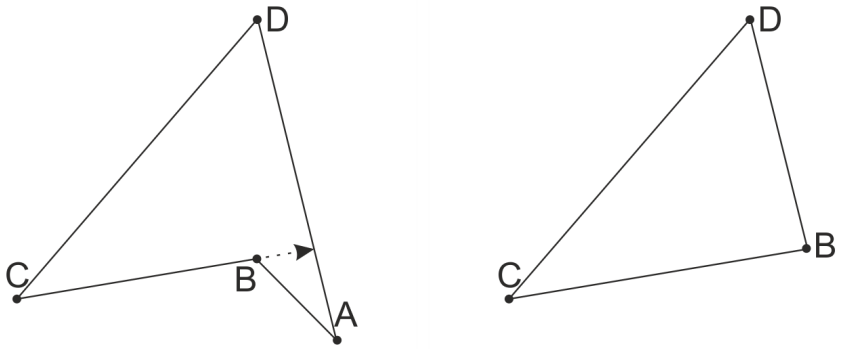
Der Pseudocode des Algorithmus:

Algorithmus 4 Erstellung einer Pseudotriangulierung aus einer balancierten Triangulierung

```
1  /* P - Polygons */
2  for all (Pseudodreiecke) do
3      if ((Pseudodreieck liegt vollständig außerhalb von P) ∨ (ist die Fläche des Pseudodreiecks gleich null)) then
4          Entferne Pseudodreieck
5      end if
6  end for
7  Erstelle eine Liste mit Kanten der Pseudodreiecke
8  for all (Kanten des Pseudodreiecks) do
9      Teile die Punkte des Polygons mit der Kante in zwei Gruppen
10     Erstelle aus jeder Gruppe ein Polygon
11     for all (Punkte aus Gruppe 1) do
12         Suche einen Punkt, der im Polygon der Gruppe 2 liegt und die maximale Distanz zur Kante hat
13         if (Gibt es so einen Punkt) then
14             Entferne diese Kante aus der Liste
15             Teile die Kante mit dem Punkt in zwei Kanten auf
16             Füge sie in die Liste ein
17         end if
18     end for
19     for all (Punkte aus Gruppe 2) do
20         Suche einen Punkt, der im Polygon der Gruppe 1 liegt und die maximale Distanz zur Kante hat
21         if (Gibt es so einen Punkt) then
22             Entferne diese Kante aus der Liste
23             Teile die Kante mit dem Punkt in zwei Kanten auf
24             Füge sie in die Liste ein
25         end if
26     end for
27 end for
```

2.3.2 Edge Event

Ein Edge Event tritt genau dann auf, wenn der konvexe Winkel (BAD) eines Pseudodreiecks ($ABCD$) beim Schrumpfen des Polygons einen Winkel von 0° bekommt. Anders gesagt klappt die konvexe Ecke einfach zusammen (Abbildung 2.15).



(a) (b)

Abbildung 2.15: Edge Event

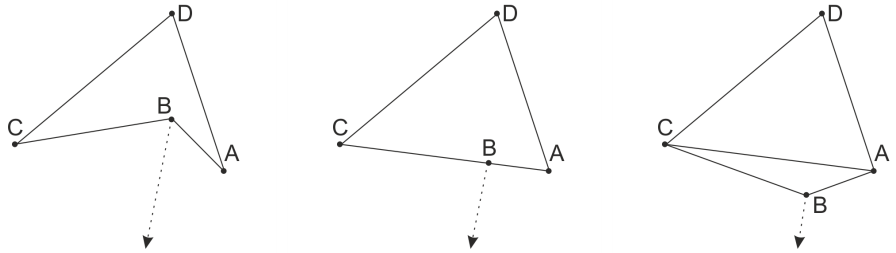
Da ein Pseudodreieck immer genau drei konvexe Ecken hat, bekommt es eine neue konvexe Ecke (CBD) und verliert eine konkave Ecke (ABC). Bei jedem Edge Event in einem Pseudodreieck reduziert sich somit die Anzahl der Ecken.

Die Berechnung von Edge Events kann mit Hilfe von den in Kapitel 2.2.4 vorgestellten Gleichungen durchgeführt werden. Jede konvexe Ecke des Pseudodreiecks wird als ein unabhängiges Dreieck betrachtet und in die Gleichung für die Event-Suche eingesetzt. Somit ergeben sich pro Pseudodreieck drei Gleichungen mit drei unterschiedlichen Lösungen. Die kleinste Lösung ist das nächste Edge Event dieses Pseudodreiecks.

Im letzten Schritt des Schrumpfens hat ein Pseudodreieck nur noch drei Ecken und wird somit zu einem normalen Dreieck. Damit besteht keine Notwendigkeit eine Gleichung für jede konvexe Ecke aufzustellen. Die Suche nach dem Edge Event wird nach dem gleichen Prinzip wie in Algorithmus 2 durchgeführt.

2.3.3 Split Event

Als Basis für ein Split Event dienen die konkaven Ecken eines Pseudodreiecks. Als Ausgangssituation wird ein Pseudodreick ($ABCD$) mit konkavem Winkel (ABC) genommen, dessen Winkelhalbierende nicht in das Innere des Pseudodreiecks verläuft (Abbildung 2.16a). Ein Split Event tritt genau dann auf, wenn der Winkel einer konkaven Ecke (ABC) des Pseudodreiecks ($ABCD$) beim Schrumpfen des Polygons die Größe von 180° erreicht (Abbildung 2.16b). Beim Auftreten des Events liegen die Punkte (A), (B) und (C) auf einer Geraden.



(a) (b) (c)

Abbildung 2.16: Split Event

Beim weiteren Schrumpfen des Polygons bewegt sich der Punkt (B) weiter entlang seiner Winkelhalbierenden und würde somit die Struktur des Pseudodreiecks zerstören. Um die Struktur des Pseudodreiecks zu erhalten, muss vom Pseudodreieck ($ABCD$) das Dreieck (ABC) abgespaltet werden (Abbildung 2.16c). Dadurch verliert das Pseudodreieck ($ABCD$) eine konkave Ecke (ABC) und es entsteht ein neues Dreieck (ABC). Falls (ABC) die letzte konkave Ecke des Pseudodreiecks ist, entstehen aus diesem Pseudodreieck zwei Dreiecke.

Jedes Split Event führt somit auch zur einer Reduktion der Anzahl an Ecken im Ursprungs-Pseudodreieck, erhöht dabei aber die Gesamtzahl der Pseudodreiecke, in diesem Fall durch ein Dreieck repräsentiert.

Für die Berechnung von Split Events werden auch die Gleichungen aus Kapitel 2.2.4 eingesetzt. In diesem Fall ist jede konkave Ecke des Pseudodreiecks ein unabhängiges Dreieck und wird in die Gleichung für die Event-Suche eingesetzt. Die Anzahl konkaven Ecken in einem Pseudodreieck ist durch keine Kriterien begrenzt und kann somit beliebig groß sein, dementsprechend variiert auch die Anzahl der Gleichungen und deren Lösungen. Die kleinste Lösung ist das nächste Split Event dieses Pseudodreiecks.

2.3.4 Algorithmus

Algorithmus 5 berechnet das Straight Skeleton eines Polygons auf Basis der Pseudotriangulierung. Die einzelnen Schritte des Algorithmus sind in Abbildung 2.17 veranschaulicht.

Algorithmus 5 Erstellung von Straight Skeletons mittels Pseudotriangulierung

INITIALISIERUNG

Erstelle eine balancierte Pseudotriangulierung des Polygons.

Berechne für jedes Pseudodreieck ein Event und füge es in eine Liste ein.

Sortiere die Liste aufsteigend.

SCHRITT

Solange die Pseudotriangulierung des Polygons mindestens ein Dreieck mit einer Fläche größer null beinhaltet

Nimm das kleinste Element aus der Liste und schrumpfe das Polygon bis zu diesem Ereignis.

Passe die Pseudotriangulierung entsprechend an.

Berechne die neuen Events für die angepassten Pseudodreiecke.

Füge die neuen Events sortiert in die Liste ein.

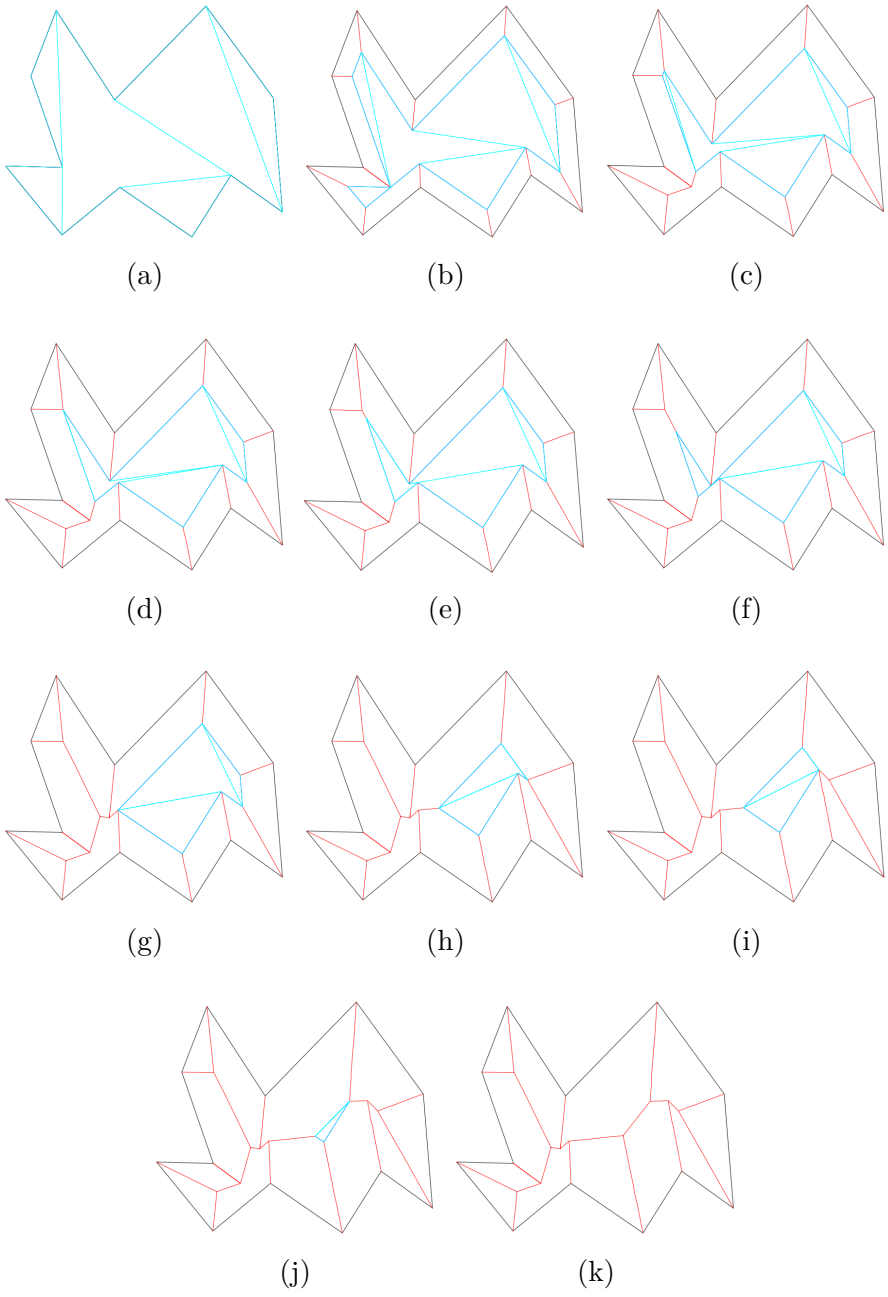


Abbildung 2.17: Schritt für Schritt Aufbau eines Straight Skeletons mit Hilfe von Algorithmus 5

Legende:
Schwarz: Linien des Basis-Polygons
Rot: Linien des Straight Skeletons
Cyan: Linien der Pseudotriangulierung im aktuellen Schritt

Kapitel 3

Software

Das Unterkapitel 3.1 bietet einen ausführlichen Überblick über die grafische Oberfläche des Programms und beinhaltet detaillierte Informationen zur dessen Bedienung. Die Beschreibung der Implementierung des Programms befindet sich im Unterkapitel 3.2.

3.1 Programmbeschreibung

Die Mindestanforderungen für die Ausführungg des Programms um das Programm auszuführen sind:

- Betriebssystem
  - Linux mit Kernel 2.6.x oder höher
  - Mac OS X 10.6 oder höher
  - Windows XP oder höher
- Sun JRE 1.7 oder höher, siehe [19]
- 2MB freier Festplattenspeicher

3.1.1 Bedienungsanleitung

Dieses Programm dient vor allem als Hilfsmittel zur Untersuchung der Datenstruktur Straight Skeleton. Es verfügt über eine sehr einfache, intuitive und benutzerfreundliche grafische Oberfläche. Durch umfangreiche farbliche Konfigurationsmöglichkeiten kann die grafische Benutzerschnittstelle den Benutzervorlieben angepasst werden und ermöglicht somit ein angenehmes Arbeiten bei der Untersuchung von Straight Skeletons (Abbildung 3.1).

Die grafische Oberfläche des Programms wurde in zwei Bereiche aufgeteilt. Der große weiße Bereich auf der linke Seite ist ein grafischer Editor, der die schnelle Erstellung von Polygonen ermöglicht. Die Eingabe von Punkten des Polygons erfolgt durch einen einfachen Maus-Klick. Die Punkte können entweder im oder entgegen des Uhrzeigersinns eingegeben werden. Die Eingabe eines Polygons wird durch einen einfachen Maus-Klick in die Nähe des ersten Punktes des Polygons beendet. Sich schneidende Kanten des Polygons sind nicht erlaubt und werden vom Programm nicht akzeptiert.

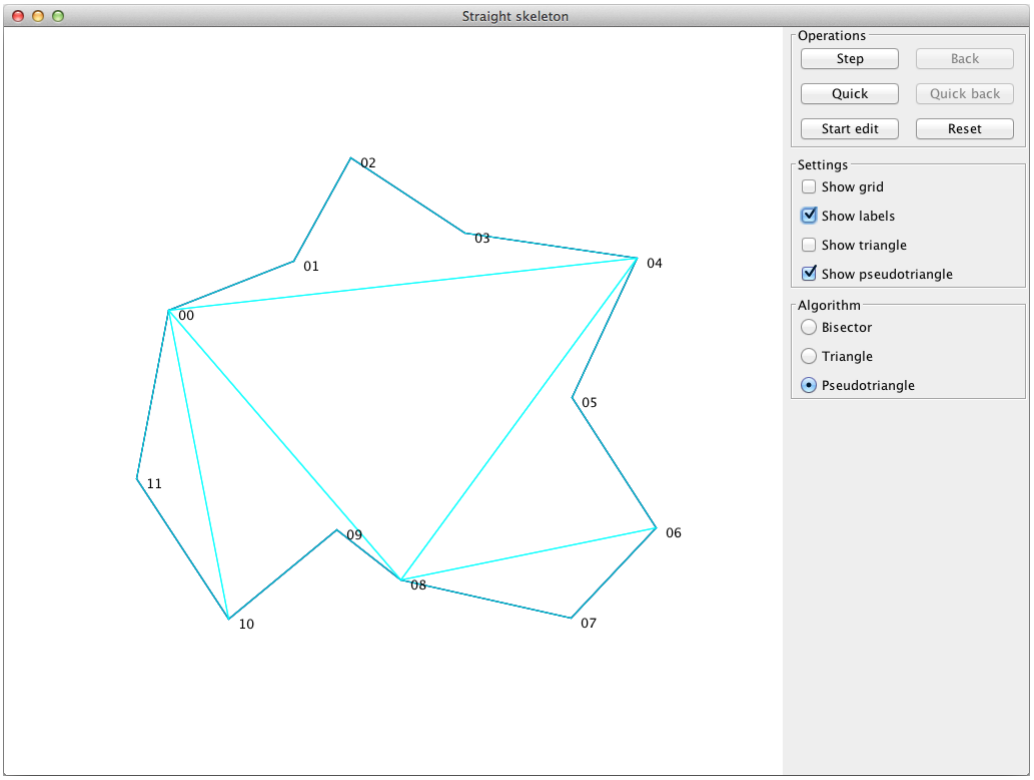


Abbildung 3.1: Programm zur Erstellung von Straight Skeletons

Der rechte Bereich ist in mehrere Bereiche mit diversen Funktionalitäten aufgeteilt.

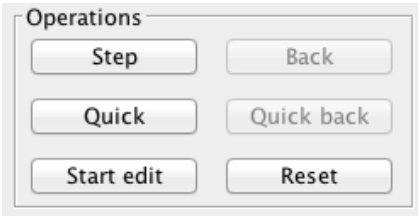


Abbildung 3.2: Operations Panel

Der wichtigste Bereich nach dem Editor ist das "Operations Panel", welches in Abbildung 3.2 dargestellt ist. Es dient zum schnellen oder Schritt-für-Schritt Aufbau von einem Straight Skeleton. Jeder bereits durchgeführte Schritt kann auch rückgängig gemacht werden. Diese Operation ermöglich ergleiche der Situationen vor und nach dem Schritt anzustellen, was sehr wichtig ist für eine einfache und schnelle Analyse der Änderungen, die durch einen Schritt hervorgerufen wurden.

Eine zusätzliche Besonderheit von diesem Panel ist der "Edit Modus". Gelingt es nicht, bei der Eingabe des Polygons die Punkte in die richtige Konstellation zu bringen, können sie mit Hilfe des "Edit Modus" nachträglich verschoben werden. Durch die wiederholte Verschiebung einzelner Punkte kann das Polygon nach und nach in die gewünschte Form gebracht werden. Bereits durchgeführte Schritte zum Aufbau des Straight Skeltons werden beim Aktivieren des "Edit Modus" verworfen. Das ist notwendig, da durch die Änderung der Form des Polygons sich auch sein Straight Skeleton ändert.

Die Buttons auf dem Panel werden entsprechend der jeweiligen Situation zum aktuellen Zeitpunkt automatisch aktiviert oder deaktiviert. Dies erhöht die Benutzerfreundlichkeit, da man nur erlaubte Aktionen durchführen kann und keine entsprechenden Fehlerdialoge den Benutzer überlasten.

Abbildung 3.3: Settings Panel

Das Panel "Settings" ist für zusätzliche Darstellungsmöglichkeiten zuständig und ist in Abbildung 3.3 dargestellt. Es bietet z.B. die Möglichkeit, ein Raster einzublenden, damit sich der Benutzer beim Erstellen des Polygons besser orientieren kann. Diese Funktion ist zu jedem Zeitpunkt verfügbar. Die Labels der Punkte können auch nach Bedarf jederzeit ein- bzw. ausgeblendet werden. Um die Schritte der Algorithmen, welche auf einer Triangulierung bzw. einer Pseudotriangulierung basieren, besser zu verstehen, kann die entsprechende Struktur eingeblendet werden. Die Darstellung der Triangulierung bzw. der Pseudotriangulierung ist vom ausgewählten

Algorithmus unabhängig und verfügbar, sobald die vollständige Definition des Polygons vorliegt. Der “Edit Modus” stellt hierfür eine Ausnahme dar.

Das “Algorithm Panel” dient zur Auswahl des Algorithmus der zum Aufbau von Straight Skeleton verwendet werden soll und ist in Abbildung 3.4 dargestellt. Er kann in jedem Aufbauschritt neu ausgewählt werden. Somit ist dieses Panel ab dem Zeitpunkt der vollständigen Definition des Polygons verfügbar. Der “Edit Modus” stellt auch hier eine Ausnahme dar.

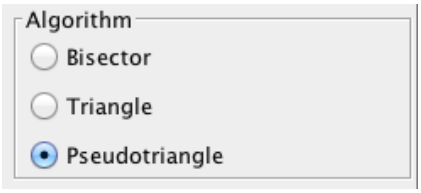


Abbildung 3.4: Algorithm Panel

3.1.2 Polygondateiformat

Um ein bereits erstelltes Polygon immer wieder verwenden zu können, verfügt das Programm über die Möglichkeit, eine erstellte Konfiguration zu speichern bzw. wieder zu laden. Die entsprechenden Operationen befinden sich in Menü “File”. Die Operation “Save” ist erst verfügbar sobald die vollständige Definition des Polygons vorliegt. Die Operation “Open” steht durchgehend zur Verfügung. Auch hier stellt der “Edit Modus” eine Ausnahme dar.

Die Koordinaten der einzelnen Punkte werden in der Eingabereihenfolge als Ganzzahlen in einer Textdatei gespeichert. Jede Zeile entspricht einem Punkt des Polygons, welcher durch seine x und y Koordinate, getrennt durch ein Leerzeichen, angegeben wird. Folgendes Beispiel veranschaulicht das einfache Dateiformat:

```
256 184
444 105
387 321
568 363
326 549
149 434
```

Der Dateiname muss die Endung “ske” haben, um vom Programm geöffnet werden zu können.

3.2 Implementierung

Im Unterkapitel 3.2.1 werden die wichtigsten Aspekte der Softwarearchitektur dieses Projektes vorgestellt. Und in 3.2.2 befinden sich ausführliche Informationen zu den Problemen und Entscheidungen, welche im Laufe der Implementierung aufgetreten sind bzw. getroffen wurden.

3.2.1 Softwarearchitektur

Eine zentrale Anforderung an das Programm ist die Möglichkeit den Algorithmus zur Laufzeit in jedem Schritt ändern zu können. Zur Erfüllung dieser Anforderung wurde beim Softwaredesign darauf geachtet, eine einfache und leicht erweiterbare Lösung zu finden. Bei der Analyse der möglichen Designansätze wurde das Entwurfsmuster Strategie ausgewählt . Es erfüllt alle gestellten Anforderungen und wurde somit in die Implementierung übernommen.

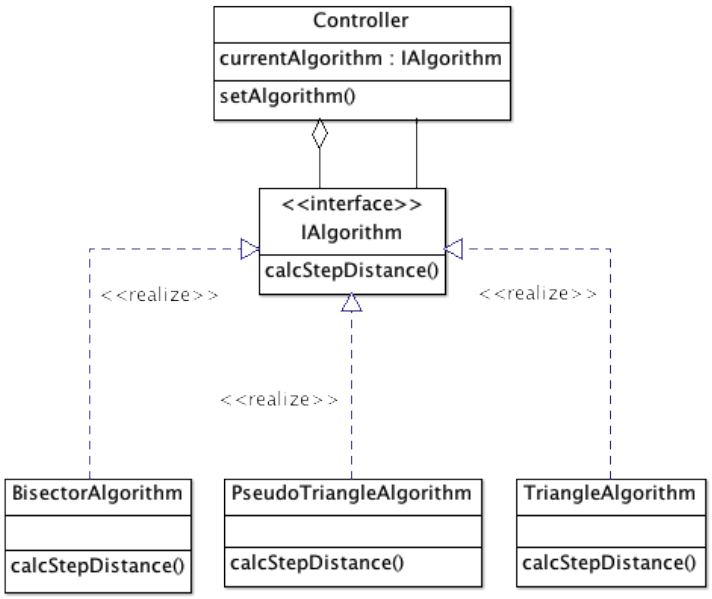


Abbildung 3.5: Strategie Pattern

Das Interface “IAlgorithm” definiert nur eine Schnittstelle für alle implementierten Algorithmen. Die Implementierung der eigentlichen Algorithmen befindet sich jeweils in den Klassen “BisectorAlgorithm”, “PseudoTriangleAlgorithm” und “TriangleAlgorithm”. Die Klasse “Controller” hat eine Mem-

bervariab vom Typ “IAlgorithm” mit der Referenz auf die konkrete Implementierung des Algorithmus. Dadurch wird der konkrete Algorithmus nur über die Schnittstelle eingebunden und kann somit auch zur Laufzeit des Programmes bequem per Mausklick dynamisch gegen eine andere Implementierung ausgetauscht werden. Der Mausklick ruft in diesem Fall die Methode “setAlgorithm()” auf und setzt damit den gewünschten Algorithmus, der bei der Berechnung des nächsten Schrittes verwendet wird.

3.2.2 Implementierungsdetails

In diesem Kapitel werden folgende Themen behandelt: Änderungen an den Algorithmen und deren Zusammenführung als Vorbereitungen zur Implementierung des Programms. Eine Analyse verschiedener Kombinationen von mehreren Events, welche zum gleichen Zeitpunkt bzw. im gleichen Punkt auftreten, sowie die Analyse von Situationen mit parallelen gegenüberliegenden Kanten. Am Ende des Kapitels wird die Wahl der Gleichung zur Event-Detektierung begründet und ein geeignetes Näherungsverfahren vorgestellt.

Implementierung der Algorithmen

Die Implementierung der Algorithmen weicht etwas von den in den Kapiteln 2.1.3, 2.2.5 und 2.3.4 beschriebenen Algorithmen ab. Alle drei Algorithmen haben die gleiche Struktur, welche aus der Initialisierung und einem sich wiederholenden Schritt besteht. Diese Struktur wurde etwas abgeändert, um dem Programm zusätzliche Flexibilität zu geben. Die Sortierung der Events wurde aus der Initialisierung entfernt. Die Suche nach dem nächsten Event wurde in den Schritt des jeweiligen Algorithmus verschoben und wird vor jedem Schritt erneut durchgeführt. Die Notwendigkeit einer sortierten Liste mit Events entfällt damit. Die durchgeführten Änderungen werden beispielhaft in Algorithmus 6 veranschaulicht.

Algorithmus 6 Erstellung von Straight Skeletons über die Winkelhalbierenden mit veränderter Initialisierung.

INITIALISIERUNG

Bestimme die Winkelhalbierenden für alle Punkte des Polygons.

SCHRITT

Solange der Flächeninhalt des Polygons größer null ist

Berechne alle Edge- und Split Events und nimm das kleinste Event.

Schrumpfe das Polygon bis zu diesem Ereignis.

Berechne die Winkelhalbierenden für alle neuen Ecken, die durch das Event entstanden sind.

Eine weitere Besonderheit ist die Erweiterung der Initialisierung, sodass die Initialisierungsschritte für alle drei Algorithmen gleich durchgeführt werden. Somit werden nach der Eingabe des Polygons die Winkelhalbierenden, die Triangulierung und die Pseudotriangulierung des Polygons erstellt. Unabhängig vom ausgewählten Algorithmus werden die erstellten Strukturen in jedem Schritt des Algorithmus entsprechend angepasst bzw. aktualisiert. Man könnte sagen, dass alle drei Algorithmen zu einem Algorithmus 7 verknüpft wurden. Die Wahl des Algorithmus beeinflusst also nur die Art der Berechnung des nächsten Schrittes beim Schrumpfen des Polygons.

Algorithmus 7 Erstellung von Straight Skeletons mit allen drei Algorithmen

INITIALISIERUNG

Bestimme die Winkelhalbierenden für alle Punkte des Polygons, erstelle eine Triangulierung und eine Pseudotriangulierung des Polygons.

SCHRITT

Solange der Flächeninhalt des Polygons größer null ist

Berechne entsprechend dem gewählten Algorithmus alle Events und nimm das kleinste Element.

Schrumpfe das Polygon bis zu diesem Ereignis.

Berechne die Winkelhalbierenden für alle neuen Ecken, die durch das Event entstanden sind, aktualisiere die Triangulierung und die Pseudotriangulierung des Polygons.

Die durchgeführten Änderungen der Algorithmen in Kombination mit der zusätzlichen Speicherung der Daten wurden durch das in der Softwareentwicklung bekannte "Strategy Pattern" realisiert und ermöglichen einen Wechsel der Algorithmen in jedem Schrumpfschritt.

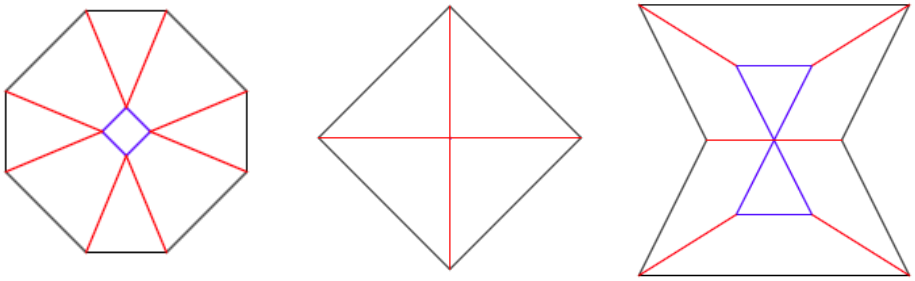
Leider bringen die durchgeführten Änderungen auch eine gewisse Verschlechterung der Laufzeit mit sich. Sie wird durch die Verschiebung der Suche nach dem nächsten Event und der Erweiterung des Initialisierungschrittes verursacht. Bei einem Test mit 20 Punkten lässt sich der Performanceverlust nicht wahrnehmen und wird wahrscheinlich erst bei größeren Datenmengen wahrnehmbar sein.

Mehrere Events in einem Punkt

In diesem Kapitel werden nur Edge- und Split Events aus den Kapiteln 2.1.1 und 2.1.2 betrachtet, da sie die Änderungen der Struktur des Polygons beim Aufbau eines Straight Skeletons repräsentieren. Die restlichen Events sind in diesem Fall nicht relevant, da sie die Ereignisse im Inneren des Polygons widerspiegeln.

Nach der Durchführung eines Schrittes müssen die Daten vom Polygon entsprechend angepasst werden. Im Falle eines Edge Events wird eine Kante

entfernt und zwei Punkte mit den gleichen Koordinaten werden zu einem Punkt vereinigt. Die Anpassung der Daten des Polygons im Falle von mehreren Edge Events in einem Punkt wird nach dem gleichen Prinzip durchgeführt und stellt keine besondere Herausforderung dar.



(a) (b) (c)

Abbildung 3.6: Kombinationen aus mehreren Events

a) 4 Edge Events mit dem gleichen Schrumpfschritt, aber in unterschiedlichen Punkten
b) 4 Edge Events in einem Punkt
c) 2 Split Events in einem Punkt

Im Falle eines Split Events werden die Punkte und Kanten des Polygons entsprechend in zwei Polygone aufgeteilt. Die Triangulierung und Pseudotriangulierung des Polygons wurden bereits während der Initialisierung erstellt und müssen somit auch zwischen den zwei entstandenen Polygonen aufgeteilt werden. Bei mehreren Split Events in einem Schritt müssen alle oben genannten Daten des Polygons entsprechend in mehrere Polygone aufgeteilt werden. Falls ein oder mehre Edge Events in dem gleichen Punkt stattfinden, wird die Anpassung des Polygons sehr komplex. Da mehrere Events in einem Punkt sehr selten auftreten, wurde aufgrund der hohen Komplexität entschieden, dass dieses Problem im Programm nicht behandelt wird, da es auf die Ausführung der Algorithmen keinen Einfluss hat und nur die Anpassungen des Polygons betrifft. Einige Beispiele mit mehreren Edge- bzw. Split Events in einem Punkt sind in Abbildung 3.6 dargestellt.

Bei der Arbeit mit dem Programm müssen solche Situationen also vermieden werden. Die Konfigurationen mit mehreren Events in einem Punkt können im “Edit Modus” meist durch kleine Verschiebungen der betroffenen Punkte aufgelöst werden. Wurden die Verschiebungen richtig durchgeführt, ändern sich nur die Koordinaten des Events, die Struktur des Straight Skeletons bleibt dabei unverändert.

Parallele gegenüberliegende Kanten

Im allgemeinen Fall beeinflusst die Parallelität von einigen Kanten des Polygons die Ausführung der drei Algorithmen keinesfalls. Allerdings gibt es eine wichtige Ausnahme, auf die in diesem Kapitel im Detail eingegangen wird.

Die Ausnahme bilden parallele gegenüberliegende Kanten von zwei Split Events. Ein Beispiel ist in Abbildung 3.6c dargestellt. Situationen dieser Art bereiten Algorithmus 1, der auf den Winkelhalbierenden des Polygons basiert, gewisse Schwierigkeiten.

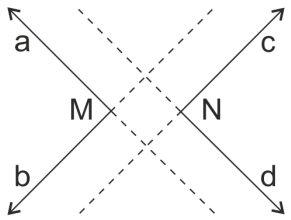


Abbildung 3.7: Split Events mit paarweise parallelen Kanten

Wie in Kapitel 2.3 bereits beschrieben, werden bei der Berechnung eines Split Events beide Kanten der konkaven Ecke in das Innere des Polygons verlängert bis sie auf die Kante des Polygons treffen.

In dieser Situation sind die Kanten von zwei konkaven Ecken (M) und (N) paarweise parallel (Abbildung 3.7). Die Kante (a) ist parallel zur Kante (d) und Kante (b) ist parallel zur Kante (c). Bei der Verlängerung von Kanten wird immer nur ein Schnittpunkt mit der gegenüberliegenden Kante gefunden und nicht zwei. Somit können einige Split Events nicht detektiert werden. Die anderen vorgestellten Algorithmen arbeiten mit Dreiecken bzw. Pseudodreiecken des Polygons und sind somit von diesem Problem nicht betroffen. Split Events dieser Art können mit diesen Algorithmen problemlos gefunden werden.

Lösen der Gleichungen zur Eventdetektion

Der wichtigste Schritt bei der Ausführung der Algorithmen aus dem Kapiteln 2.1.3 und 2.2.5 ist die Lösung einer der drei Gleichungen 2.25, 2.28 oder 2.31. Nach der Zusammenführung sind die Gleichungen relativ groß und unübersichtlich. Die gesuchte Variable (h) kommt immer in den gleichen Konstruktionen vor. Die Variable (h) hat immer die Potenz zwei und befindet sich mit mehreren anderen Parametern unter einer Wurzel. Diese Konstruktion verhindert das Umformen der Gleichung in kanonische Form. Ein Umformungsversuch mit Hilfe der Programme Mathematica und Scilab war auch nicht erfolgreich. Damit ist die klassische Lösung der Gleichung leider nicht möglich.

Daher wurde entschieden ein Näherungsverfahren zur Lösung der Gleichungen zu verwenden. Bei der Wahl des Mathematikprogramms zur Lösung der Gleichung ist die Entscheidung auf Octave gefallen. Octave ist sehr einfach in der Bedienung, schnell und erlaubt die Verwendung von benutzerdefinierten Funktionen. Außerdem existiert eine frei verfügbare Bibliothek für die Programmiersprache Java, die es erlaubt mathematische Probleme, die in Oktave modelliert wurden, in Java zu lösen.

Wie in Kapitel 2.2.4 bereits erwähnt, hat die Gleichung 2.19 immer zwei Lösungen, die sich durch das Vorzeichen (+ oder -) in der Formel unterscheiden. Während der Zusammensetzung der Formel ist unbekannt, welche der zwei Lösungen sich im Polygon befindet. Somit müssen beide Möglichkeiten berücksichtigt werden. Bei einer Kombination aus drei Punkten eines Dreiecks ergeben sich also sechs mögliche Lösungen. Um keine der Lösungen zu verlieren, müssen mehrere Gleichungen mit allen vorhandenen Punktkombinationen aufgestellt werden. Im Fall von Gleichung 2.25 ergeben sich 4 Möglichkeiten für die Geradengleichungen und 2 Möglichkeiten für den Punkt, somit insgesamt 8 benutzerdefinierte Funktionen, die jeweils in einer Text-Datei abgespeichert werden. Bei den Gleichungen 2.28 und 2.31 ergeben sich dementsprechend 16 und 8 benutzerdefinierte Funktionen. Jede der Funktionen wird dann aus dem Java-Programm geladen und aufgerufen. Jede gefundene Lösung wird auf die Positionierung im Polygon überprüft. Als Event wird die Lösung mit der kleinsten Schrittgröße ausgewählt.

In der Testphase wurde ein sehr hoher Performanceverlust festgestellt. Bereits bei einer Punktmenge der Größe 5 haben die Berechnungen ca. 3-4 Sekunden gedauert. Die Arbeit mit dem Programm wurde dadurch sehr mühsam. Bei der Analyse hat sich herausgestellt, dass das Öffnen und Schließen von Textdateien mit den benutzerdefinierten Funktionen, und die Übergabe dieser an Octave sehr viel Zeit in Anspruch nimmt. Teilweise wurden komplexe Lösungen gefunden, die nicht verwendet werden können. Hin und wieder geriet Octave bei der Suche nach der Lösung in eine Endlosschleife. In diesem Fall musste die Arbeit der Octave-Bibliothek abgebrochen und neu gestartet werden, was zu zusätzlichem Zeitverlust geführt hat.

Nach der Analyse der Funktionen aus den Gleichungen 2.25, 2.28 und 2.31 wurde entschieden, ein eigenes, einfacheres Näherungsverfahren zu implementieren, welches an die Besonderheiten der ausgewählten Gleichung angepasst wurde. Die Wahl der Gleichung und der Implementierung des Näherungsverfahrens wird in den nächsten zwei Kapiteln im Detail beschrieben.

Auswahl der geeignetsten Gleichung

Evaluierung der Gleichung 2.21

Wie bereits erwähnt wurde, werden mit dieser Gleichung nur die Edge Events und keine Split Events gefunden. Somit eignet sich diese Gleichung nicht für die Implementierung.

Evaluierung der Gleichung 2.28

Die Funktion dieser Gleichung kann in Punkten, an denen Events stattfinden, undefiniert sein. Situationen dieser Art können natürlich während der Implementierung extra behandelt werden, bedeuten aber zusätzlichen Aufwand und erhöhte Komplexität der Implementierung des Näherungsverfahrens. Da es aber bessere Verfahren gibt, wird diese Gleichung nicht weiter betrachtet.

Evaluierung der Gleichungen 2.25 und 2.31

Bei der Wahl zwischen diesen zwei Gleichungen wurden die Charakteristika der Funktionen in Kombination mit dem Algorithmus, basierend auf der Triangulierung des Polygons, analysiert. Untersucht wurden die folgenden zwei Situationen:

In der ersten Situation besteht das Polygon aus nur drei Punkten, somit sind alle äußeren Kantens des Polygon auch die Kanten der Triangulierung. Die Eingangsdaten für beide Gleichungen wurden gleich gewählt. Die Ergebnisse sind in Abbildungen 3.8 und 3.10 dargestellt. Wie man sieht ist die Funktion der Gleichung 2.25 in zwei Bereiche aufgeteilt: streng monoton fallend bis zum Eintreten des Events und danach streng monoton steigend. Die Funktion hat nur eine Nullstelle im positiven x-Bereich, die dem Event des Dreiecks entspricht. Die Funktion der Gleichung 2.31 ist eine konvexe Parabel mit einem Minima als Nullstelle. Ein Näherungsverfahren ist demnach für beide Funktionen gut realisierbar.

Die zweite Situation besteht aus einem Dreieck, welches ausschließlich aus inneren Kanten besteht. Außerdem findet eine Drehung der Kanten während des Schrumpfens des Polygons in diesem Dreieck statt. Die Ergebnisse sind in den Abbildungen 3.9 und 3.11 dargestellt.

Beide Funktionen haben sehr ähnlichen Charakter und besitzen zwei Nullstellen im positiven x-Bereich. Abbildung 3.12 zeigt die Funktionen im Bereich der Nullstellen im Detail. Der Wert der Funktionen zwischen den Nullstellen hängt vom Abstand zwischen der Geraden und dem Punkt während der Kantendrehung ab. Die Funktion von Gleichung 2.25 hat viel geringere Werte in diesem Bereich als die Funktion von Gleichung 2.31. Dies ergibt sich

durch die geometrische Gegebenheit, dass die Fläche des Dreiecks auch bei geringem Abstand relativ große Werte aufweisen kann. Diese Eigenschaft in Kombination mit einem relativ kleinen Näherungsschritt macht das Überspringen der ersten Nullstelle während der Näherung sehr unwahrscheinlich. Außerdem weist die Funktion der Gleichung 2.31 immer höhere Steigungen auf als die Funktion von Gleichung 2.25 und eignet sie sich somit besser zur Bestimmung der Nullstellen mit Hilfe eines Näherungsverfahrens.

Ergebnis: Gleichung 2.31 wird in der Implementierung der Näherungsverfahren verwendet.

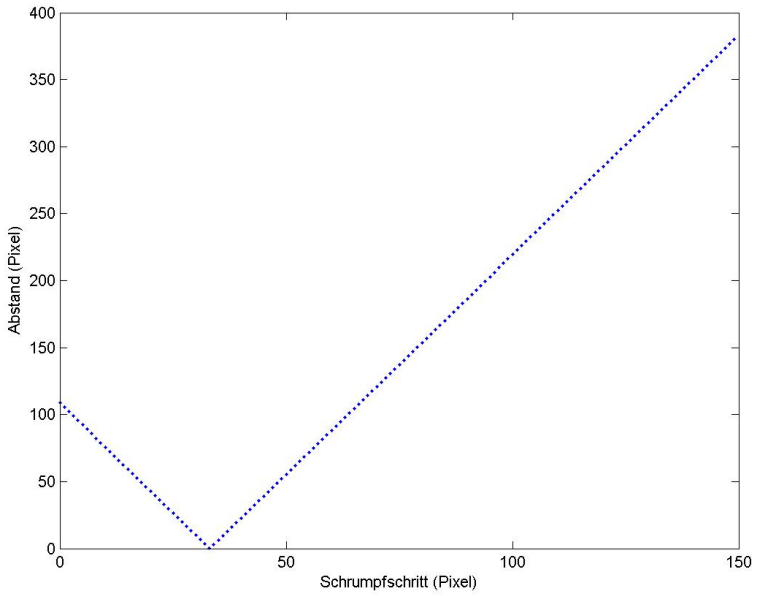


Abbildung 3.8: Dreiecksfläche in Abhängigkeit von Schrumpfschritt (1) Kanten des Dreiecks sind äußere Kanten des Polygons

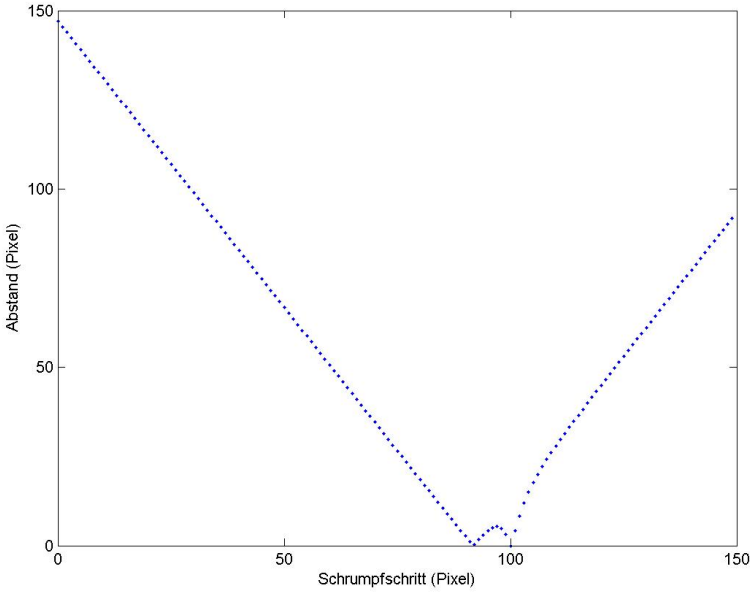


Abbildung 3.9: Dreiecksfläche in Abhängigkeit von Schrumpfschritt (2) Kanten des Dreiecks sind innere Kanten des Polygons

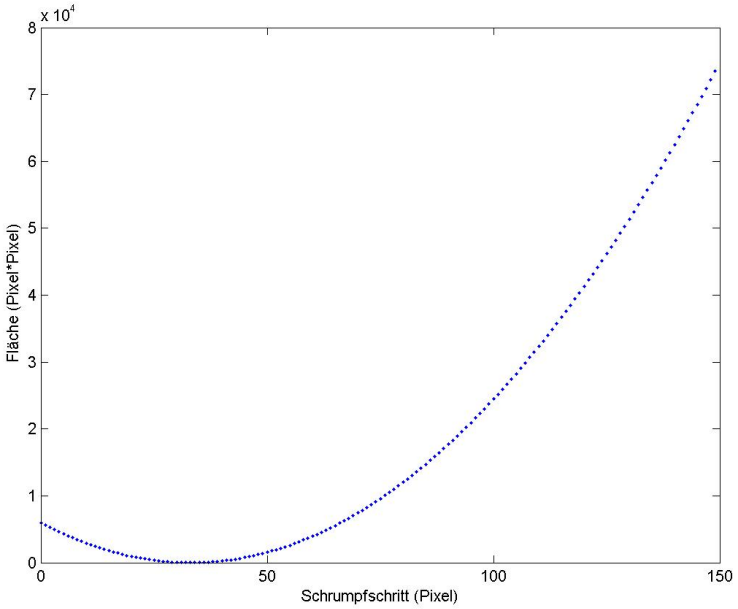


Abbildung 3.10: Abstand zwischen Gerade und Punkt des Dreiecks (1) Kanten des Dreiecks sind äußere Kanten des Polygons

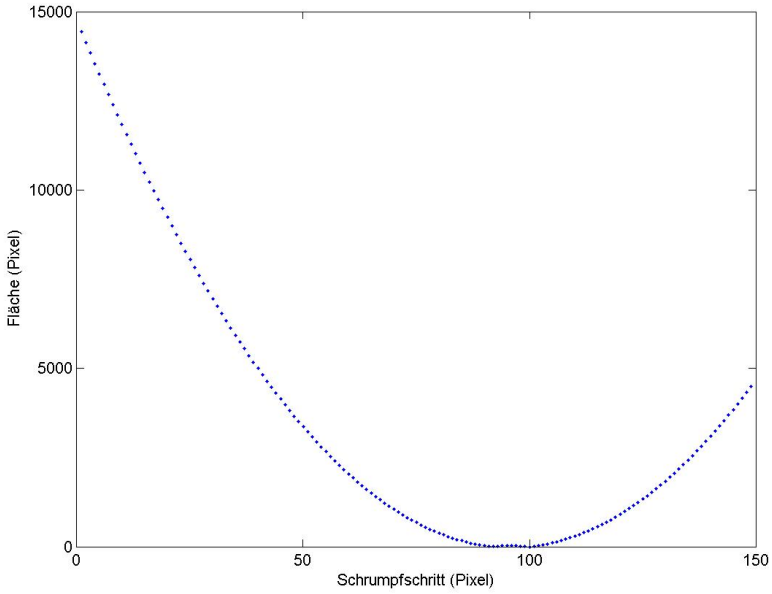


Abbildung 3.11: Abstand zwischen Gerade und Punkt des Dreiecks (2)
Kanten des Dreiecks sind innere Kanten des Polygons

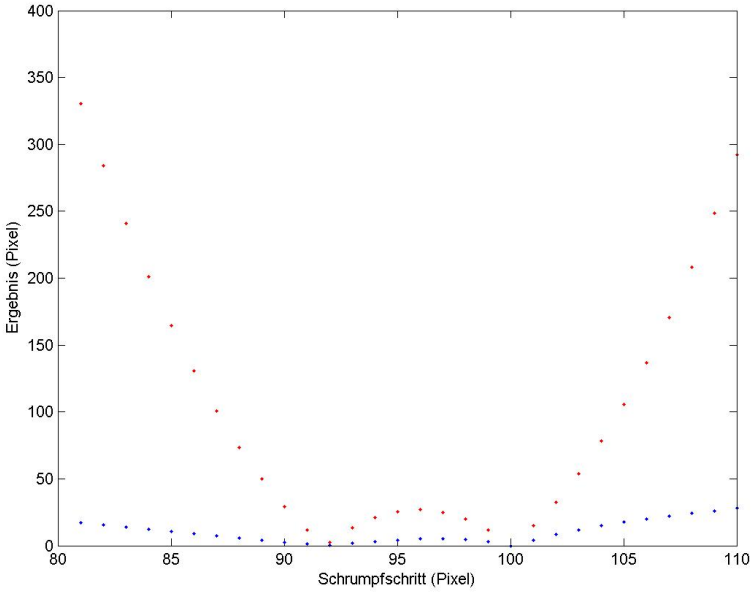


Abbildung 3.12: Vergleich der Funktionen 2 und 4
Kanten des Dreiecks sind innere Kanten des Polygons
Rot: Abstand zwischen der Kante und dem Punkt des Dreiecks
Blau: Fläche des Dreiecks

Implementierung des Näherungsverfahrens

Die meist verbreiteten Näherungsverfahren, welche mit nicht linearen Funktionen arbeiten (wie z.B. Gradientenabstieg oder das Newtonverfahren), basieren auf der Ableitung der Zielfunktion. Dieser Ansatz kann grundsätzlich auch für die ausgewählte Gleichung angewendet werden. Die ständige Änderung der Parameter der Gleichung in jedem Schritt (ein Parameter-Set pro Dreieck, bzw. 3 Parameter-Sets pro Pseudodreieck), würde immer wieder eine Neuberechnung der Ableitung verursachen, was sich bei so großen Gleichungen relativ schwierig gestaltet. Außerdem garantieren diese Methoden bei mehreren Minima nicht immer die Lösung mit dem kleinsten positivem x-Wert. Somit könnten einige Events übersprungen werden. Basierend auf dem Wissen über den Charakter der Zielfunktion wurde ein eigenes Näherungsverfahren implementiert, das die angesprochenen Problemstellen berücksichtigt.

Aufgabenstellung: Implementierung eines Näherungsverfahrens für die Suche der Nullstelle im positiven x-Bereich mit dem kleinsten x-Wert für die gewählte Gleichung.

Der gewählte Ansatz ist den klassischen Methoden sehr ähnlich. Man startet mit einem festgelegten Startwert und einer festgelegten Schrittgröße und nähert sich Schritt für Schritt der Nullstelle der Funktion. Da der Charakter der Funktion bekannt ist, ist auch die Suchrichtung von Anfang an bekannt. Die Suche startet mit dem Wert $x = 1$ und bewegt sich der x-Achse entlang in positive Richtung mit Schrittgröße S. In jedem Schritt wird der Wert der Gleichung berechnet und mit dem Wert des vorigen Schrittes verglichen. Ist der aktuelle Wert kleiner als der Wert vom vorherigen Schritt, findet ein Richtungswechsel statt und die Schrittgröße wird verkleinert. Die richtige Wahl der Größe des Schrittes ist sehr wichtig. Mit einer größeren Schrittweite sinkt die notwendige Anzahl von Iterationen, aber die Wahscheinlichkeit, dass bei der Suche nicht die Lösung mit dem kleinsten x-Wert gefunden wird, steigt. Ein relativ kleiner Schritt garantiert als Ergebnis die Lösung mit dem kleinsten x-Wert, dafür verschlechtert sich die Performance des Verfahrens. Bei der Wahl der Schrittgröße wurde berücksichtigt, dass die richtige Nullstelle für das richtige Ausführen des Algorithmus von enormer Wichtigkeit ist. Außerdem arbeitet der Prototyp mit relativ kleiner Fläche, somit bringt der Performanceverlust bei den Eventberechnungen kleine oder gar keine spürbaren Auswirkungen. Bei mehrfachen Tests fiel die Entscheidung auf Schrittgröße 0.6 Pixel. Zusätzlich wurde eine weitere Schrittverkleinerung eingebaut, sobald der Wert der Funktion kleiner als 30 Pixel * Pixel ist. Sie sorgt zusätzlich während der Suche nach einer Nullstelle dafür, dass die Nullstelle mit dem kleinsten x-Wert nicht übersprungen wird.

Einen weiterer Vorteil ist die Möglichkeit der Aufteilung der Gleichung in zwei Teile: die Berechnung der neuen Punkte und die Berechnung der Fläche des Dreiecks. Gleich nach der Berechnung der neuen Punkte kann ihre Lage in Bezug auf das Polygon geprüft werden. Somit wird die Berechnung der Fläche ausschließlich mit den drei Punkten innerhalb des Polygons durchgeführt und die Aufstellung von mehreren Gleichungen für jede mögliche Kombination, welche in Kapitel 3.2.2 beschrieben wurde, ist nicht mehr notwendig.

Zusammengefasst: der implementierte Algorithmus 8 sucht durch Annäherung die Nullstelle der Funktion im positiven x-Bereich mit dem kleinsten x-Wert für die gewählte Gleichung.

Algorithmus 8 Näherungsverfahren

```
1  /*Initialisierung*/
2  double initialStep = calculateSpaceOfTriangle(...);
3  double resultLastIteration = initialStep;
4  /*Näherung*/
5  Solange (true)
6      /*Aktualisiere die prüfbare Stelle*/
7      h = updateH(h, step, stepdirection);
8      result = calculateAreaOfTriangle(...);
9      /*Akzeptierte Nullstelle*/
10     if (0.0 < result ∧ result < 0.65) then
11         break;
12     end if
13     /*Keine Lösung gefunden*/
14     if (step < 0.0001)  then
15         break;
16     end if
17     /*Aktualisierung der Bewegungsrichtung und der Schrittgröße */
18     if (checkDirection(resultLastIteration, result)) then
19         stepdirection = changeDirection(stepdirection);
20         step = reduceStep(step);
21     end if
22     if (result < 30) then
23         step = reduceStep(step);
24     end if
25     resultLastIteration = result;

26 return h;
```

Kapitel 4

Evaluierung

In diesem Kapitel wird die Laufzeit von Algorithmus 5, welcher im Rahmen dieser Arbeit vorgestellt wurde und auf einer Pseudotriangulierung des Polygons basiert, evaluiert. Die Laufzeitanalyse für die Initialisierung kann ähnlich wie bei den Algorithmen 1 und 2 durchgeführt werden.

Initialisierung

- Die Erstellung der Pseudotriangulierung kann in $\mathcal{O}(n\ log(n))$ durchgeführt werden.
- Die Anzahl der Events entspricht zu diesem Zeitpunkt der Anzahl der Pseudodreiecke einer Pseudotriangulierung und beträgt im Worst Case $\mathcal{O}(n)$ (d.h die Pseudotriangulierung ist eine Triangulierung).
- Die Sortierung der Eventliste kann in $\mathcal{O}(n\ log(n))$ durchgeführt werden.
- Das Lösen der Gleichungen 2-4 wird nicht in die Laufzeitabschätzung miteinbezogen.

Somit kann die Initialisierung in $\mathcal{O}(n\ log(n))$ durchgeführt werden.

Schritt

Die Laufzeitanalyse des Schrittes gestaltet sich jedoch sehr schwierig, da die Anzahl der Wiederholungen nicht bekannt ist. Im Worst Case gibt es keine Verbesserungen im Vergleich zu Algorithmus 2. Außerdem gibt es keine Informationen über die mögliche Anzahl von Split Events in einem Pseudodreieck. Man darf auch nicht außer Acht lassen, dass unterschiedliche Triangulierungen bzw. Pseudotriangulierungen für ein Polygon erstellt werden können. Die

Wahl anderer Triangulierungen bzw. Pseudotriangulierungen kann somit sowohl positive, als auch negative Auswirkungen auf die Anzahl der Schritte bei der Erstellung eines Straight Skeletons haben. Somit ist eine formale Zeitabschätzung der Anzahl an Wiederholungen des Schrittes von Algorithmus 5 leider nicht möglich.

Um einen Eindruck über die Laufzeit von Algorithmus 5 zu bekommen wurde ein praktischer Ansatz gewählt, bei dem mit einem selbst entwickelten Generator zufällige, nicht konvexe Polygone vorgegebener Größe (Anzahl der Punkte) erstellt werden. Für die generierten Polygone wurden mit den Algorithmen 5 und 2 die Straight Skeletons erstellt. Im Initialisierungsschritt von Algorithmus 2 wurde einmal eine beliebige Triangulierung und einmal eine Delaunay Triangulierung verwendet, um einen Eindruck davon zu bekommen, wie sich die Art der Triangulierung auf die Laufzeit auswirkt. Aus den Daten, welche bei der Erstellung des Straight Skeletons gewonnen wurden, wurde die durchschnittliche Anzahl der benötigten Schritte pro Algorithmus für ein Polygon gegebener Größe ermittelt. Bei der Zählung der Schritte wurde nicht darauf geachtet, ob mehrere Ereignisse zur gleichen Zeit auftreten. Es wurden nur die Anzahl der gemachten Schrumpfschritte betrachtet. Die Ergebnisse sind in Abbildung 4.1 dargestellt.

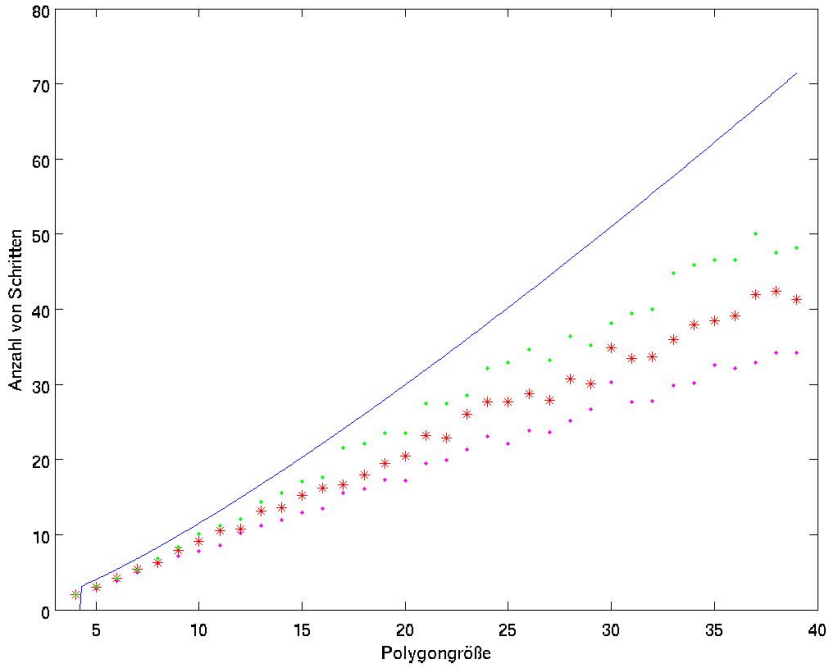


Abbildung 4.1: Evaluierung von Algorithmus 5

Legende:
Blau: Funktion $y = (n * log(n))/2$
Grün: Benötigte Schritte von Algorithmus 2 (Beliebige Triangulierung)
Purpur: Benötigte Schritte von Algorithmus 2 (Delaunay Triangulierung)
Rot: Benötigte Schritte von Algorithmus 5 (Balancierte Pseudotriangulierung)

Wie in Abbildung 4.1 ersichtlich, liegt die durchschnittliche Anzahl an benötigten Schritten bei allen Algorithmen deutlich unter $\mathcal{O}(n\ log(n))$, der oberen Schranke für Algorithmus 2 in der Durchschnittssituation. Wie bereits erwähnt, wurden bei der Evaluierung die benötigten Schritte des Algorithmus und nicht die Events gezählt. Bei der Zählung der Events könnten die Ergebnisse im Vergleich zu den vorgelegten Ergebnissen etwas nach oben abweichen. Die Abweichung wurde in diesem Fall vernachlässigt, da die Wahrscheinlichkeit für gleichzeitige Events in einem Polygon, welches mittels eines Zufallsgenerators erstellt wurde, sehr gering ist. Außerdem würden solche Events bei allen Algorithmen gleichzeitig auftreten und das Verhältnis zwischen den Ergebnissen der einzelnen Algorithmen nicht beeinflussen.

Wie bereits angedeutet ist Algorithmus 2 zweimal in den Ergebnissen aufge-

führt: einmal basierend auf einer beliebigen Triangulierung und einmal basierend auf einer Delaunay Triangulierung. Bei kleinen Punktmengen zeigen alle drei Algorithmen fast dieselben Ergebnisse. Mit steigender Polygongröße (ca. 10) können deutliche Unterschiede zwischen den einzelnen Algorithmen beobachtet werden. Der Algorithmus 2 in der Variante mit der Delaunay Triangulierung benötigt am wenigsten Schritte. Danach folgt der Algorithmus 5 mit einer balancierten Pseudotrianglulierung. Die schlechtesten Ergebnisse weist Algorithmus 2 mit einer beliebigen Triangulierung auf.

Ein weiterer Vergleich der Schrittanzahl, welche jeder Algorithmus benötigt, basierend auf einem Polygon der Größe 100 ist in Abbildung 4.2 dargestellt.

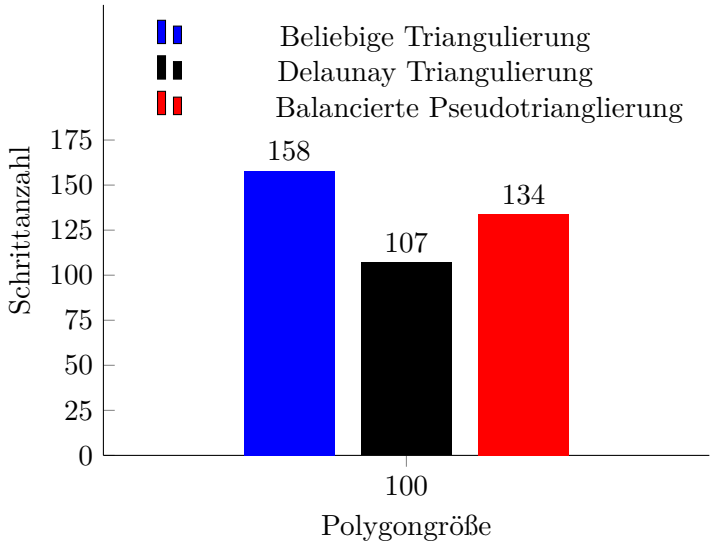
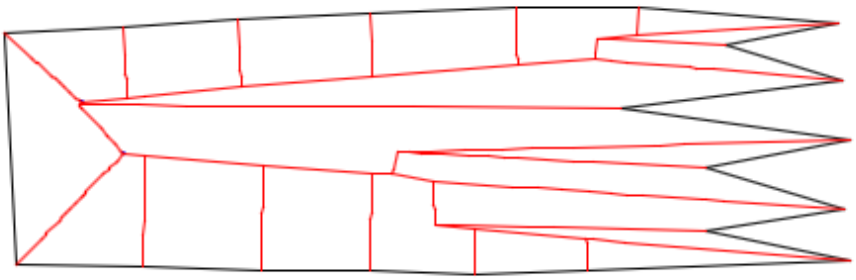


Abbildung 4.2: Evaluierung von Algorithmus 5 mit einem Polygon Größe 100

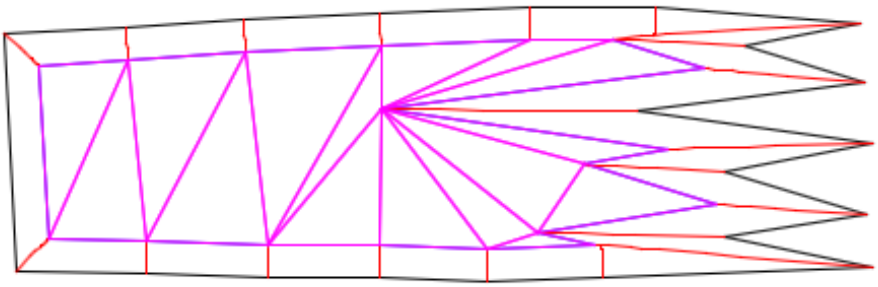
Dies kann auf die Balancierung der Delaunay-Triangulierung und der Pseudotrianglulierung zurück geführt werden. Durch diese Eigenschaft wird die Anzahl der Events reduziert, welche keine Änderungen in der Struktur des Straight Skeletons auslösen und somit die durchschnittliche Laufzeit der Algorithmen verbessert.

Obwohl Algorithmus 2 in der Variante mit der Delaunay Triangulierung in der Evaluierung die besten Ergebnisse gezeigt hat, gibt es durchaus die Polygone für welche sich Algorithmus 5 mit einer balancierten Pseudotriangulierung sich deutlich besser eignet. Ein Beispiel dafür ist in Abbildung 4.3a dargestellt. Die inneren spitzen Ecken des Polygons schneiden die Kanten der Triangulierung von recht nach links durch und lösen dabei die Split-Events

aus (siehe Abbildung 4.3b). Algorithmus 2 mit Delaunay Triangulierung benötigt für den Aufbau des Straight Skeletons von diesem Polygon 33 Schritte. Algorithmus 5 mit einer balancierten Pseudotrianglulierung baut es bereits in 25 Schritten auf. Der Grund dafür ist die besondere Form des Polygons, welche die Anzahl der Flip-Events maximiert. Bei der Generierung von zufälligen konvexen Polygonen kommen Formen dieser Art mit sehr geringer Wahrscheinlichkeit vor und was sich auch in der Evaluierung wiederspiegelt.



(a)



(b)

Abbildung 4.3: Polygon mit vielen Flip-Events

Legende:
Schwarz: Linien des Basis-Polygons
Rot: Linien des Straight Skeletons
Purpur: Linien der Triangulierung im aktuellen Schritt

Alle drei angeführten Algorithmen haben im Durchschnitt eine Laufzeit von $\mathcal{O}(n\ log(n))$.

Kapitel 5

Andere Arbeiten zu Straight Skeletons

In diesem Kapitel werden einige verwandte Arbeiten zum Thema Straight Skeletons betrachtet bzw. vorgestellt.

In "Computing Straight Skeletons of Planar Straight-Line Graphs Based on Motorcycle Graphs"[12] und "Motorcycle Graphs and Straight Skeletons"[6] beschäftigen sich die Autoren mit der geometrischen Struktur Motorcycle Graph, welche mit Straight Skeletons verwandt ist. Zur Erstellung dieses Graphen wählt man einen Punkt in der Ebene, der mit konstanter Geschwindigkeit entlang eines geraden Strahls in eine vorgegebene Richtung bewegt wird. Dieser Punkt, welcher in diesem Kontext auch Motorcycle genannt wird, hinterlässt bei seiner Bewegung einen Pfad, welcher von anderen Motorcycles nicht überquert werden darf. Die Menge all dieser Pfade bildet den Motorcycle Graph. Dieser Ansatz lässt sich sehr leicht auf Straight Skeletons übertragen. Die Punkte des Polygons werden zu Motorcycles. Die Richtung jedes Motorcycles wird durch die Winkelhalbierende und die Geschwindigkeit durch die Winkelgröße definiert. Nach dem Zusammenstoß von zwei oder mehreren Motorcycles (entspricht einem Ereignis) werden Richtung und Geschwindigkeit eines Motorcycles neu definiert, die verbleibenden, am Zusammenstoß beteiligten Motorcycles, bewegen sich nicht weiter. Auch unterschiedliche Schrumpfgeschwindigkeiten von einzelnen Kanten können durch Anpassung der Richtung und Geschwindigkeit abgebildet werden. Die Verwandtschaft von Motorcycle Graphs und Straight Skeletons ermöglicht das Übertragen gewisser Eigenschaften bzw. Algorithmen auf die jeweils andere Datenstruktur. Die Autoren stellen ein Algorithmus zur Erstellung von Straight Skeletons mit Laufzeit $\mathcal{O}(n\ \sqrt{n}\ log(n))$ vor. Dieses Problem wird in $\mathcal{O}(n\ log^2(n))$ auf die Berechnung von Motorcycle Graphs reduziert. Durch

die Kombination der Ergebnisse entsteht ein Algorithmus, welcher in der Lage ist in $\mathcal{O}(n\ log^2(n) + r\ \sqrt{r}\ log(r))$ für degenerative Polygone und in $\mathcal{O}(n\ log^2(n) + r^{(17/11+\epsilon)})$ für nicht degenerative Polygone ein Straight Skeleton zu erstellen. n bezeichnet hierbei die Anzahl der Kanten des Polygons und r die Anzahl der Kanten des Straight Skeletons.

Die Autoren von "Computing the straight skeleton of a monotone polygon in $\mathcal{O}(n\ log(n))$ time"[8] beschäftigen sich mit monotonen Polygonen. Das ist eine Gruppe von Polygonen, die durch folgende Eigenschaften definiert werden. Ein Polygon heißt monoton im Bezug auf eine Gerade g bzw. eine Richtung, wenn es mit einer Geraden in zwei monotone Ketten zerlegbar ist. Schneidet man eine dieser Ketten mit einer auf g normal stehenden Geraden h, so ergibt sich immer nur ein Schnittpunkt. Man könnte auch sagen, dass h immer nur eine Kante der monotonen Kette schneidet. Nimmt man als Gerade g z.B. die x-Achse, dann sind die Punkte beider Ketten aufsteigend nach ihrer x-Koordinate sortiert. Diese besondere Eigenschaft kann in diversen Algorithmen genutzt werden um die Laufzeiten zu verbessern. In [8] wird ein effizienter und einfacher Algorithmus zur Erstellung von Straight Skeletons für diese geometrische Struktur vorgestellt. Die Laufzeit für die Berechnung aller Edge Events beträgt bei diesem Algorithmus, gleich wie bei Algorithmus 1, $\mathcal{O}(n)$. Die Berechnung von Split Events kann jedoch bereits in $\mathcal{O}(n\ log(n))$ durchgeführt werden, was eine deutliche Verbesserung darstellt. Für die Abarbeitung der Events werden maximal $\mathcal{O}(n)$ Schritte benötigt. Somit ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(n\ log(n))$ und ein Speicherbedarf von $\mathcal{O}(n)$.

In der Arbeit "Straight Skeleton Implementation"[10] befassen sich die Autoren mit der Implemetierung eines Algorithmus zur Erstellung von Straight Skeletons. Es werden zwei Algorithmen vorgestellt: einer für konvexe und einer für konkave Polygone. Auch für Polygone mit Löchern kann ein Straight Skeleton erstellt werden. Ein wichtiger Teil in der Implementierung sind die Datenstrukturen LAV (list of active vertices) und SLAV (set of circular list of active vertices), die jeweils in dem Algorithmus für konvexe bzw. konkave Polygone verwendet wurden. Diese Arbeit beinhaltet eine detaillierte Beschreibung spezieller Situationen, z.B. mehrere Split Events zur gleichen Zeit. Die Berechnung des Events basiert, gleich wie bei Algorithmus 1, auf den Winkelhalbierenden der Punkte eines Polygons. Die Laufzeit des Programms wurde über Tests mit Polygonen verschiedener Größe verifiziert und beträgt $\mathcal{O}(n^2)$ mit $\mathcal{O}(n)$ Speicherbedarf, wobei n der Anzahl der Kanten des Polygons entspricht.

Kapitel 6

Zusammenfassung und Ausblick

In dieser Arbeit wurden zwei bereits bekannte Algorithmen zur Erstellung von Straight Skeletons beschrieben, welche auf den Winkelhalbierenden, sowie einer Triangulierung des Polygons basieren. Weiters wurde ein neuer Algorithmus entwickelt, welcher mittels Pseudotriangulierung des Polygons arbeitet.
Für die Visualisierung der Arbeitsweise der einzelnen Algorithmen wurde ein Programm mit einer grafischen Benutzeroberfläche implementiert. Bei der Implementierung des Programms wurden kleinere Änderungen an den Algorithmen vorgenommen, welche dem Programm zusätzliche Flexibilität bei der Erstellung von Straight Skeletons geben. Für die Event-Berechnung in Algorithmen, basierend auf einer Triangulierung bzw. einer Pseudotriangulierung wurde ein Näherungsverfahren entwickelt und implementiert.
Abschließend wurde eine Evaluierung der Algorithmen durchgeführt. Bem direkten Vergleich der benötigten Schritte wurde festgestellt, dass die geringste Schrittanzahl von Algorithmus 2 in der Kombination mit der Delaunay Triangulierung und die größte Schrittanzahl vom Algorithmus 2 in der Kombination mit einer beliebigen Triangulierung benötigt werden. Die Schrittanzahl von Algorithmus 5 liegt zwischen der Schrittanzahl der vorher genannten Algorithmen. Die obere Schranke für eine Durchschnittssituation liegt somit für alle drei Algorithmen bei $\mathcal{O}(n\ log(n))$.

6.1 Ausblick

In der Testphase des Programms wurde beobachtet, dass die Polygone in Bezug auf Algorithmus 2 in der Variante mit Delaunay Triangulierung und Algorithmus 5 in drei Gruppen unterteilt werden können: eine Gruppe, in welcher Algorithmus 2 schneller als Algorithmus 5 ist, eine Gruppe mit der umgekehrten Eigenschaften und eine Gruppe, in welcher die beiden Algorithmen gleich viele Schritte benötigen. Daraus folgt, dass die beiden Algorithmen in bestimmten Situationen einige Schritte durchführen, welche als Zwischenschritte angesehen werden können die für den Aufbau von Straight Skeletons nicht relevant sind. Durch die Kombination der Vorteile beider Algorithmen könnte eine Verbesserung der Laufzeit erreicht werden. Eine Delaunay Triangulierung und eine Pseudotriangulierung können in $\mathcal{O}(n\ log(n))$ aufgebaut werden, somit bleibt die Laufzeit für die Initialisierung des Algorithmus unverändert. Bei der Wahl des nächsten Schrittes würde jeder Ansatz ein Event mit der kleinsten Schrittgröße liefern (ein Event wird nach Delaunay Triangulierung und das andere Event wird nach Pseudotriangulierung berechnet). Für die Ausführung des Schrittes wird in diesem Fall der größere der beiden Schritte gewählt. Durch diese Vereinigung werden alle Schritte, welche eine Änderung in der Struktur des Straight Skeletons verursachen und in beiden Algorithmen zur gleichen Zeit auftreten, auf jeden Fall durchgeführt. Einige Zwischenschritte, welche keine Änderungen in der Struktur des Straight Skeletons verursachen und nur in einem Algorithmus auftreten, könnten somit übersprungen werden. Dadurch sollte sich für die Kombination der Algorithmen eine etwas bessere Laufzeit ergeben.

Literaturverzeichnis

[1] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In J.-Y. Cai and C. Wong, editors, *Computing and Combinatorics*, volume 1090 of *Lecture Notes in Computer Science*, pages 117–126. Springer Berlin Heidelberg, 1996.

[2] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. A novel type of skeleton for polygons. *Journal of Universal Computer Science*, 1(12):752–761, dec 1995. `http://www.jucs.org/jucs_1_12/a_novel_type_of`.

[3] G. Barequet, M. T. Goodrich, A. Levi-Steiner, and D. Steiner. Straight-skeleton based contour interpolation. In *SODA*, pages 119–127. ACM/-SIAM, 2003.

[4] J. Bieling. Pseudodreiecks-Zerlegung. Technical report, Rheinische Friedrich-Wilhelms-Universität Bonn, Juni 2008.

[5] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[6] S.-W. Cheng and A. Vigneron. Motorcycle graphs and straight skeletons. *Algorithmica*, 47(2):159–182, 2007.

[7] L. P. Chew. Constrained delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[8] G. K. Das, A. Mukhopadhyay, S. C. Nandy, S. Patil, and S. V. Rao. Computing the straight skeleton of a monotone polygon in o(n log n) time. In *CCCG*, pages 207–210, 2010.

[9] S. Eckelmann. *Triangulierung eines Polygons in 2D.* Technische Universität Chemnitz.

[10] P. Felker and S. Obdrzalek. Straight skeleton implementation. *Proceedings of spring conference on computer graphics*, pages 210–218, 1998.

[11] R. Görke. Ein schneller konstruktionsalgorithmus für eine quickest-path-map bezüglich der city-metrik. Master's thesis, Master's thesis, Fakultät für Informatik, Universität Karlsruhe, 2004.

[12] S. Huber and M. Held. Computing straight skeletons of planar straight-line graphs based on motorcycle graphs. In *CCCG*, pages 187–190, 2010.

[13] S. Janusch. Konzeption & Realisierung eines prozeduralen Ansatzes zur Erzeugung von Gebäuden. Master's thesis, Hochschule Darmstadt, 2007.

[14] R. Klein. *Algorithmische Geometrie*, volume 392. Springer.de, 1997.

[15] H.-D. Meyerhenke, Henning und Hecker. Parallele algorithmische geometrie anhand von delaunay-triangulationen1. Mai 2005.

[16] P. Palfrader, M. Held, and S. Huber. On computing straight skeletons by means of kinetic triangulations. In L. Epstein and P. Ferragina, editors, *Algorithms – ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 766–777. Springer Berlin Heidelberg, 2012.

[17] M. Pocchiola and G. Vegter. The visibility complex. *International Journal of Computational Geometry & Applications*, 06(03):279–308, 1996.

[18] G. Rote, F. Santos, and I. Streinu. Pseudo-Triangulations-a survey. *Contemporary Mathematics*, 453:343–410, 2008.

[19] C. Ullenboom. *Java ist auch eine Insel (10. Auflage)*. Galileo Computimg, http://openbook.galileocomputing.de/javainsel/.

[20] E. Yakersberg. Morphing between geometric shapes using straight-skeleton-based interpolation. Master's thesis, Department of Computer Science, Technion, Haifa, Israel, 2004.